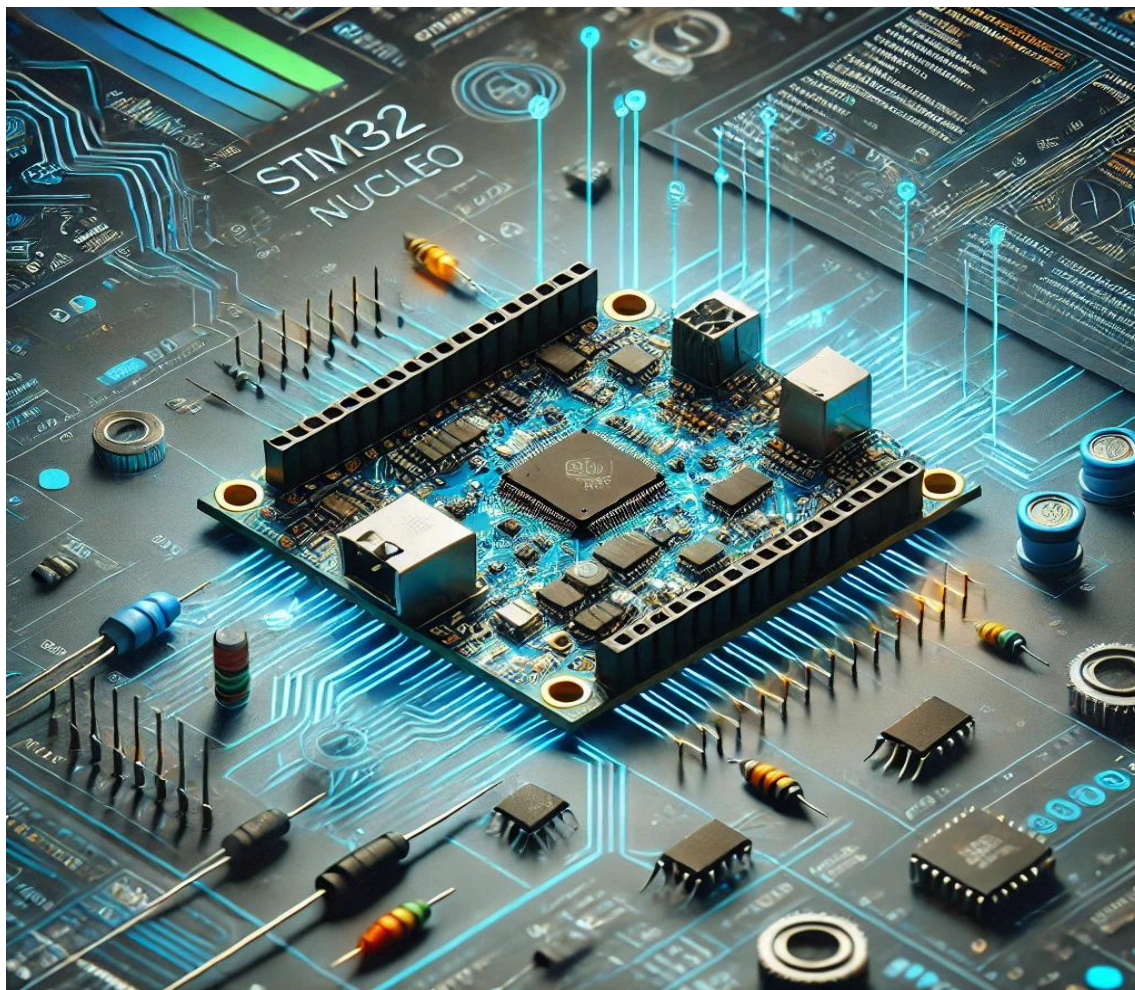


CREACION DE DRIVERS EN STM32CUBE IDE



Curso: Técnicas Digitales II

Profesores: Ing. Rubén Darío Mansilla, Ing. Lucas Abdala

Integrantes:

- Barrientos Lucas
- Cuellar Agustin
- Vera Monasterio Candela

Fecha de entrega: Septiembre 2024

5. Informe

5.1 Introducción

El uso de sistemas embebidos se ha expandido en diversas aplicaciones tecnológicas. Este informe se centra en el desarrollo de un driver General Purpose Input/Output (GPIO) para facilitar la comunicación entre un microcontrolador y dispositivos periféricos.

