

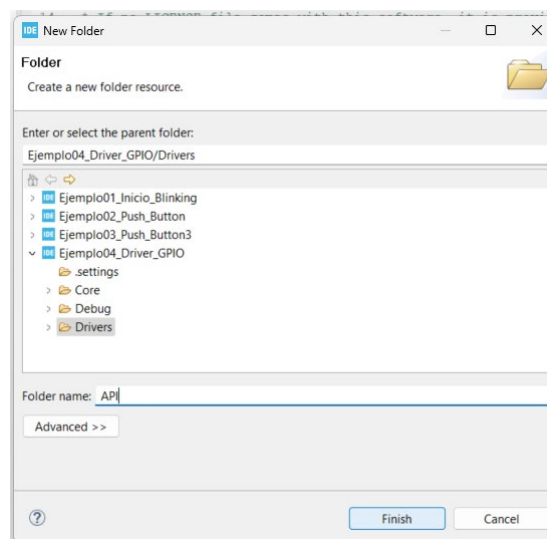
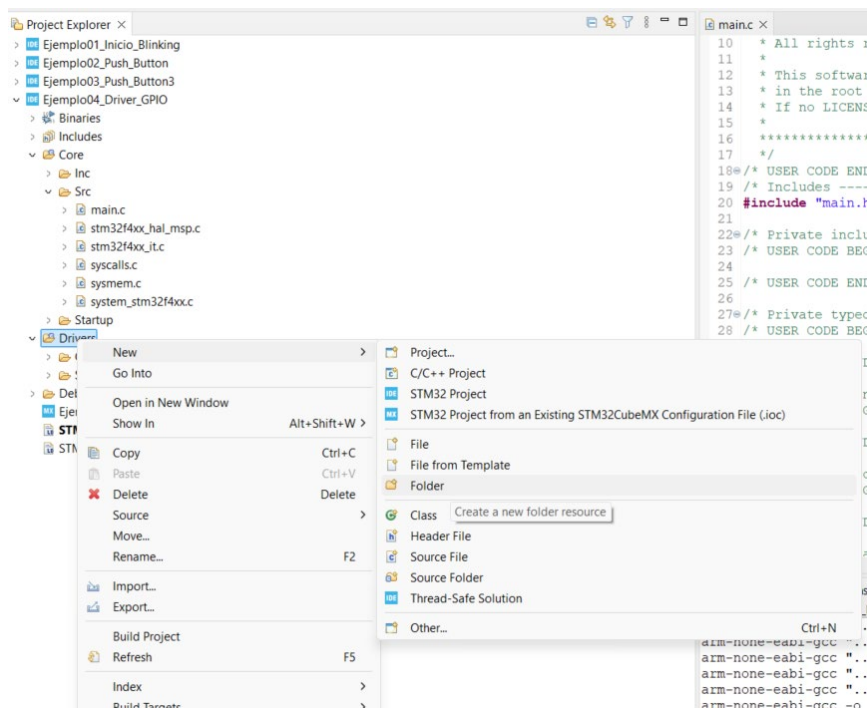
Guía Práctica para creación de drivers en STM32CubeIDE

Creación de driver GPIO

Partimos de la base del proyecto anterior **Ejemplo03_Push_Button3**. Lo copiamos y lo pegamos dentro del espacio **Project Explorer**, renombramos el proyecto:

Ejemplo04_Driver_GPIO; eliminamos los archivos “.launch” y renombramos el archivo “.ioc” con el nombre de nuestro proyecto para poder usarlo de ser necesario.

Comenzamos:

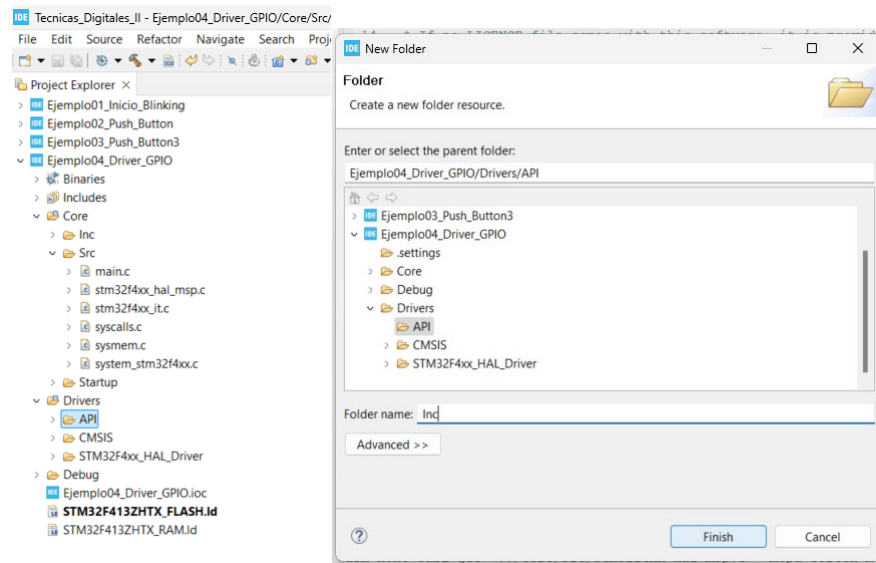


Autor: Prof. Ing. Rubén D. Mansilla

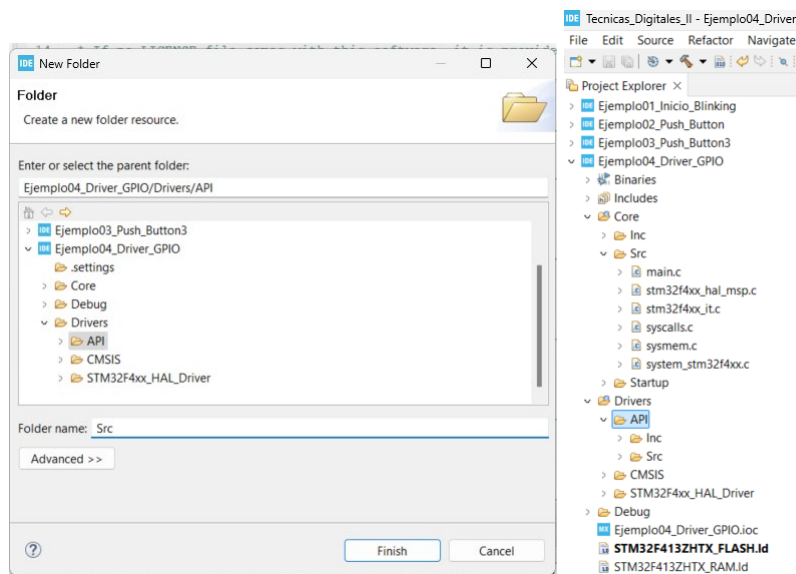
Técnicas Digitales II

Departamento Electrónica

Creamos la carpeta **API** en **Drivers**. En esta carpeta irán todos los drivers que creemos o que agreguemos a nuestro proyecto. Figuras de arriba.



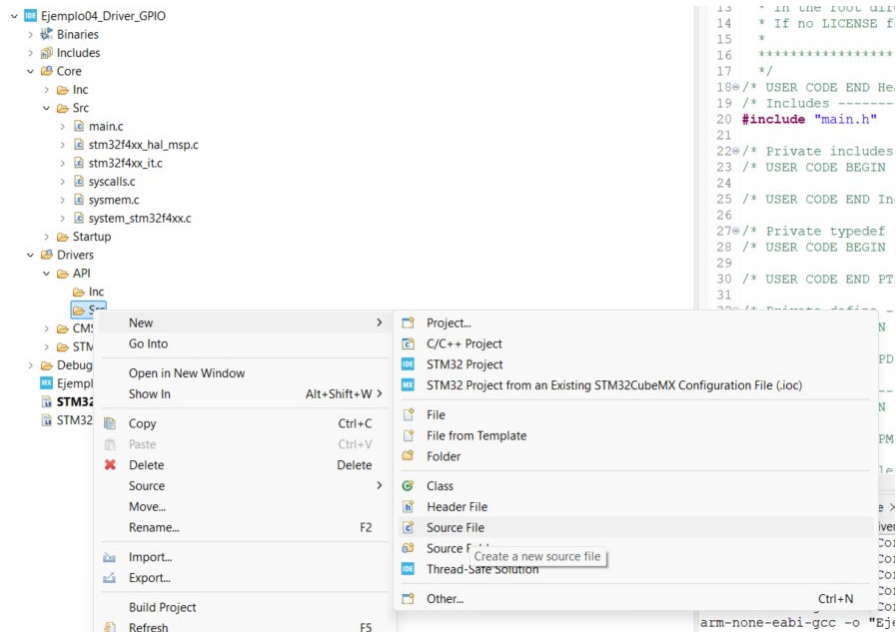
Creamos carpeta **Inc** ("Include") donde irán nuestros archivos *header* (*.h)



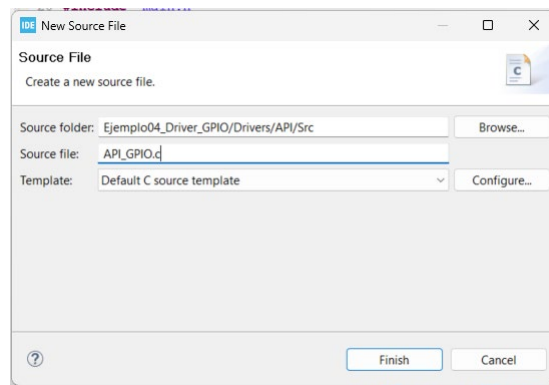
Creamos nuestra carpeta **Src** ("Sources") donde irán nuestros archivos fuente de los drivers que incluyamos en nuestro proyecto.

