UNIVERSIDAD NACIONAL DE COLOMBIA

# Ejercicio
# Control LED


## Alejandro Ramírez Jaramillo

*Universidad Nacional de Colombia sede Manizales Email: alramirezja@unal.edu.co*


## Problema

El usuario interctua con el controlador al presionar un botón. La luz tiene tres niveles de intensidad: OFF, DIMMED y BRIGHT, dependiendo del tiempo entre toques sucesivos, el controlador intercambia entre estos niveles de luz. Por ejemplo en estado dimerizado, si el segundo toque se hace rapidamente (antes del tiempo de switcheo Tsw=4 unidades de tiempo) después del toque que causó que el controlador entrara en estado dimerizado (ya sea desde el estado OFF o BRIGHT), el controlador aumenta el nivel a BRIGHT. Por el contrario, si el secundo toque sucede después del tiempo de switcheo, el controlador apaga la luz. Si el controlador de luz ha estado en estado OFF por mucho tiempo (mayor o igual a Tidle=20 unidades de tiempo), se deberá reactivar yendo directamente al nivel BRIGHT.

Se requiere controlar un led haciendo uso de un botón para un sistema de iluminación, el cual tiene 3 estados: Apagado, encendido y dimerizado, estos estados cambiarán cuando se presione el botón, sin embargo la secuencia de cambio estará determinada por el estado actual y el tiempo entre cada cambio de estados.

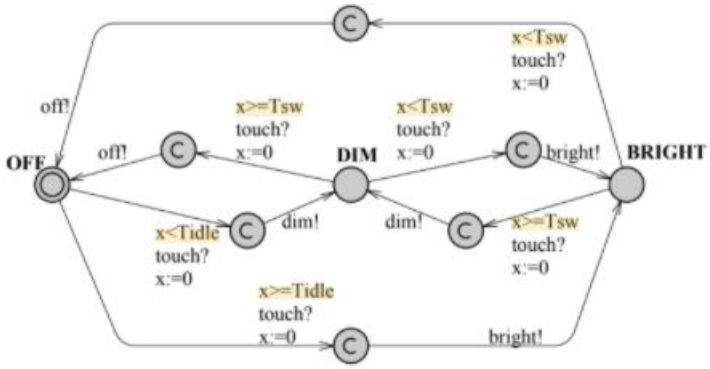
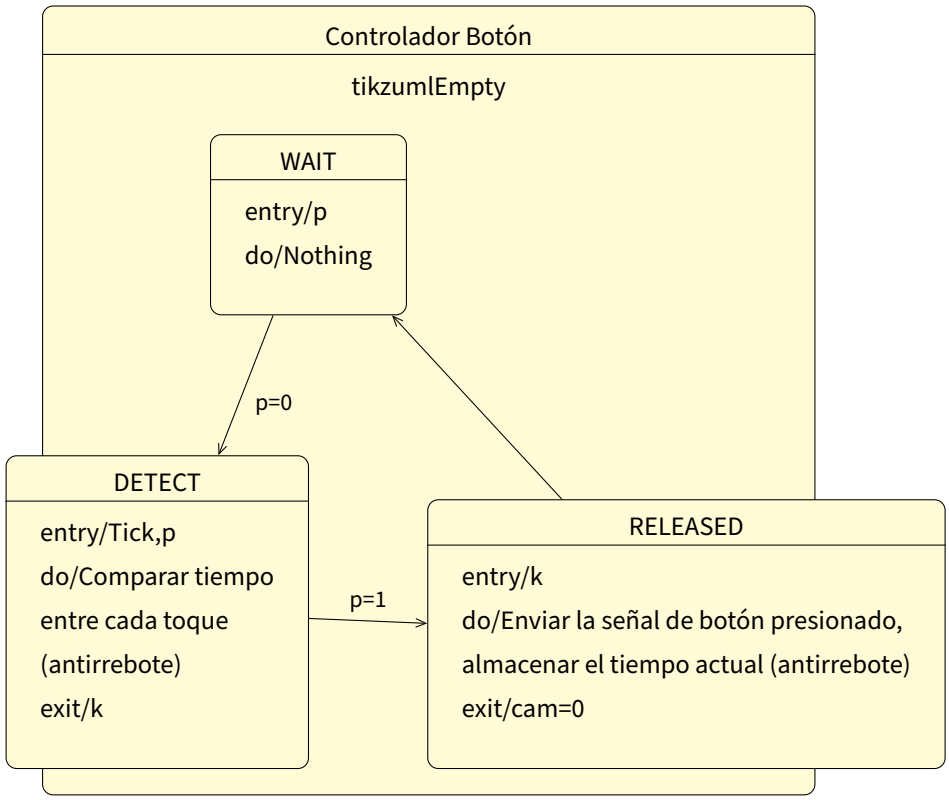