Objetivo general

El objetivo es desarrollar e implementar un driver GPIO en un sistema embebido, optimizando la gestión de entradas y salidas digitales.

Importancia de utilizar drivers en aplicaciones embebidas

Los drivers son esenciales en sistemas embebidos, ya que actúan como intermediarios entre el software y el hardware. Facilitan la abstracción del hardware, mejoran la modularidad del código y permiten un desarrollo más eficiente, asegurando una comunicación efectiva con los dispositivos conectados.

Descripción del Proceso

Desarrollo del Driver GPIO

Paso 1: Selección de la aplicación base

Para este proyecto, se eligió una aplicación sencilla pero funcional que involucra el control de tres LEDs y un pulsador en la placa **STM32 Nucleo F429ZI**. La aplicación consiste en encender y apagar de forma secuencial los LEDs conectados a los pines **GPIO** de la placa. Además, la dirección de la secuencia de encendido puede invertirse al presionar un botón.

Paso 2: Creación del driver

Se creó un nuevo proyecto aparte en **STM32Cube IDE** específicamente para implementar el driver GPIO personalizado. El proceso siguió los pasos definidos en la guía de creación de drivers, donde primero:

1. En el directorio de **Drivers**, se creó una carpeta llamada **API** para contener las funciones del nuevo driver.
2. Dentro de la carpeta API, se crearon dos **subdirectorios**:
 - **Src**: Para almacenar el archivo fuente del driver (**API_GPIO.c**).
 - **Inc**: Para almacenar el archivo de cabecera del driver (**API_GPIO.h**).

En estos archivos, se desarrollaron las funciones personalizadas para controlar los pines GPIO, reemplazando las funciones HAL por nuevas funciones que abstraen el manejo de LEDs y el botón.

Paso 3: Implementación de funciones

El driver personalizado incluye las siguientes funciones:

- **MX_GPIO_Init()**: Inicializa los pines GPIO que corresponden a los LEDs y el botón.
- **LED_On(led_T led)**: Enciende uno de los LEDs especificados (LD1, LD2 o LD3).
- **LED_Off(led_T led)**: Apaga uno de los LEDs especificados.
- **readButton_GPIO()**: Lee el estado del botón conectado al pin PC13.

Estas nuevas funciones reemplazaron las funciones originales de HAL para hacer más claro y sencillo el control de los LEDs y el botón:

- **HAL_GPIO_WritePin()** fue reemplazada por **LED_On()** y **LED_Off()**.
- **HAL_GPIO_ReadPin()** fue reemplazada por **readButton_GPIO()**.
- **MX_GPIO_Init()** se mantiene en el driver, pero ahora controla todos los pines relevantes (LEDs y botón).

Para probar el correcto funcionamiento del driver, se implementaron las nuevas funciones en un archivo `main.c`. El código controla tres LEDs (conectados a **PB0**, **PB7** y **PB14**) que se encienden y apagan secuencialmente con un retardo de 200 ms entre cada LED. Además, al presionar un botón (conectado a PC13), se invierte la dirección de la secuencia. Esto permitió verificar que las funciones del driver para encender, apagar y leer el estado del botón funcionaban como se esperaba.

5.2 Aplicaciones desarrolladas

Modificación de las aplicaciones

Aplicacion 1

La única acción inicial fue copiar los archivos de Drivers/API y pegarlos en la carpeta Drivers del nuevo proyecto (App_1_1_grupo_6_2024). A continuación, modificamos el archivo `main.c`, cambiando las funciones antiguas de la HAL por las nuevas funciones creadas en nuestro driver personalizado.

El driver incluye las siguientes funciones:

- `MX_GPIO_Init()`: Inicializa los pines GPIO que corresponden a los LEDs y al botón.
- `LED_On(led_T led)`: Enciende uno de los LEDs especificados (LD1, LD2 o LD3).
- `LED_Off(led_T led)`: Apaga uno de los LEDs especificados.
- `readButton_GPIO()`: Lee el estado del botón conectado al pin PC13.

