Laboratorio #2

Laura Camila Blanco Gómez
Escuela de Ingeniería
Universidad Sergio Arboleda
Bogotá, Colombia
laura.blanco01@usa.edu.co

Santiago Cáceres Linares
Escuela de Ingeniería
Universidad Sergio Arboleda
Bogotá, Colombia
santiago.caceres01@usa.edu.co

I. Introducción

En este laboratorio se busca introducir conocimientos sobre la placa de desarrollo **Stm32f411 Blackpill**, el programador **St Link** y el IDE **STM32CubeIDE**.

Se busca implementar un circuito electrónico que involucre la interacción entre un pulsador, 2 LEDs por medio de un contador. El objetivo principal es desarrollar un sistema capaz de contar y visualizar las pulsaciones realizadas por el usuario, así como de manejar la respuesta a la acción del usuario en un tiempo determinado.

El circuito consta de tres componentes principales: un pulsador conectado a (B5), dos LEDs conectados (B4 y B3) y un contador que registra la cantidad de veces que el pulsador es presionado. El pulsador se utiliza como entrada para detectar la acción del usuario, mientras que los LEDs tienen la función de visualizar la respuesta del sistema ante dicha acción.

II. Marco teórico

Como recapitulación de ciertos de los tópicos tratados durante el laboratorio se parte para el correcto entendimiento de algunos conceptos mencionados en la sección anterior.

Stm32f411 Blackpill: El STM32F411 Blackpill es parte de la serie STM32F4 de microcontroladores de STMicroelectronics. Estos microcontroladores están basados en la arquitectura ARM Cortex-M4 y ofrecen un alto rendimiento y numerosas características, como una amplia gama de periféricos y capacidades de procesamiento digital de señales (DSP).

STM32CubeIDE: STM32CubeIDE es un entorno de desarrollo integrado (IDE) proporcionado por STMicroelectronics para programar y depurar microcontroladores STM32. Ofrece una interfaz gráfica intuitiva, herramientas de depuración avanzadas y una amplia gama de bibliotecas y ejemplos de código para facilitar el desarrollo de aplicaciones en STM32.

HAL(Hardware Abstraction Layer): HAL es una capa de abstracción de hardware proporcionada por STMicroelectronics. Permite a los desarrolladores interactuar con los periféricos del microcontrolador de manera independiente del hardware subyacente. Esto simplifica el desarrollo de aplicaciones al proporcionar una API coherente y portátil para controlar los periféricos en diferentes microcontroladores STM32.

GPIO(General Purpose Input/Output): Los pines GPIO son pines configurables en un microcontrolador que se pueden utilizar para entrada o salida digital. En el caso del STM32F411 Blackpill, estos pines se pueden utilizar para leer entradas digitales (por ejemplo, el estado de un botón) o controlar salidas digitales (por ejemplo, encender o apagar un LED).

III. Procedimiento

Para iniciar el laboratorio se realizan las siguientes configuraciones se selecciona el pin B5 como una **GPIO_Input**y los pines B4 y B3 se seleccionan como **GPIO_Output**, adicionalmente a estas configuraciones para los pines B4 y B3 se les configura el **GPIO output level** en HIGH y al Pin B5 se le activa la resistencia de pull-up que viene en la tarjeta estas configuración se ve en las figuras 1 y 2. Finalmente, para es terminar las configuraciones iniciales se hacer la respectiva configuración del reloj.

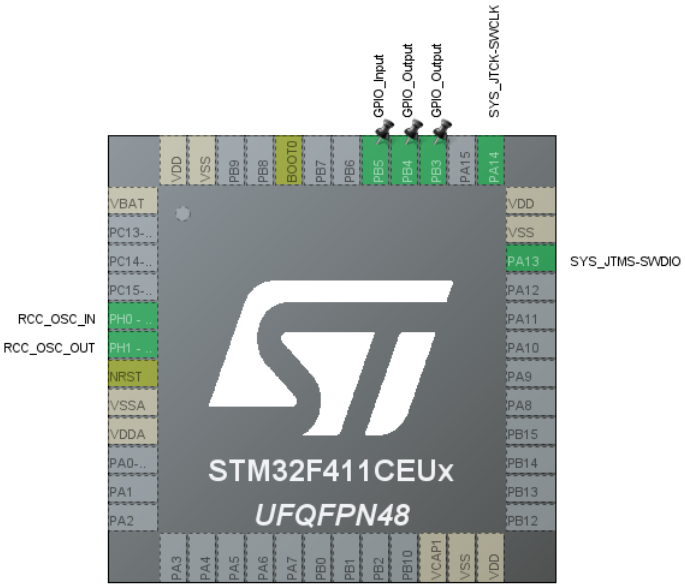

Figure 1: Configuración pines desde el Microprocesador.