Técnicas Digitales II

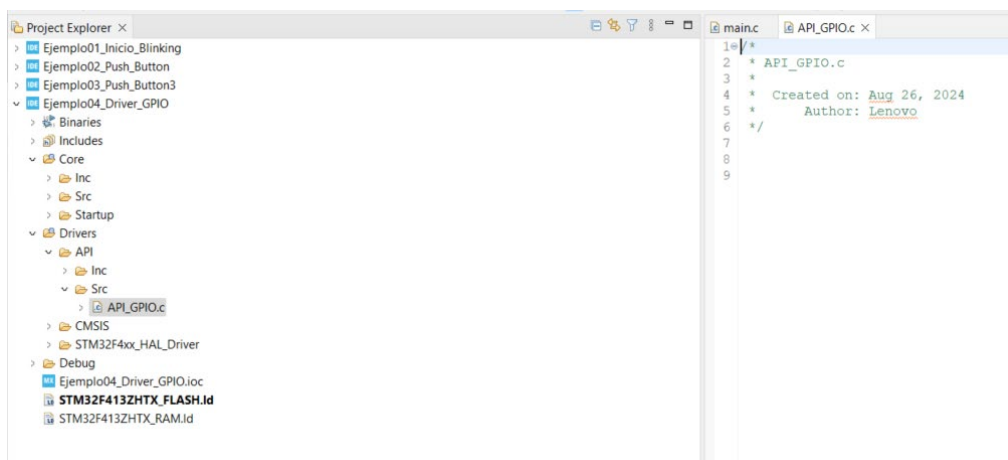
Departamento Electrónica



Creamos nuestro archivo fuente (“source”) en la carpeta **API**.



Le ponemos un nombre que, se sugiere, sea indicativo de la función que cumple: en este caso lo llamaré **API_GPIO.c** y será una plantilla típica para archivo tipo *source* en lenguaje C.



Autor: Prof. Ing. Rubén D. Mansilla

Técnicas Digitales II

Departamento Electrónica

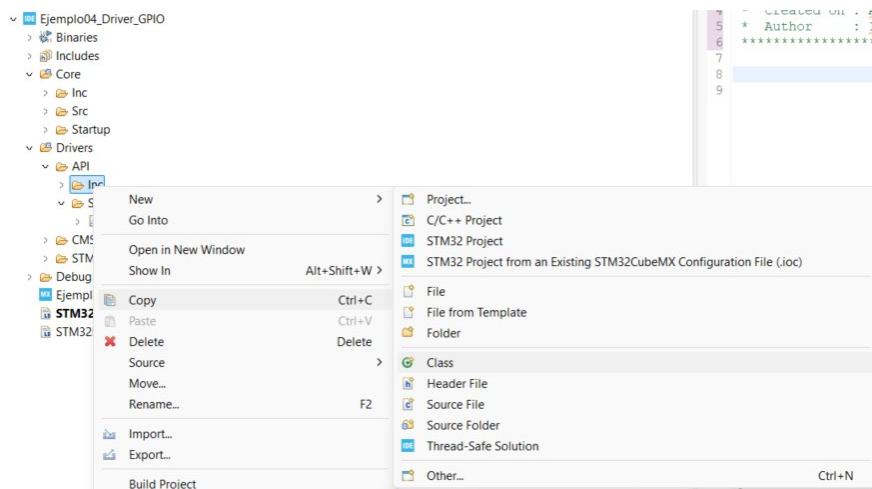
Y se crea la plantilla para nuestro archivo fuente *API_GPIO.c* que podemos editar personalizándolo con nuestro nombre de autor y alguna aclaración.



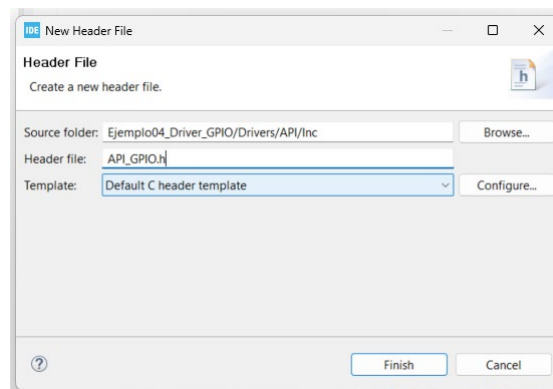
```
1 //*****  
2 * API_GPIO.c  
3 *  
4 * Created on : Aug 26, 2024  
5 * Author : Ing. Rubén D. Mansilla  
6 *  
7 *****/  
8  
9
```

Por ejemplo, lo que vemos en figura de arriba.

Luego hacemos lo mismo para crear nuestro archivo *header* correspondiente a este *source*.



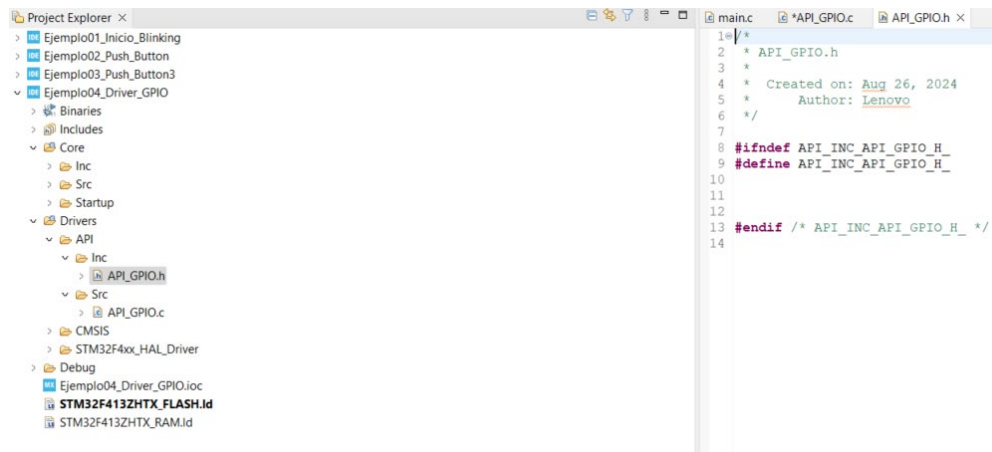
Creamos un archivo tipo “*header*”



Le ponemos un nombre que, se sugiere, sea indicativo de la función que cumple: en este caso lo llamaré **API_GPIO.h** y será una plantilla típica para archivo tipo *header* en lenguaje C.