Estas nuevas funciones reemplazan las funciones originales de HAL, facilitando el control de los LEDs y el botón. En concreto, `HAL_GPIO_WritePin()` fue reemplazada por `LED_On()` y `LED_Off()`, mientras que `HAL_GPIO_ReadPin()` fue sustituida por `readButton_GPIO()`. La función `MX_GPIO_Init()` se mantiene en el driver, pero ahora controla todos los pines relevantes (LEDs y botón).

Lo mismo para las apps 2, 3 y 4.

Autor de cada aplicación:

- **Aplicación 1.1:** Cuellar Agustin
- **Aplicación 1.2:** Cuellar Agustin
- **Aplicación 1.3:** Vera Monasterio Candela
- **Aplicación 1.4:** Barrientos Lucas

Observaciones:

Dificultades encontradas

En esta fase del proyecto, realizamos el proceso de compilación y depuración para la aplicación 1. Durante el proceso de **compilación**, encontramos un **error**: el compilador no podía encontrar la ruta del driver que contenía la carpeta API. Para resolver este problema, consultamos una IA, que nos indicó los siguientes pasos:

1. Ir al menú "Project" y seleccionar "Properties".
2. En la ventana de propiedades, navegar a "C/C++ General > Paths and Symbols".
3. Seleccionar la pestaña "Includes".
4. Asegurarse de que la ruta Drivers/API/Inc esté agregada para el compilador. Esto le indicará al compilador que busque el archivo de cabecera en esa carpeta.
5. Agregar la ruta completa de Drivers/API/Inc en la sección del compilador C.

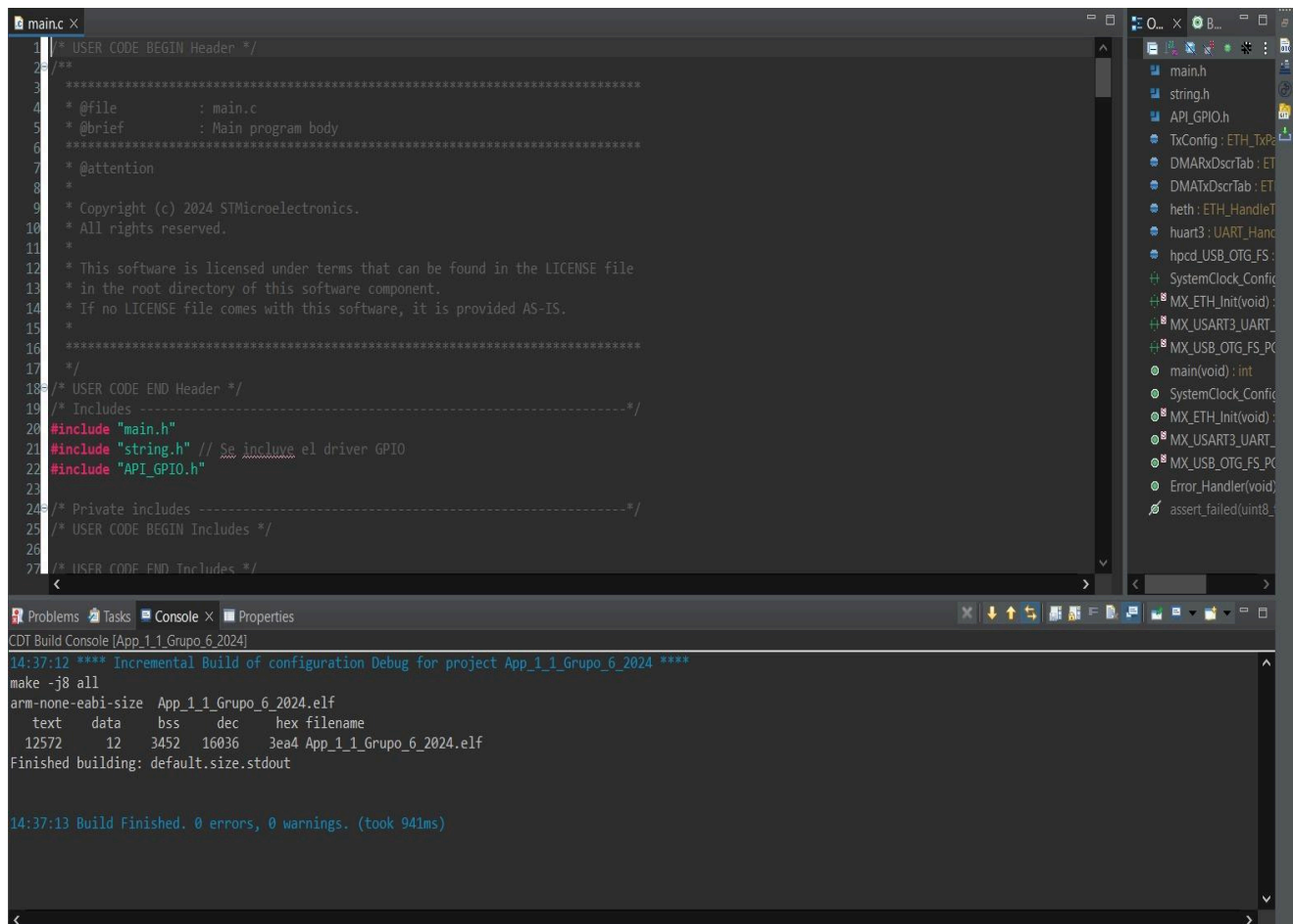
Después de realizar estos ajustes, el programa funcionó correctamente. Lo probamos en la placa y todo funcionó de maravilla.