**Figure 1.** Máquina de estados control de iluminación

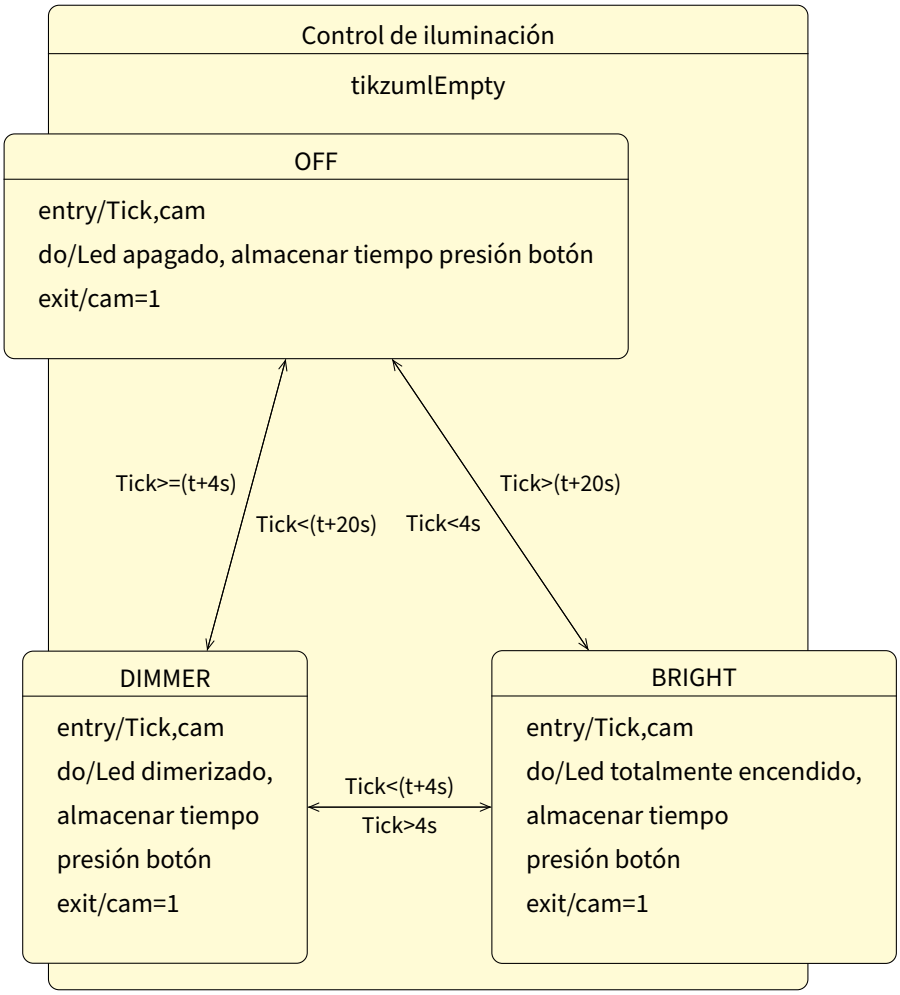
# 1 Solución

## 1.1 Máquina de estados botón




- WAIT: Estado de espera, el botón no ha sido presionado, por lo tanto se esta leyendo su estado para detectar cuando pase.
- DETECT: El botón ha sido presionado, pero aún no se libera. Cuando se implementa el antirebote se aplica en este estado, comparando el tiempo entre cada detección, solo se tiene en cuenta la detección si han pasado 50 milisegundos o más entre cada una.
- RELEASED: El botón ha sido liberado, entonces se pondrá una variable en 0 que indicará que ocurrió el evento y puede ejecutarse el cambio de estados en el controlador del led, si se ha implementado el antirebote solo se pondrá la variable en cero si se cumplió la condición de tiempo en el estado DETECT.