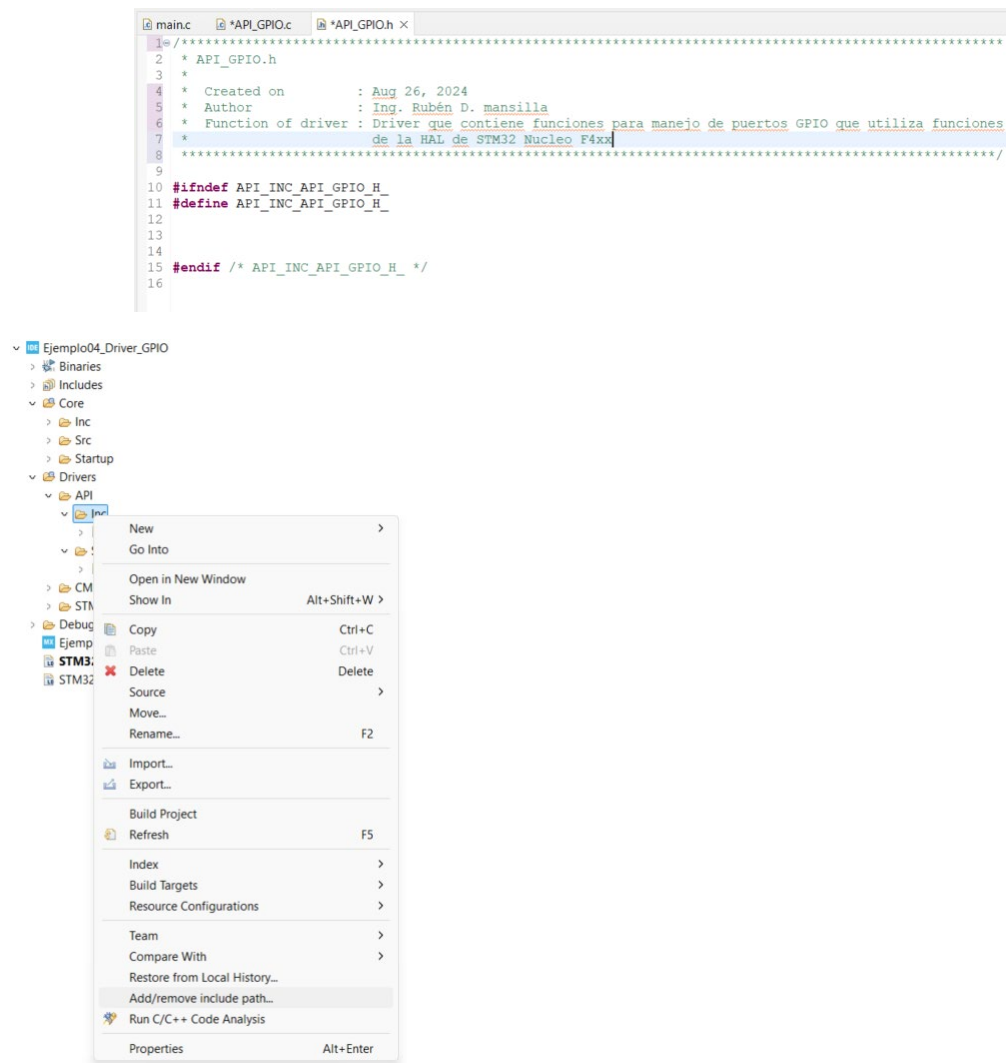
Técnicas Digitales II

Departamento Electrónica



Y se crea una plantilla *header* para nuestro driver **API_GPIO**

Lo podemos editar y personalizar

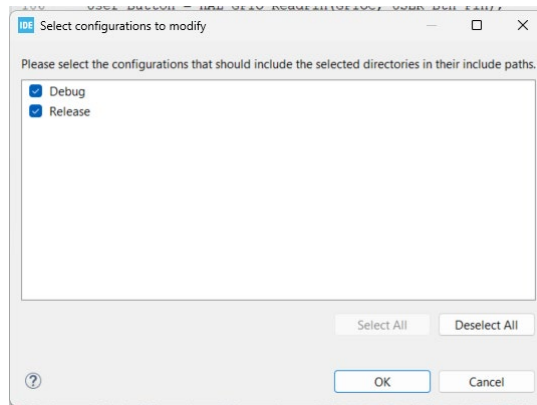


Antes que nada, me ubico sobre la carpeta **Inc** de **API** y con click derecho sobre esta selecciono **Add/remove include path...**

Autor: Prof. Ing. Rubén D. Mansilla

Técnicas Digitales II

Departamento Electrónica



Para asegurarme de incluir este *path* para que nuestro proyecto tome los drivers nuevos al compilar y al trabajar con los mismos. Asegurarse que estén seleccionados los ítems con tilde azul.

Ahora falta copiar la inicialización del GPIO desde *main.c*

Preparación del archivo **API_GPIO.c**

```
main.c  API_GPIO.c  API_GPIO.h
1  /*
2  * API_GPIO.c
3  *
4  * Created on : Aug 26, 2024
5  * Author    : Ing. Rubén D. Mansilla
6  */
7
8  /* Includes */
9  #include "main.h"
10
11 /*Defines */
12
13 /*Declaration of variables */
14 //Valores esperados para LDx: LD1_Pin|LD3_Pin|LD2_Pin
15 led_t LDx;
16
17 /** Function definition */
18 /**
19  * @brief GPIO Initialization Function
20  * @param None
21  * @retval None
22  */
23
```

Preparo los campos que tendrá el archivo *sources* como se ve en la figura de arriba.

Aquí debo incluir también le *header* correspondiente a *source* en la línea 10:

#include "API_GPIO.h"

Que no está en la figura de arriba.

Técnicas Digitales II

Departamento Electrónica

```
1= *****
2 * API_GPIO.h
3 *
4 * Created on      : Aug 26, 2024
5 * Author         : Ing. Rubén D. mansilla
6 * Function of driver : Driver que contiene funciones para manejo de puertos GPIO que utiliza funciones
7 *                  de la HAL de STM32 Nucleo F4xx
8 *****
9
10 #ifndef API_INC_API_GPIO_H_
11 #define API_INC_API_GPIO_H_
12
13 /* Exported types *****
14 typedef uint16_t led_t; /*Importante que el tipo sea uint16_t, si no, no funciona LD3_Pin*/
15
16
17 /* Exported functions prototypes *****
18 void MX_GPIO_Init(void);
19
20
21
22 #endif /* API_INC_API_GPIO_H_ */
23
```

Preparo los campos del archivo *header* como se ve en la figura de arriba. Definimos el prototipo de la función de inicialización del módulo GPIO como se ve en línea 18. Esta declaración del prototipo lo copiamos del archivo *main.c*, línea 53 figura de abajo.

```
main.c x API_GPIO.c API_GPIO.h
31
32 /* Private define -----
33 /* USER CODE BEGIN PD */
34
35 /* USER CODE END PD */
36
37 /* Private macro -----
38 /* USER CODE BEGIN PM */
39
40 /* USER CODE END PM */
41
42 /* Private variables -----
43 UART_HandleTypeDef huart3;
44
45 PCD_HandleTypeDef hpcd_USB_OTG_FS;
46
47 /* USER CODE BEGIN PV */
48
49 /* USER CODE END PV */
50
51 /* Private function prototypes -----
52 void SystemClock_Config(void);
53 static void MX_GPIO_Init(void);
54 static void MX_USART3_UART_Init(void);
55 static void MX_USB_OTG_FS_PCD_Init(void);
56 /* USER CODE BEGIN PFP */
57
58 /* USER CODE END PFP */
59
```

Elimino “*static*” que restringe el ámbito de esta función al archivo donde se encuentra declarada. Y “comento” (elimino) esta línea en *main.c* porque voy a pasar a declararla y usarla desde el driver que estamos creando. Línea 53 en figura de arriba.

Luego copio todo el código de desarrollo de la función de inicialización desde *main.c* y lo pego en *API_GPIO.c*. este código lo genera automáticamente el IDE, (el Mx), cuando inicializo mi proyecto y configuro el GPIO desde el *ioc*.