5.3 Link al repositorio grupal

https://github.com/codecuellar/Grupo_6_TDII_2024

En las siguientes capturas de pantalla podemos observar como queda cada aplicación una vez finalizada. Los problemas que se presentaron fueron solucionados después de agregar las direcciones que nos recomendó la IA haciendo que se pueda compilar correctamente, sin ningún error.

App 1.1



The screenshot displays an IDE with two main panels. The top panel shows the source code of `main.c`, which includes a header section with copyright information for 2024 STMicroelectronics and a main function. The code includes `main.h`, `string.h`, and `API_GPIO.h`. The bottom panel shows the CDT Build Console output for the project `App_1_1_Grupo_6_2024`. The output indicates a successful incremental build with 0 errors and 0 warnings, taking 941ms.

```
1  /* USER CODE BEGIN Header */
2  /**
3   * @file          : main.c
4   * @brief         : Main program body
5   * @attention
6   *
7   * Copyright (c) 2024 STMicroelectronics.
8   * All rights reserved.
9   *
10  * This software is licensed under terms that can be found in the LICENSE file
11  * in the root directory of this software component.
12  * If no LICENSE file comes with this software, it is provided AS-IS.
13  *
14  *
15  */
16  /* USER CODE END Header */
17
18  /* Includes -----*/
19  #include "main.h"
20  #include "string.h" // Se incluye el driver GPIO
21  #include "API_GPIO.h"
22
23  /* Private includes -----*/
24
25  /* USER CODE BEGIN Includes */
26
27  /* USER CODE END Includes */
```

```
CDT Build Console [App_1_1_Grupo_6_2024]
14:37:12 **** Incremental Build of configuration Debug for project App_1_1_Grupo_6_2024 ****
make -j8 all
arm-none-eabi-size App_1_1_Grupo_6_2024.elf
text    data    bss     dec     hex filename
12572   12      3452   16036   3ea4 App_1_1_Grupo_6_2024.elf
Finished building: default.size.stdout

14:37:13 Build Finished. 0 errors, 0 warnings. (took 941ms)
```