## 1.2 Máquina de estados control de luz




- OFF: Estado en el que comienza el sistema, el led esta apagado, cuando se presiona el botón se compara el tiempo actual con la última vez que se cambió de estado, según el resultado se pasa al estado DIMMER o BRIGHT.
- DIMMER:El led esta dimerizado, cuando se presiona el botón se compara el tiempo actual con la última vez que se cambió de estado, según el resultado se pasa al estado OFF o BRIGHT.
- Entrada/Salida:El led esta totalmente encendido, cuando se presiona el botón se compara el tiempo actual con la última vez que se cambió de estado, según el resultado se pasa al estado OFF o DIMMER.

## 1.3 Código

Para el ejercicio se usará: Las GPIO y una base de tiempo generada por el mismo sistema, las máquinas de estados del control de botón y el led, las cuales se guardarán en archivos .c y .h distintos al archivo principal del código para ser incluidas en este, de manera que en el ejercicio se trabajará en 4 archivos: main.c, main.h, fsm.c y fsm.h, siendo los archivos fsm donde se encuentran las máquinas de estados finitos.

En primer lugar se explicará la fuente de la base de tiempo, esta es creada por el HAL y genera un tick cada milisegundo, este no puede ser modificado por el usuario, solo leído por una función declarada dentro del archivo stm32l4xx_hal.c

Es una función simple, pero retorna el valor de un contador que se ha estado aumentando en 1 cada milisegundo desde que se inicia el sistema. Usaremos este valor retornado para marcar los

tiempos necesarios en el programa.

```
con=HAL_GetTick();        //Lea el tiempo actual
```

El archivo main.c se puede programar de varias maneras para implementar un round robin, en esta guía se van a explicar 2 métodos: manual y con hilos, ambos casos funcionarán correctamente pero el método con hilos se relaciona más con sistemas operativos.

### 1.3.1 main.c (manual)

En este archivo se llamarán las funciones que controlan el botón, el Led y se configurarán las GPIOs que se usarán (mediante el HAL).
En primer lugar se incluye el archivo main.h, donde a su vez esta incluido fsm.h, gracias a esto las funciones de los controladores del Led y el botón se podrán llamar en el archivo en el que estamos trabajando.

```
#include "main.h"
```

Después se configuran las GPIO, se configuró el pin 13 del puerto C como una entrada pues el botín está conectado a este y se configura el pin 5 del puerto A como salida ya que esta conectado al led de la tarjeta.

Entonces en la sección principal se inicia el HAL junto con el reloj del sistema y las GPIO.

```
HAL_Init();

SystemClock_Config();

MX_GPIO_Init();
```

Dentro de un while se llaman a las funciones del controlador botón y controlador led para que se ejecuten indefinidamente. Sin embargo se hará uso de la base de tiempo para que esta ejecución se haga cada un milisegundo, para ello usamos de la función HAL_GetTick(), el programa solo se ejecutará cada vez que esta aumente en uno (1 milisegundo).

```
if (HAL_GetTick()>marca){//Si ha pasado un milisegundo desde la ultima ejecucion
  BotonHandler();
  LedHandler();
  marca=HAL_GetTick();  //Almacene el tiempo actual
}
```

### 1.3.2 main.c (Hilos)

En este archivo se llamarán las funciones que controlan el botón, el Led y se configurarán las GPIOs que se usarán (mediante el HAL).
En primer lugar se incluye el archivo main.h, donde a su vez esta incluido fsm.h, gracias a esto las funciones de los controladores del Led y el botón se podrán llamar en el archivo en el que estamos trabajando.