Técnicas Digitales II

Departamento Electrónica

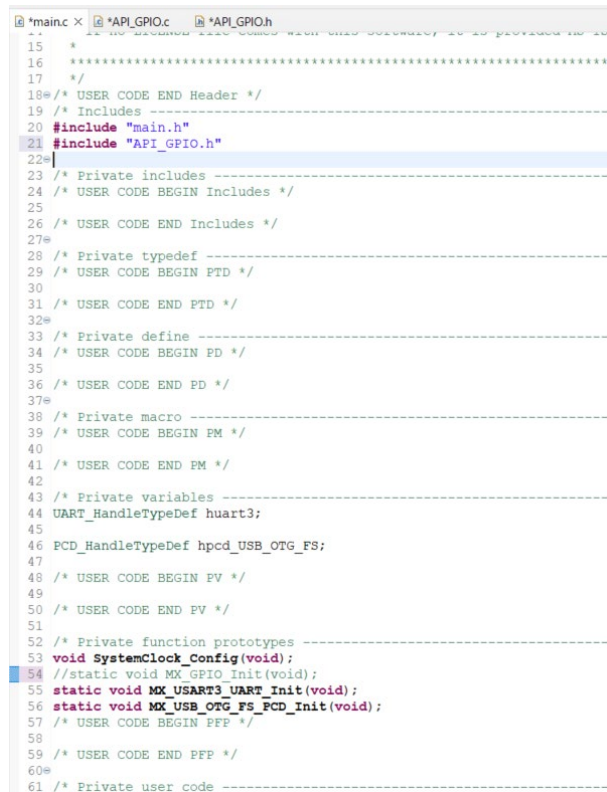
```
main.c x API_GPIO.c API_GPIO.h
245  * @brief GPIO Initialization Function
246  * @param None
247  * @retval None
248  */
249 static void MX_GPIO_Init(void)
250 {
251     GPIO_InitTypeDef GPIO_InitStruct = {0};
252     /* USER CODE BEGIN MX_GPIO_Init_1 */
253     /* USER CODE END MX_GPIO_Init_1 */
254
255     /* GPIO Ports Clock Enable */
256     __HAL_RCC_GPIOC_CLK_ENABLE();
257     __HAL_RCC_GPIOH_CLK_ENABLE();
258     __HAL_RCC_GPIOB_CLK_ENABLE();
259     __HAL_RCC_GPIOD_CLK_ENABLE();
260     __HAL_RCC_GPIOG_CLK_ENABLE();
261     __HAL_RCC_GPIOA_CLK_ENABLE();
262
263     /*Configure GPIO pin Output Level */
264     HAL_GPIO_WritePin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin, GPIO_PIN_RESET);
265
266     /*Configure GPIO pin Output Level */
267     HAL_GPIO_WritePin(USB_PowerSwitchOn_GPIO_Port, USB_PowerSwitchOn_Pin, GPIO_PIN_RESET);
268
269     /*Configure GPIO pin : USER_Btn_Pin */
270     GPIO_InitStruct.Pin = USER_Btn_Pin;
271     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
272     GPIO_InitStruct.Pull = GPIO_NOPULL;
273     HAL_GPIO_Init(USER_Btn_GPIO_Port, &GPIO_InitStruct);
274
275     /*Configure GPIO pins : LD1_Pin LD3_Pin LD2_Pin */
276     GPIO_InitStruct.Pin = LD1_Pin|LD3_Pin|LD2_Pin;
277     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
278     GPIO_InitStruct.Pull = GPIO_NOPULL;
279     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
280     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
281
282     /*Configure GPIO pin : USB_PowerSwitchOn_Pin */
283     GPIO_InitStruct.Pin = USB_PowerSwitchOn_Pin;
284     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
285     GPIO_InitStruct.Pull = GPIO_NOPULL;
286     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
287     HAL_GPIO_Init(USB_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);
288
289     /*Configure GPIO pin : USB_OverCurrent_Pin */
290     GPIO_InitStruct.Pin = USB_OverCurrent_Pin;
291     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
292     GPIO_InitStruct.Pull = GPIO_NOPULL;
293     HAL_GPIO_Init(USB_OverCurrent_GPIO_Port, &GPIO_InitStruct);
294
295
```

El código de desarrollo de la función de inicialización del módulo GPIO se encuentra en *main.c* desde la línea 244 hasta la línea 297. La corto y la pego en *API_GPIO.c*, en el campo “*Function definition*” a partir de la línea 17. Ver figura de abajo

```
main.c x API_GPIO.c x API_GPIO.h
14 //Valores esperados para LDx: LD1_Pin|LD3_Pin|LD2_Pin
15 led_t LDx;
16
17 /** Function definition *****/
18 /**
19  * @brief GPIO Initialization Function
20  * @param None
21  * @retval None
22  */
23 static void MX_GPIO_Init(void)
24 {
25     GPIO_InitTypeDef GPIO_InitStruct = {0};
26     /* USER CODE BEGIN MX_GPIO_Init_1 */
27     /* USER CODE END MX_GPIO_Init_1 */
28
29     /* GPIO Ports Clock Enable */
30     __HAL_RCC_GPIOC_CLK_ENABLE();
31     __HAL_RCC_GPIOH_CLK_ENABLE();
32     __HAL_RCC_GPIOB_CLK_ENABLE();
33     __HAL_RCC_GPIOD_CLK_ENABLE();
34     __HAL_RCC_GPIOG_CLK_ENABLE();
35     __HAL_RCC_GPIOA_CLK_ENABLE();
36
37     /*Configure GPIO pin Output Level */
38     HAL_GPIO_WritePin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin, GPIO_PIN_RESET);
39
40     /*Configure GPIO pin Output Level */
41     HAL_GPIO_WritePin(USB_PowerSwitchOn_GPIO_Port, USB_PowerSwitchOn_Pin, GPIO_PIN_RESET);
42
43     /*Configure GPIO pin : USER_Btn_Pin */
44     GPIO_InitStruct.Pin = USER_Btn_Pin;
45     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
46     GPIO_InitStruct.Pull = GPIO_NOPULL;
47     HAL_GPIO_Init(USER_Btn_GPIO_Port, &GPIO_InitStruct);
48
49     /*Configure GPIO pins : LD1_Pin LD3_Pin LD2_Pin */
50     GPIO_InitStruct.Pin = LD1_Pin|LD3_Pin|LD2_Pin;
51     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
52     GPIO_InitStruct.Pull = GPIO_NOPULL;
53     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
54     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
55
56     /*Configure GPIO pin : USB_PowerSwitchOn_Pin */
57     GPIO_InitStruct.Pin = USB_PowerSwitchOn_Pin;
58     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
59     GPIO_InitStruct.Pull = GPIO_NOPULL;
60     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
61     HAL_GPIO_Init(USB_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);
62
63     /*Configure GPIO pin : USB_OverCurrent_Pin */
64
```

Autor: Prof. Ing. Rubén D. Mansilla

Regreso a *main.c* e incluyo mi archivo *header* en el campo correspondiente a inclusión de librerías, a partir de la línea 19 de la figura de abajo.



```
15 *
16 *****
17 */
18 /* USER CODE END Header */
19 /* Includes -----
20 #include "main.h"
21 #include "API_GPIO.h"
22
23 /* Private includes -----
24 /* USER CODE BEGIN Includes */
25
26 /* USER CODE END Includes */
27
28 /* Private typedef -----
29 /* USER CODE BEGIN PTD */
30
31 /* USER CODE END PTD */
32
33 /* Private define -----
34 /* USER CODE BEGIN PD */
35
36 /* USER CODE END PD */
37
38 /* Private macro -----
39 /* USER CODE BEGIN PM */
40
41 /* USER CODE END PM */
42
43 /* Private variables -----
44 UART_HandleTypeDef huart3;
45
46 PCD_HandleTypeDef hpcd_USB_OTG_FS;
47
48 /* USER CODE BEGIN PV */
49
50 /* USER CODE END PV */
51
52 /* Private function prototypes -----
53 void SystemClock_Config(void);
54 //static void MX_GPIO_Init(void);
55 static void MX_USART3_UART_Init(void);
56 static void MX_USB_OTG_FS_PCD_Init(void);
57 /* USER CODE BEGIN PFP */
58
59 /* USER CODE END PFP */
60
61 /* Private user code -----
```

Ya tengo la declaración de la función de inicialización de GPIO: **MX_GPIO_Init()** en *API_GPIO.h*, su desarrollo en *API:GPIO.c* y está incluida en las librerías de mi proyecto en *main.c*

NOTA1: Los nombres de los archivos *header* y *source* del driver deben ser escritos con su extensión correspondiente (sin omitir la extensión) “.h” y “.c” respectivamente para que el IDE genere la plantilla, si no, solo creara un archivo vacío.

NOTA2: Asegurarse de eliminar el prefijo “static” en la declaración de la función de inicialización del módulo GPIO, figura de abajo, línea 23, si no al compilar dará error.

Técnicas Digitales II

Departamento Electrónica

```
main.c API_GPIO.c x API_GPIO.h mainh
1= /*****
2  * API_GPIO.c
3  *
4  * Created on : Aug 26, 2024
5  * Author    : Ing. Rubén D. Mansilla
6  *****/
7
8 /* Includes *****/
9 #include "main.h"
10
11 /*Defines *****/
12
13 /*Declaration of variables *****/
14 //Valores esperados para LDx: LD1_Pin|LD3_Pin|LD2_Pin
15 //led_t LDx;
16
17 /*** Function definition *****/
18 /**
19  * @brief GPIO Initialization Function
20  * @param None
21  * @retval None
22  */
23 void MX_GPIO_Init(void)
24 {
25     GPIO_InitTypeDef GPIO_InitStruct = {0};
26 /* USER CODE BEGIN MX_GPIO_Init_1 */
27 /* USER CODE END MX_GPIO_Init_1 */
28
29     /* GPIO Ports Clock Enable */
30     __HAL_RCC_GPIOC_CLK_ENABLE();
31     __HAL_RCC_GPIOH_CLK_ENABLE();
32     __HAL_RCC_GPIOB_CLK_ENABLE();
33     __HAL_RCC_GPIOD_CLK_ENABLE();
34     __HAL_RCC_GPIOG_CLK_ENABLE();
35     __HAL_RCC_GPIOA_CLK_ENABLE();
36
37     /*Configure GPIO pin Output Level */
38     HAL_GPIO_WritePin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin, GPIO_PIN_RESET);
39
40     /*Configure GPIO pin Output Level */
```