App 1.2

The screenshot shows an IDE with a C source file named `main.c` and a build console window. The source code includes headers, defines private variables, and includes the `API_GPIO.h` driver. The console output shows the build process for `App_1_2_Grupo_6_2024.elf`, including a memory layout table and a successful build message.

```
1 /* USER CODE BEGIN Header */
2 /**
3  * @file          : main.c
4  * @brief         : Main program body
5  * @attention
6  *
7  * Copyright (c) 2024 STMicroelectronics.
8  * All rights reserved.
9  *
10 * This software is licensed under terms that can be found in the LICENSE file
11 * in the root directory of this software component.
12 * If no LICENSE file comes with this software, it is provided AS-IS.
13 *
14 *
15 */
16 /* USER CODE END Header */
17
18 /* Includes */
19 #include "main.h"
20 #include "API_GPIO.h" // See include el driver GPIO
21 #include "string.h"
22
23 /* Private variables */
24 ETH_HandleTypeDef heth;
25 UART_HandleTypeDef huart3;
26
```

CDT Build Console [App_1_2_Grupo_6_2024]

```
arm-none-eabi-size App_1_2_Grupo_6_2024.elf
arm-none-eabi-objdump -h -S App_1_2_Grupo_6_2024.elf > "App_1_2_Grupo_6_2024.list"
text    data    bss    dec    hex filename
6452    20      1580    8052    1f74 App_1_2_Grupo_6_2024.elf
Finished building: default.size.stdout

Finished building: App_1_2_Grupo_6_2024.list

02:28:52 Build Finished. 0 errors, 0 warnings. (took 23s.107ms)
```

App 1.3

The screenshot shows an IDE with a C source file named `main.c` and a build console window. The source code includes headers, defines private variables, and includes the `API_GPIO.h` driver. The console output shows the build process for `App_1_3_Grupo_6_2024.elf`, including a memory layout table and a successful build message.

```
1 /* USER CODE BEGIN Header */
2 /**
3  * @file          : main.c
4  * @brief         : Main program body
5  * @attention
6  *
7  * Copyright (c) 2024 STMicroelectronics.
8  * All rights reserved.
9  *
10 * This software is licensed under terms that can be found in the LICENSE file
11 * in the root directory of this software component.
12 * If no LICENSE file comes with this software, it is provided AS-IS.
13 *
14 *
15 */
16 /* USER CODE END Header */
17
18 /* Includes */
19 #include "main.h"
20 #include "API_GPIO.h" // Include el driver GPIO
21 #include "string.h"
22
23 /* Private includes */
24
25 /* USER CODE BEGIN Includes */
26
27 /* USER CODE END Includes */
28
```