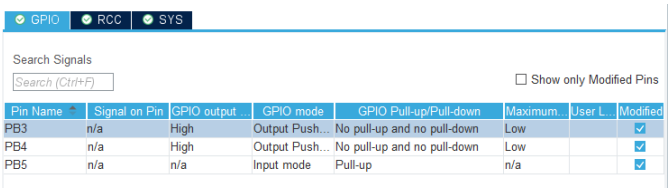

Figure 2: Configuración pines GPIO.

Posteriormente a la configuración de los pines se guardan los cambios y se genera el **código C** una vez generado el código ”general” el cual incluye librerías como:

- **HAL**: La cual nos permite acceder a los registros GPIO del microprocesador sin la necesidad de sabernos la posiciones de memoria que están asignadas a estos registros, además esta librería también nos permitirá encender, apagar y leer el estado del pulsador, para esta práctica de laboratorio se hizo uso de las siguientes **librerias HAL**
  - HAL_GPIO_WritePin().
  - HAL_Delay().
  - HAL_GPIO_ReadPin().
  - HAL_Init().
- **SystemClocl_Config:** Esta librería tiene las configuraciones del reloj que se hicieron previamente.

Una vez teniendo todo esto se procedió a realizar el siguiente código el cual se divide en 3 partes principales las cuales son

- **Declaracion de variables**

```
#include "main.h"
int cont = 0;
int contPress = 0;
int contTiempo = 0;
GPIO_PinState ultimoEstadoBoton = GPIO_PIN_RESET;
GPIO_PinState estadoBoton;
```

- **Logica Contador**

```
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
HAL_Delay(3);
cont++;
estadoBoton = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_5);
if((estadoBoton == GPIO_PIN_RESET && ultimoEstadoBoton == GPIO_PIN_RESET ) &&
    ((cont ≥ 0 && cont ≤99) || (cont > 100 && cont ≤199) ||(cont > 200 && cont ≤ 299))){

    if (contTiempo < 3 && ((cont ≥ 30 && cont ≤ 55) || (cont ≥ 115 && cont ≤ 260))) {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
        HAL_Delay(3);
        cont++;
        contPress++;
        contTiempo = 0;
        cont = 0;
    }

}else if((cont == 100 || cont == 200 || cont == 300 ) && estadoBoton == GPIO_PIN_SET ){
    contTiempo++;
}
```

- **Logica visualizacion pulso led**

```
if(contTiempo == 3){
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
    for(int i = 0; i < contPress; i++){
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_Delay(1000);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
        HAL_Delay(1000);
    }
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
    contTiempo = 0;
    contPress = 0;
    cont = 0;
}
```

Finalmente, en la figura 3 se muestras un modelo de montaje electrónico realizado para la práctica de laboratorio.

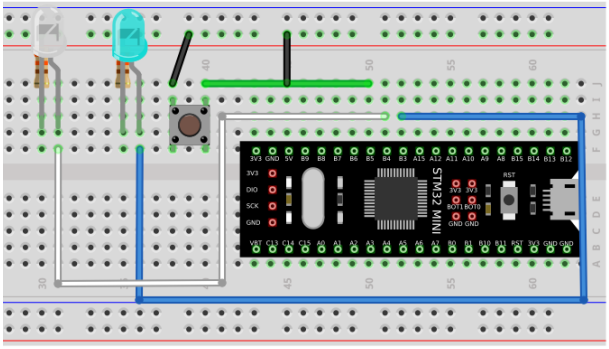

Figure 3: Configuración pines GPIO.

IV. Resultados y análisis de resultados

Este apartado del laboratorio se dividirá en las mismas partes en las cuales fue dividido el código mencionado anteriormente en el procedimiento.

- **Declaracion de variables** Las variables usadas en el código son las siguientes:
  - int cont: Esta variable será la encargada de manejar el ”tiempo” que usa el sistema que va de 0 a 300

- int contPress: Esta variable se encarga de almacenar el número de pulsaciones que ingresa el usuario, el rango de esta variable va de 0 n donde n son el número de pulsaciones ingresadas por el usuario.
- int contTiempo: Esta variable se encarga de contar el tiempo que transcurre sin ser pulsado el botón, el rango de esta variable va de 0 a 3.
- GPIO_PinState ultimoEstadoBoton: Esta variable tendrá el valor por defecto **GPIO_PIN_RESET**
- GPIO_PinState estadoBoton: esta variable es la encargada de almacenar el estatus que retorna la función **HAL_GPIO_ReadPin()** del pin B5, esta variable puede tomar valores como **GPIO_PIN_RESET** y **GPIO_PIN_SET**