NOTA3: Comentar la declaración de la variable **LDx** en *API_GPIO.c* línea 15 porque todavía no la usamos y dará error al compilar. Figura de abajo. La des comentaremos una vez que declaremos e implementemos las funciones del driver.

```
1= /*****
2  * API_GPIO.c
3  *
4  * Created on : Aug 26, 2024
5  * Author    : Ing. Rubén D. Mansilla
6  *****/
7
8 /* Includes *****/
9 #include "main.h"
10
11 /*Defines *****/
12
13 /*Declaration of variables *****/
14 //Valores esperados para LDx: LD1_Pin|LD3_Pin|LD2_Pin
15 //led_t LDx;
16
17 /*** Function definition *****/
18 /**
19  * @brief GPIO Initialization Function
20  * @param None
21  * @retval None
22  */
23 void MX_GPIO_Init(void)
24 {
25     GPIO_InitTypeDef GPIO_InitStruct = {0};
26 /* USER CODE BEGIN MX_GPIO_Init_1 */
27 /* USER CODE END MX_GPIO_Init_1 */
28
29     /* GPIO Ports Clock Enable */
30     __HAL_RCC_GPIOC_CLK_ENABLE();
31     __HAL_RCC_GPIOH_CLK_ENABLE();
32     __HAL_RCC_GPIOB_CLK_ENABLE();
33     __HAL_RCC_GPIOD_CLK_ENABLE();
34     __HAL_RCC_GPIOG_CLK_ENABLE();
35     __HAL_RCC_GPIOA_CLK_ENABLE();
36
37     /*Configure GPIO pin Output Level */
38     HAL_GPIO_WritePin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin, GPIO_PIN_RESET);
39
40     /*Configure GPIO pin Output Level */
```

Hasta aquí debería compilar sin errores ni warnings.

Técnicas Digitales II

Departamento Electrónica

```
Problems Tasks Console Properties
CDT Build Console [Ejemplo04_Driver_GPIO]

arm-none-eabi-size Ejemplo04_Driver_GPIO.elf
arm-none-eabi-objdump -h -S Ejemplo04_Driver_GPIO.elf > "Ejemplo04_Driver
text data bss dec hex filename
11988 12 2892 14892 3a2c Ejemplo04_Driver_GPIO.elf
Finished building: default.size.stdout

Finished building: Ejemplo04_Driver_GPIO.list

17:32:58 Build Finished. 0 errors, 0 warnings. (took 1s.673ms)
```

Declaración de funciones del driver GPIO. Figura de abajo.

```
@ main.c @ API_GPIO.c @ *API_GPIO.h x @ main.h
1= /*****
2 * API_GPIO.h
3 *
4 * Created on : Aug 26, 2024
5 * Author : Ing. Rubén D. mansilla
6 * Function of driver : Driver que contiene funciones para manejo de puertos GPIO que utiliza funciones
7 * de la HAL de STM32 Nucleo F4xx
8 *****/
9
10 #ifndef API_INC_API_GPIO_H_
11 #define API_INC_API_GPIO_H_
12
13 /* Exported types *****/
14 typedef uint16_t led_t; /*Importante que el tipo sea uint16_t, si no, no funciona LD3_Pin*/
15
16 /* Exported functions prototypes *****/
17
18 void MX_GPIO_Init(void);
19 void writeLedOn_GPIO(led_t LDx);
20 void writeLedOff_GPIO(led_t LDx);
21 void toggleLed_GPIO(led_t LDx);
22
23 #endif /* API_INC_API_GPIO_H_ */
```

Implementación de funciones básicas de GPIO

```
@ main.c @ *API_GPIO.c x @ *API_GPIO.h @ main.h
62
63 /*Configure GPIO pin : USB_OverCurrent_Pin */
64 GPIO_InitStruct.Pin = USB_OverCurrent_Pin;
65 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
66 GPIO_InitStruct.Pull = GPIO_NOPULL;
67 HAL_GPIO_Init(USB_OverCurrent_GPIO_Port, &GPIO_InitStruct);
68
69 /* USER CODE BEGIN MX_GPIO_Init_2 */
70 /* USER CODE END MX_GPIO_Init_2 */
71 }
72
73 /**
74 * @brief GPIO Led on Function
75 * @param led_t LDx
76 * @retval None
77 */
78 void writeLedOn_GPIO(led_t LDx)
79 {
80 HAL_GPIO_WritePin(GPIOB, LDx, GPIO_PIN_SET);
81 }
82
83 /**
84 * @brief GPIO Led off Function
85 * @param led_t LDx
86 * @retval None
87 */
88 void writeLedOff_GPIO(led_t LDx)
89 {
90 HAL_GPIO_WritePin(GPIOB, LDx, GPIO_PIN_RESET);
91 }
92
93 /**
94 * @brief GPIO toggle Led Function
95 * @param led_t LDx
96 * @retval None
97 */
98 void toggleLed_GPIO(led_t LDx)
99 {
100 HAL_GPIO_TogglePin(GPIOB, LDx);
101 }
```

@brief → Descripción de la función (que hace).

@param → Parámetros que recibe la función.

@retval → Valores que retorna la función.

Autor: Prof. Ing. Rubén D. Mansilla

Técnicas Digitales II

Departamento Electrónica

```
*main.c *API_GPIO.c *API_GPIO.h main.h
1 //*****
2 * API_GPIO.c
3 *
4 * Created on : Aug 26, 2024
5 * Author : Ing. Rubén D. Mansilla
6 *****
7
8 /* Includes *****/
9 #include "main.h"
10
11 /*Defines *****/
12
13 /*Declaration of variables *****/
14 //Valores esperados para LDx: LD1_Pin|LD3_Pin|LD2_Pin
15 led_t LDx;
16
17 /** Function definition *****/
18 /**
19  * @brief GPIO Initialization Function
20  * @param None
21  * @retval None
22  */
23 void MX_GPIO_Init(void)
24 {
25     GPIO_InitTypeDef GPIO_InitStruct = {0};
26     /* USER CODE BEGIN MX_GPIO_Init_1 */
27     /* USER CODE END MX_GPIO_Init_1 */
28
29     /* GPIO Ports Clock Enable */
30     HAL_RCC_GPIOC_CLK_ENABLE();
31     HAL_RCC_GPIOH_CLK_ENABLE();
32     HAL_RCC_GPIOB_CLK_ENABLE();
33     HAL_RCC_GPIOD_CLK_ENABLE();
34     HAL_RCC_GPIOG_CLK_ENABLE();
35     HAL_RCC_GPIOA_CLK_ENABLE();
```

Des comento (activo) la declaración de la variable que usarán las funciones del driver. Figura de arriba.

En *main.c* en el campo “Private define” (línea 33) defino las etiquetas que usaré para referenciar a los Leds de la placa que utiliza el driver GPIO. Figura de abajo.

```
*main.c *API_GPIO.c *API_GPIO.h main.h
64 /* Private user code -----*/
65 /* USER CODE BEGIN 0 */
66
67 /* USER CODE END 0 */
68
69 /**
70  * @brief The application entry point.
71  * @retval int
72  */
73 int main(void)
74 {
75
76     /* USER CODE BEGIN 1 */
77
78     /* USER CODE END 1 */
79
80     /* MCU Configuration-----*/
81
82     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
83     HAL_Init();
84
85     /* USER CODE BEGIN Init */
86     uint16_t LEDS[3] = {LED1, LED2, LED3}; /*Creo vector de LEDs de usuario*/
87     uint16_t User_Button;
88     /* USER CODE END Init */
89
90     /* Configure the system clock */
91     SystemClock_Config();
92
93     /* USER CODE BEGIN SysInit */
94
95     /* USER CODE END SysInit */
96
97     /* Initialize all configured peripherals */
98     MX_GPIO_Init(); /*Inicializa los LEDs de usuario en 0" (RESET), el boton de usuario es
99                     inicializa en modo activo por flanco creciente*/
100     MX_USART3_UART_Init();
101     MX_USB_OTG_FS_PCD_Init();
102     /* USER CODE BEGIN 2 */
103
104     ...
```

Acondicionamos nuestro *main.c* para adecuar nuestra aplicación al uso del driver que creamos para GPIO. Esto reemplazará el uso de funciones de la HAL en *main.c*

Usamos las etiquetas que creamos recién en la creación de nuestro vector para manejo de los Leds. Línea 86 en *main.c* figura de arriba.