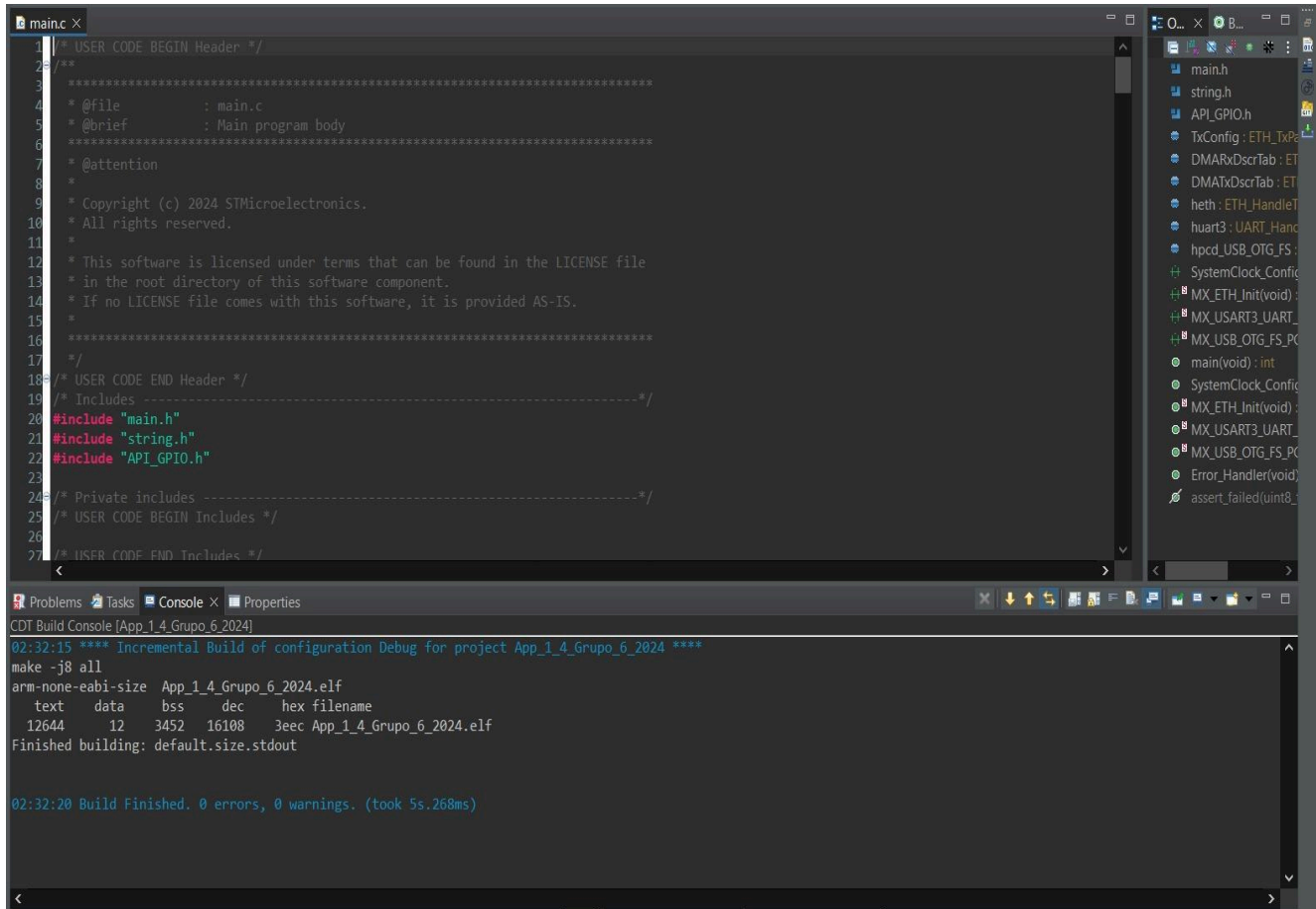
CDT Build Console [App_1_3_Grupo_6_2024]

```
arm-none-eabi-size App_1_3_Grupo_6_2024.elf
arm-none-eabi-objdump -h -S App_1_3_Grupo_6_2024.elf > "App_1_3_Grupo_6_2024.list"
text    data    bss    dec    hex filename
13136    16      3144    16296    3fa8 App_1_3_Grupo_6_2024.elf
Finished building: default.size.stdout

Finished building: App_1_3_Grupo_6_2024.list

01:16:35 Build Finished. 0 errors, 0 warnings. (took 1s.888ms)
```

App 1.4



The screenshot displays an IDE with a C source file named `main.c` and a terminal window showing the build process.

main.c Content:

```
1  /* USER CODE BEGIN Header */
2  /**
3   *
4   * @file      : main.c
5   * @brief     : Main program body
6   *
7   * @attention
8   *
9   * Copyright (c) 2024 STMicroelectronics.
10  * All rights reserved.
11  *
12  * This software is licensed under terms that can be found in the LICENSE file
13  * in the root directory of this software component.
14  * If no LICENSE file comes with this software, it is provided AS-IS.
15  *
16  */
17  /* USER CODE END Header */
18
19  /* Includes */
20  #include "main.h"
21  #include "string.h"
22  #include "API_GPIO.h"
23
24  /* Private includes */
25  /* USER CODE BEGIN Includes */
26
27  /* USER CODE END Includes */
```

Build Console Output:

```
CDT Build Console [App_1.4.Grupo_6_2024]
02:32:15 **** Incremental Build of configuration Debug for project App_1.4.Grupo_6_2024 ****
make -j8 all
arm-none-eabi-size App_1.4.Grupo_6_2024.elf
text  data  bss  dec  hex filename
12644   12  3452 16108  3eec App_1.4.Grupo_6_2024.elf
Finished building: default.size.stdout

02:32:20 Build Finished. 0 errors, 0 warnings. (took 5s.268ms)
```