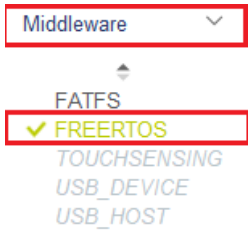
```
#include "main.h"
```

Después se configuran las GPIO, se configuró el pin 13 del puerto C como una entrada pues el botín está conectado a este y se configura el pin 5 del puerto A como salida ya que esta conectado al led de la tarjeta.

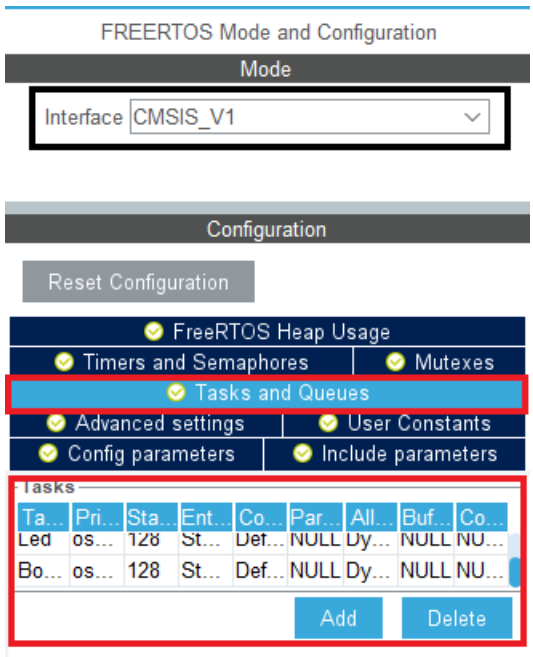
Entonces en la sección principal se inicia el HAL junto con el reloj del sistema y las GPIO.

```
HAL_Init();

SystemClock_Config();

MX_GPIO_Init();
```

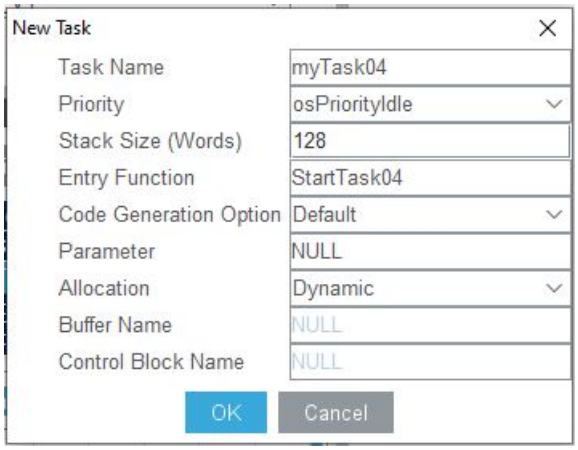
Hasta este punto todo ha sido similar al método manual, a partir de este punto se comienza a crear los hilos. Para comenzar se va a usar el HAL para crear las tareas, para hacer esto entramos a la interfaz del HAL, entramos a la sección de Middleware y seleccionamos FreeRTOS.




Al hacer eso se desplegará una ventana en la que podremos comenzar a configurarlo y crear las tareas. En primer lugar se escoge el modo en la parte superior, basta con seleccionar en la sección de interfase (cuadro negro) la opción CMSIS_V1, con lo cual se desbloqueará la sección de configuración.




Después se entrará en la pestaña de Tasks and Queues (cuadro rojo) para crear las tareas usando el botón Add (añadir) para agregar una nueva tarea, al hacerlo se desplegará una ventana como la que se muestra a continuación donde se configurará la tarea y se le dará un nombre, para finalizar se ejecutan los cambios en la HAL.