En *main.c* reemplazamos las funciones de la HAL que manejan el módulo GPIO y acomodamos los parámetros que estas funciones reciben. De esta manera nuestro

Autor: Prof. Ing. Rubén D. Mansilla

código ya no usará funciones de la HAL para el manejo del GPIO. En este ejemplo faltaría crear una función en el driver que lea el valor de entrada del pulsador de la placa.

```
main.c x API_GPIO.c API_GPIO.h main.h
93 /* USER CODE BEGIN SysInit */
94
95 /* USER CODE END SysInit */
96
97 /* Initialize all configured peripherals */
98 MX_GPIO_Init(); /*Inicializa los LEDs de usuario en 0" (RESET), el boton de usuario es activo alto e
99 inicializa en modo activo por flanco creciente*/
100 MX_USART3_UART_Init();
101 MX_USB_OTG_FS_PCD_Init();
102 /* USER CODE BEGIN 2 */
103
104 /* USER CODE END 2 */
105
106 /* Infinite loop */
107 /* USER CODE BEGIN WHILE */
108 while (1)
109 {
110     User_Button = HAL_GPIO_ReadPin(GPIOC, USER_Btn_Pin);
111     if (User_Button == 1)
112     {
113         writeLedOff_GPIO(LED3);
114         for (uint8_t i = 0; i < 3; i++)
115         {
116             writeLedOn_GPIO(LED3[i]);
117             HAL_Delay(200);
118             writeLedOff_GPIO(LED3[i]);
119             HAL_Delay(200);
120         }
121     }
122     else
123     {
124         writeLedOn_GPIO(LED3);
125     }
126 } /* USER CODE END WHILE */
127
128 /* USER CODE BEGIN 3 */
129
130 /* USER CODE END 3 */
131 }
```

NOTA4: En *API_GPIO.c* se debe incluir su correspondiente *API_GPIO.h*. En realidad, esto se hace al principio.

```
main.c x API_GPIO.c x API_GPIO.h main.h
1 //*****
2 * API_GPIO.c
3 *
4 * Created on : Aug 26, 2024
5 * Author : Ing. Rubén D. Mansilla
6 *****/
7
8 /* Includes *****/
9 #include "main.h"
10 #include "API_GPIO.h"
11
12 /*Defines *****/
13
14 /*Declaration of variables *****/
15 //Valores esperados para LDx: LD1_Pin|LD3_Pin|LD2_Pin
16 led_t LDx;
17
18 /** Function definition *****/
19 /**
20 * @brief GPIO Initialization Function
21 * @param None
22 * @retval None
23 */
24 void MX_GPIO_Init(void)
25 {
26     GPIO_InitTypeDef GPIO_InitStruct = {0};
27 /* USER CODE BEGIN MX_GPIO_Init_1 */
28 /* USER CODE END MX_GPIO_Init_1 */
29
30 /* GPIO Ports Clock Enable */
31 HAL_RCC_GPIOC_CLK_ENABLE();
32 HAL_RCC_GPIOH_CLK_ENABLE();
33 HAL_RCC_GPIOB_CLK_ENABLE();
34 HAL_RCC_GPIOA_CLK_ENABLE();
35 HAL_RCC_GPIOG_CLK_ENABLE();
36 HAL_RCC_GPIOF_CLK_ENABLE();
37
38 /*Configure GPIO pin Output Level */
39 HAL_GPIO_WritePin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin, GPIO_PIN_RESET);
40 }
```

NOTA5: En *API_GPIO.h* incluyo librería **stdint.h** para poder usar el tipo **uint16_t** para definir el tipo **led_t** que se usará en las funciones del driver GPIO.

Técnicas Digitales II

Departamento Electrónica

```
main.c API_GPIO.c API_GPIO.h × main.h
1 // *****
2 * API_GPIO.h
3 *
4 * Created on      : Aug 26, 2024
5 * Author         : Ing. Rubén D. mansilla
6 * Function of driver : Driver que contiene funciones para manejo de puertos GPIO que utiliza funciones
7 *                  : de la HAL de STM32 Nucleo F4xx
8 * *****
9
10 #ifndef API_INC_API_GPIO_H_
11 #define API_INC_API_GPIO_H_
12
13 /* Includes *****
14 #include <stdint.h> /*Para poder usar el tipo uint16_t en mi nueva definición de tipo led_t*/
15
16 /* Exported types *****
17 typedef uint16_t led_t; /*Importante que el tipo sea uint16_t, si no, no funciona LD3_Pin*/
18
19
20 /* Exported functions prototypes *****
21 void MX_GPIO_Init(void);
22 void writeLedOn_GPIO(led_t LDx);
23 void writeLedOff_GPIO(led_t LDx);
24 void toggleLed_GPIO(led_t LDx);
25
26 #endif /* API_INC_API_GPIO_H_ */
```

Hechas esas modificaciones de las **NOTAS 4 y 5** compilamos sin errores ni warnings

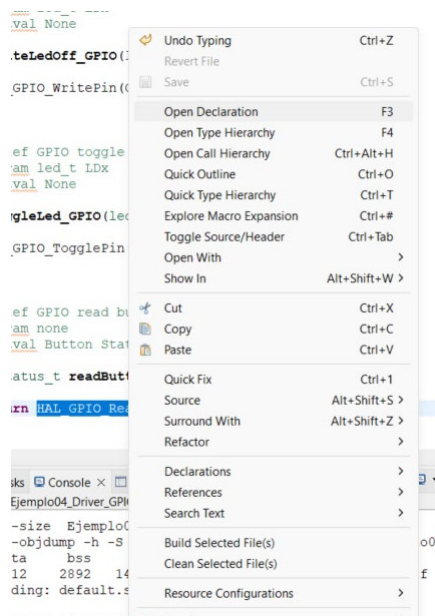
```
LD1 Build Console [Ejemplo04_Driver_GPIO]
arm-none-eabi-size Ejemplo04_Driver_GPIO.elf
arm-none-eabi-objdump -h -S Ejemplo04_Driver_GPIO.elf > "Ejemplo04_Driver
text data bss dec hex filename
12040 12 2892 14944 3a60 Ejemplo04_Driver_GPIO.elf
Finished building: default.size.stdout

Finished building: Ejemplo04_Driver_GPIO.list

18:53:01 Build Finished. 0 errors, 0 warnings. (took 1s.727ms)
```

Se agrega la función de lectura del estado del pulsador de la placa al driver GPIO.

Para crear la función **readButton_GPIO()** y usar la función de la HAL correspondiente hay que ver qué tipo de valor devuelve esta función. Para ver esto vamos a la declaración de la función de la HAL, seleccionando la función y haciendo click derecho:



Luego seleccionamos **Open Declaration** y vamos al archivo donde se declara esta función y vemos que devuelve un valor de tipo **GPIO_PinState**.

Autor: Prof. Ing. Rubén D. Mansilla

Técnicas Digitales II

Departamento Electrónica

```
main.c API_GPIO.c API_GPIO.h main.h stm32f4xx_hal_gpio.c ×
347 }
348 }
349 }
350 }
351 /**
352  * @}
353  */
354
355 /** @defgroup GPIO_Exported_Functions_Group2 IO operation functions
356  * @brief GPIO Read and Write
357  *
358  @verbatim
359  ##### IO operation functions #####
360  #####
361
362  @endverbatim
363  * @{}
364  */
365
366
367 /**
368  * @brief Reads the specified input port pin.
369  * @param GPIOx where x can be (A..K) to select the GPIO peripheral for STM32F
370  * @param GPIO_Pin specifies the port bit to read.
371  * @param This parameter can be GPIO_PIN_x where x can be (0..15).
372  * @retval The input port pin value.
373  */
374
375 HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
376 {
377     GPIO_PinState bitstatus;
378
379     /* Check the parameters */
380     assert_param(IS_GPIO_PIN(GPIO_Pin));
381
382     if((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
383     {
384         bitstatus = GPIO_PIN_SET;
385     }
386     else
```

Click derecho sobre este tipo y tenemos la declaración del mismo.

```
main.c API_GPIO.c API_GPIO.h main.h stm32f4xx_hal_gpio.c stm32f4xx_hal_gpio.h ×
58 This parameter can be a value of @ref GPIO_speed_
59
60 uint32_t Alternate; /*!< Peripheral to be connected to the selected pins.
61 This parameter can be a value of @ref GPIO_Alter
62 )GPIO_InitTypeDef;
63
64 /**
65  * @brief GPIO Bit SET and Bit RESET enumeration
66  */
67 typedef enum
68 {
69     GPIO_PIN_RESET = 0,
70     GPIO_PIN_SET
71 }GPIO_PinState;
72 /**
73  * @{}
74  */
75
```

Vemos que es una enumeración que bien podría resolverse con una variable de tipo booleana en nuestro prototipo de función. Incluimos entonces en nuestro *header* la librería *stdbool.h* (línea 15). Creamos un tipo personalizado **buttonStatus_t** que será tipo **bool** (línea 19) y declaramos la función **readButton_GPIO()** que devolverá un valor de tipo **buttonStatus_t** (buleano) (Línea 26). Figura de abajo.