- **Logica Contador:** Para esta apartado se le da un delay de 3ms para que el contador no avance de una manera acelerada, se hizo uso de esto ya que no se hizo uso de los registros **TIM** y **PREESCALER**, en esta parte del código el PIN B4 siempre permanece apagado ya que se está esperando a que los valores de los contadores **contTiempo** y **contPress** para saber si el pulsador fure presionada se realizas la siguiente validación.

```
estadoBoton = ...
    HAL_GPIO_ReadPin(GPIOB, ...
    GPIO_PIN_5);
if((estadoBoton == ...
    GPIO_PIN_RESET && ...
    ultimoEstadoBoton == ...
    GPIO_PIN_RESET ) &&
        ((cont ≥ 0 && cont ≤99) ...
            || (cont > 100 && ...
            cont ≤199) ||(cont > ...
            200 && cont ≤ 299))){
}
```

Esta validación se encarga primero de ver si el estado actual del pulsador es igual al último estado de este mismo, adicionalmente se maneja unos rangos de tiempo los cuales son.

$$\begin{bmatrix} cont, & >= 0 & <= 99 \\ cont, & > 100 & <= 199 \\ cont, & > 2000 & <= 299 \end{bmatrix}$$

En estos tres rangos de valores para **cont** es donde se puede incrementar el contador **contPress**, adicionalmente antes de realizar el aumento de esta variable contador se hace una segunda validación la cual es la siguiente.

```
if (contTiempo < 3 && ((cont ≥ ...
    30 && cont ≤ 55) || (cont ≥ ...
    115 && cont ≤ 260))) {
        HAL_GPIO_WritePin(GPIOB, ...
            GPIO_PIN_3, ...
            GPIO_PIN_SET);
        HAL_Delay(3);
        HAL_GPIO_WritePin(GPIOB, ...
            GPIO_PIN_3, ...
            GPIO_PIN_RESET);
        HAL_Delay(3);
        cont++;
        contPress++;
        contTiempo = 0;
        cont = 0;
}
```

En la cual se revisa que la variable contador **contTiempo** se a menor a 3, esto se realiza debido a que si dicha variable tiene un valor de 3 se contara como que el usuario dejo de pulsar el botón, adicionalmente el contador **contTiempo** solo aumenta en múltiplos de 100 del contador **cont** por esta razón el contador **contPress** no puede aumentar cuando cont vale [100, 200 y 300]. Finalmente, esta validación también cuenta con unos rangos los cuales son.

- **rango rápido** este rango es de corto de alcance ya que solo va de 30 a 55 este rango se diseñó para cuando el usuario oprima el pulsador de manera rápida.
- **rango lento** este rango es de lago de alcance ya que va de 115 a 260 este rango se diseñó para cuando el usuario oprima el pulsador de manera más pausada igual se puedan tomar su pulsaciones de manera correcta.

- **Lógica visualización pulso led:** Esta parte de la lógica solo se ejecuta cuando el contador **contTiempo** es igual a 3 lo que quiere decir que el usuario no pulso más el botón, en este apartado solo se usa el led del PIN B4 para mostrar el número de pulsaciones que almaceno el contador **contPress** para cada apartado de encendido y apagado de este led se le da un segundo encendido y un segundo apagado para que se puede apreciar bien el número de veces que el led es encendido. Finalmente, cuando se termina el ciclo se limpian las variables **cont**, **conTiempo** y **contPress**, esto también pasa cuando se aumenta el ultimo contador mencionado exceptuando que este mismo no es vaciado al momento de este aumentar.

V. CONCLUSIONES

- Si se hubieran usado los registros **TIM** y **PREESCALER** se hubiera podido tener una forma más fácil de medir el tiempo registrado.
- Con la ayuda del registro **TIM** se puedo haber adquirido de una manera más sencilla las veces que el pulsador fue presionado.
- Otra manera de poder haber contado el número de pulsaciones pudo haber sido por medio de una **IRC** la cual se hubiera disparado al momento de presionar el pulsador.
- Se debe hacer uso de un delay muy pequeño pero no menor a tres milisegundos para la captura de datos, permitiendo así una adquisición casi instantánea de datos sin que el usuario perciba retrasos significativos pero sin que la precisión del delay no sea incorrecta.

REFERENCES

[1] D. Thurow. Getting started with stm32 black pill - getting blinky with stm32cube ide. [Online]. Available: https://daniellethurow.com/blog/2021/6/16/getting-started-with-stm32-black-pill-getting-blinky-with-stm32cube-ide

[2] L. augmented. Description of stm32f4 hal and low-layer drivers. [Online]. Available: https://www.st.com/resource/en/user_manual/um1725-description-of-stm32f4-hal-and-lowlayer-drivers-stmicroelectronics.pdf
[3] D. Bennett. Getting started with stm32 blackpill and stm32cubeide and usb cdc serial. [Online]. Available: https://www.bennettnotes.com/notes/stm32-blackpill-with-stmcubeide-usb-serial/

VI. ANEXOS