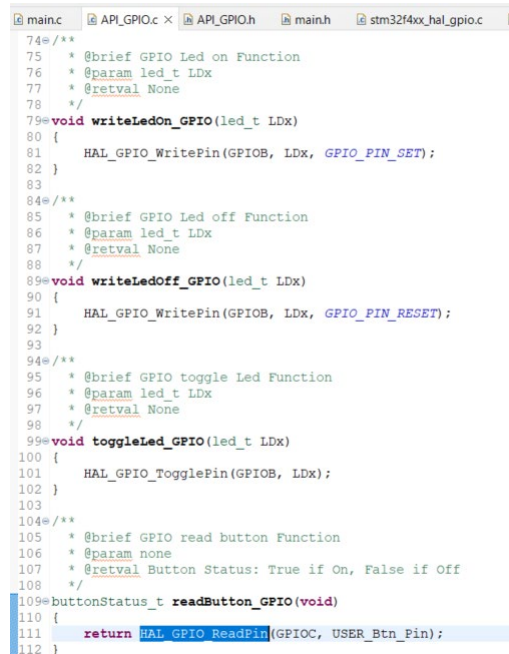
```
main.c API_GPIO.c API_GPIO.h × main.h stm32f4xx_hal_gpio.c stm32f4xx_hal_gpio.h
1 1=
2 2 * API_GPIO.h
3 3
4 4 * Created on : Aug 26, 2024
5 5 * Author : Ing. Rubén D. mansilla
6 6 * Function of driver : Driver que contiene funciones para manejo de puertos GPIO que utiliza funciones
7 7 de la HAL de STM32 Nucleo F4xx
8 8
9 9
10 10 #ifndef API_INC_API_GPIO_H_
11 11 #define API_INC_API_GPIO_H_
12 12
13 13 /* Includes
14 14 #include <stdint.h> /*Para poder usar el tipo uint16_t en mi nueva definición de tipo led_t*/
15 15 #include <stdbool.h>
16 16
17 17 /* Exported types
18 18 typedef uint16_t led_t; /*Importante que el tipo sea uint16_t, si no, no funciona LD3_Pin*/
19 19 typedef bool buttonStatus_t;
20 20
21 21 /* Exported functions prototypes
22 22 void MX_GPIO_Init(void);
23 23 void writeLedon_GPIO(led_t LDx);
24 24 void writeLedoff_GPIO(led_t LDx);
25 25 void toggleLed_GPIO(led_t LDx);
26 26 buttonStatus_t readButton_GPIO(void);
27 27
28 28 #endif /* API_INC_API_GPIO_H_ */
29 29
```

Autor: Prof. Ing. Rubén D. Mansilla

Técnicas Digitales II

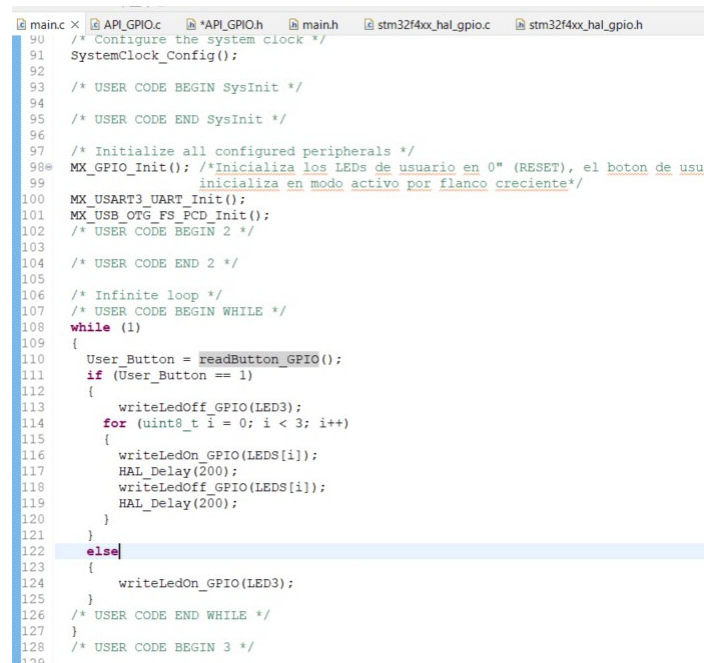
Departamento Electrónica

En nuestro *source* implementamos la función **readButton_GPIO()**. El detalle, para evitar un warning es agregar “**return**” a la función de la HAL para asegurar que mi función retorne el valor correspondiente que es coherente con un buleano que definimos como tipo *buttonStatus_t*. Figura de abajo.



```
main.c | API_GPIO.c x | API_GPIO.h | main.h | stm32f4xx_hal_gpio.c |
74 /**
75  * @brief GPIO Led on Function
76  * @param led_t Ldx
77  * @retval None
78  */
79 void writeLedOn_GPIO(led_t Ldx)
80 {
81     HAL_GPIO_WritePin(GPIOB, Ldx, GPIO_PIN_SET);
82 }
83
84 /**
85  * @brief GPIO Led off Function
86  * @param led_t Ldx
87  * @retval None
88  */
89 void writeLedOff_GPIO(led_t Ldx)
90 {
91     HAL_GPIO_WritePin(GPIOB, Ldx, GPIO_PIN_RESET);
92 }
93
94 /**
95  * @brief GPIO toggle Led Function
96  * @param led_t Ldx
97  * @retval None
98  */
99 void toggleLed_GPIO(led_t Ldx)
100 {
101     HAL_GPIO_TogglePin(GPIOB, Ldx);
102 }
103
104 /**
105  * @brief GPIO read button Function
106  * @param none
107  * @retval Button Status: True if On, False if Off
108  */
109 buttonStatus_t readButton_GPIO(void)
110 {
111     return HAL_GPIO_ReadPin(GPIOC, USER_Btn_Pin);
112 }
```

En *main.c* nuestro código ya no utilizará funciones de la HAL que son reemplazados por funciones de nuestro driver **API_GPIO** como se puede observar en la figura de abajo.

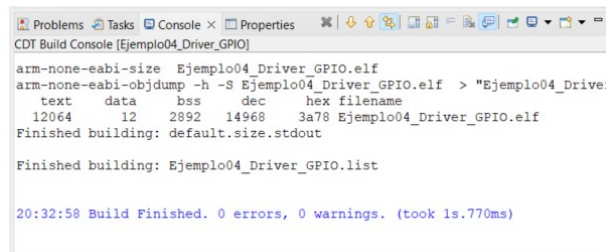


```
main.c x | API_GPIO.c | API_GPIO.h | main.h | stm32f4xx_hal_gpio.c | stm32f4xx_hal_gpio.h
90 /* Configure the system clock */
91 SystemClock_Config();
92
93 /* USER CODE BEGIN SysInit */
94
95 /* USER CODE END SysInit */
96
97 /* Initialize all configured peripherals */
98 MX_GPIO_Init(); /*Inicializa los LEDs de usuario en 0* (RESET), el boton de usu
99               /*Inicializa en modo activo por flanco creciente*/
100 MX_USART3_UART_Init();
101 MX_USB_OTG_FS_PCD_Init();
102 /* USER CODE BEGIN 2 */
103
104 /* USER CODE END 2 */
105
106 /* Infinite loop */
107 /* USER CODE BEGIN WHILE */
108 while (1)
109 {
110     User_Button = readButton_GPIO();
111     if (User_Button == 1)
112     {
113         writeLedOff_GPIO(LED3);
114         for (uint8_t i = 0; i < 3; i++)
115         {
116             writeLedOn_GPIO(LED3);
117             HAL_Delay(200);
118             writeLedOff_GPIO(LED3);
119             HAL_Delay(200);
120         }
121     }
122     else
123     {
124         writeLedOn_GPIO(LED3);
125     }
126     /* USER CODE END WHILE */
127 }
128 /* USER CODE BEGIN 3 */
129
```

Compilamos sin errores ni warnings

Técnicas Digitales II

Departamento Electrónica



```
Problems Tasks Console Properties
CDT Build Console [Ejemplo04_Driver_GPIO]

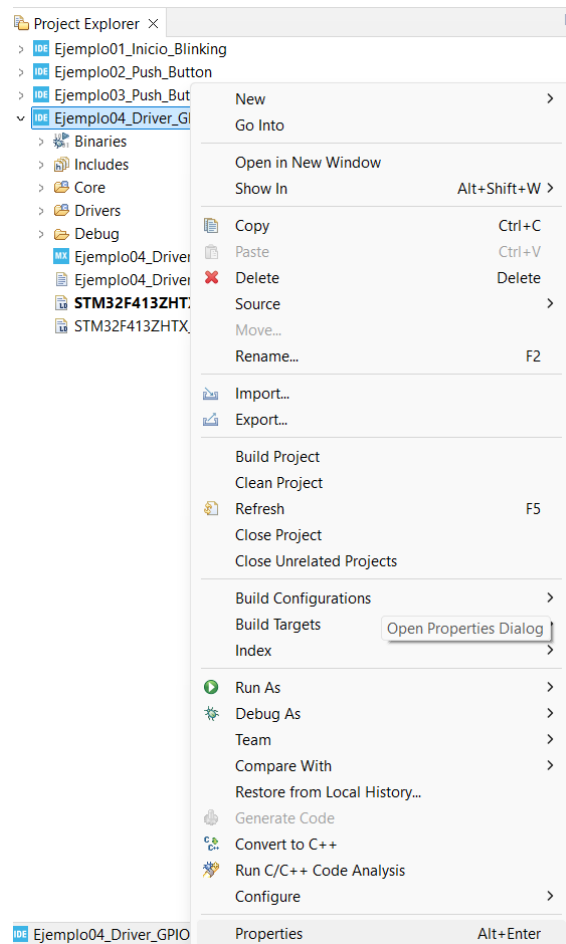
arm-none-eabi-size Ejemplo04_Driver_GPIO.elf
arm-none-eabi-objdump -h -S Ejemplo04_Driver_GPIO.elf > "Ejemplo04_Driver_GPIO.list"
text data bss dec hex filename
12064 12 2892 14968 3a78 Ejemplo04_Driver_GPIO.elf
Finished building: default.size.stdout

Finished building: Ejemplo04_Driver_GPIO.list

20:32:58 Build Finished. 0 errors, 0 warnings. (took 1s.770ms)
```

Quedaría para el siguiente proyecto reemplazar las funciones bloqueantes de *delay* de la HAL por funciones de retardo no bloqueantes que crearemos otro driver.

Nota6: Verificación de *path* para *include*: Antes de compilar se sugiere verificar que la carpeta *inc* de nuestra API (driver) se encuentre en el *path* de compilación del proyecto para que tome los archivos *header*. Si no esta en el *path* no los considera y dará error al compilar.

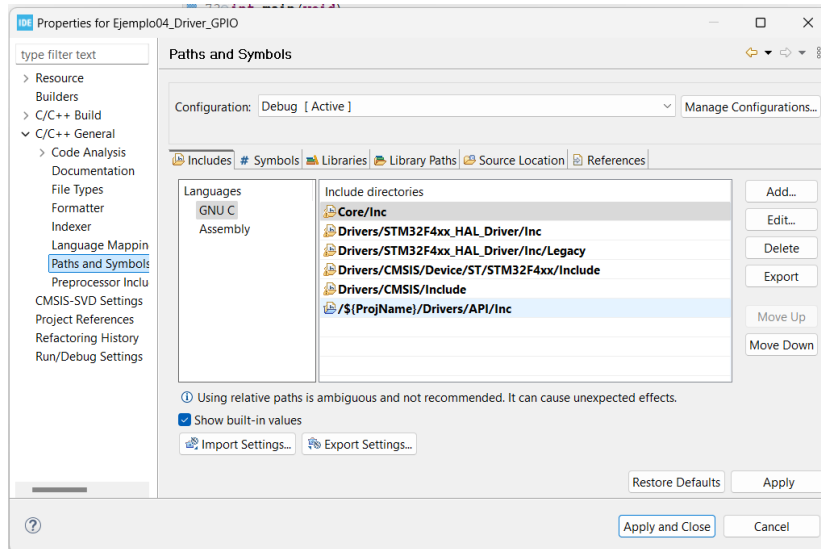


Click derecho sobre el proyecto y seleccionamos **Properties**: Figura de arriba.

Autor: Prof. Ing. Rubén D. Mansilla

Técnicas Digitales II

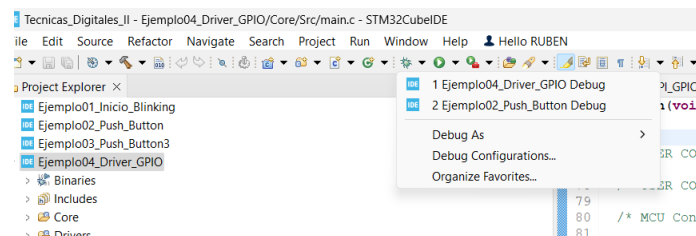
Departamento Electrónica



En la ventana de arriba, desplegamos **C/C++ General** y seleccionamos **Paths and Symbols**. Vemos el listado de las carpetas que se incluyen en la compilación. En el listado, vemos al final el path: `/$(ProjName)/Drivers/API/Inc` eso significa que el archivo *header* de nuestro driver será compilado en el proyecto junto con los demás *headers* de este. Si no figura en este listado nuestra carpeta **Drivers/API/Inc**, el archivo `API_GPIO.h` no será tenido en cuenta en la compilación y me dará errores.

Depuración del código y programación de la placa

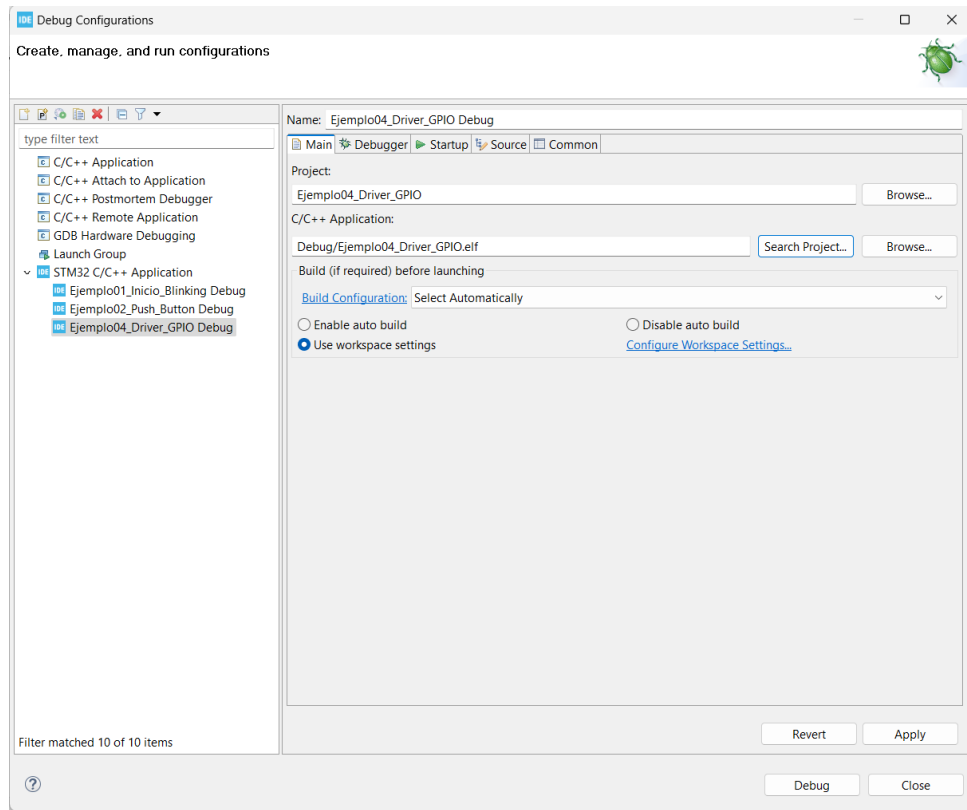
Para programar la placa vamos al escarabajo y desplegamos el menú de la figura de abajo → seleccionamos **Debug Configuration**.



Asegurarse que en el campo **Project**: este seleccionado el proyecto que queremos lanzar a debug y en el campo **C/C++ Application**: este el archivo `“.elf”` correspondiente como se ve en la figura de abajo.

Técnicas Digitales II

Departamento Electrónica



Una vez que nos aseguremos que los campos mencionados tienen los datos correctos, **Apply** y **Debug** para lanzar la depuración y programación de la placa.

NOTA7: Si hacemos doble click sobre **STM32 C/C++ Application** se actualiza el listado de proyectos para depurar. Esto se suele hacer cuando vamos a hacer Debug y programación de placa por primera vez para un proyecto.

Link al proyecto en repositorio GitHub de Técnicas Digitales II: [Ejemplo04 Driver GPIO](#)