El HAL va a crear las funciones que darán comienzó a los hilos igual que lo hizo con las funciones que configuran las GPIO, pero en este caso va a haber una por cada tarea creada (Boón y Led).

```
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void StartDefaultTask(void const * argument);
void StartLed(void const * argument);
void StartBoton(void const * argument);
```

Después dentro de la sección principal el HAL va a definir y crear cada hilo (Tarea) y darle un nombre a su manejador, cuando ya todos hayan sido creados se da inicio a la ejecución.

```
  /* definition and creation of Led */
  osThreadDef(Led, StartLed, osPriorityIdle, 0, 128);
  LedHandle = osThreadCreate(osThread(Led), NULL);

  /* definition and creation of Boton */
  osThreadDef(Boton, StartBoton, osPriorityIdle, 0, 128);
  BotonHandle = osThreadCreate(osThread(Boton), NULL);

  osKernelStart();
```

Al dar inicio el microcontrolador comenzará a ejecutar las tareas desde las funciones que se declararon al inicio de manera sucesiva y repetitiva, es decir haciendo un round robin. dentro de los hilos se llaman a las funciones que controlan el led y el botón para que se ejecuten en el round robin.

```
void StartLed(void const * argument)
{
  /* USER CODE BEGIN StartLed */
          uint32_t count = 0;
          (void) argument;
  /* Infinite loop */
  for(;;)
  {
            count = osKernelSysTick() + 10;


            while (count > osKernelSysTick())
            {
                 LedHandler();
            }

  }
  /* USER CODE END StartLed */
}

void StartBoton(void const * argument)
{
  /* USER CODE BEGIN StartBoton */
  /* Infinite loop */
```

```
          uint32_t count = 0;
          (void) argument;
/* Infinite loop */
for(;;)
{
            count = osKernelSysTick() + 10;


            while (count > osKernelSysTick())
            {
                BotonHandler();
            }

}
  /* USER CODE END StartBoton */
}
```

### 1.3.3 fsm.c

En este archivo se crearán las máquinas de estado que controlarán el led y el botón, ambas se harán usando estructuras distintas.
Include:En main se encuentra la función de la que se obtiene el tick de un milisegundo y en fsm.h se declaran las funciones que sea crearán en este archivo.

```
#include "main.h"
#include "fsm.h"
```

Y se crearán las variables que se usarán en las máquinas de estado:

```
uint32_t tsw=20000;                    //Tiempo limite para decidir los cambios de estado del
    led
uint32_t t_led=0;                      //Tiempo de la ultima transici n de estados led
uint32_t start=1;                      //Bandera que permite indicar el estado en el que
    comienzan los controladores

uint32_t t_bot=0;                      //Tiempo de la ultima detecci n del bot n
uint32_t event=1;                      //Bandera que indica que se presiono y libero el bot n
uint32_t cont=0;                       //# de veces que se ha presionado en bot n
uint32_t dim=0;                        //Bandera que indica si se dimeriza o no el led
uint32_t p;                                    //Almacena el estado del bot n
static uint32_t con=0;          //Almacena el tiempo, aumenta cada milisegundo
```

Controlador Botón: Es la función que administrará el botón, esta tiene tres estados y cambiará entre ellos según detecte los cambios en el botón de la tarjeta.

```
enum states2 {RELEASED,WAIT,DETECT} boton_state;        //Estados controlador boton
uint32_t volatile antire=0;                     //Contador que indica cuando se cumplio el
    antirrebote
void BotonHandler(void){
        if (start==1){                  //Cuando la funcion se ejecuta por primera vez:
                boton_state=WAIT;       //Se pone el estado en OFF
        }

        p=HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_13);  //Se lee el estado del boton
```

Estados:

```
        switch(boton_state){    //Cambio de estados boton
                case DETECT:            //Se detecto un touch
                        switch (p){             //Si el boton esta presionado
                        case(0):
                                antire+=1;      //Aumente el contador en 1
                                break;
                        case (1):               //Si el boton no esta presionado
                                if (antire>50){//Si el boton paso mas de 50 ms presionado (no
                                    es un rebote)
                                    boton_state=RELEASED;   //pase al estado liberado
                                }else{boton_state=WAIT;}        //Si no pase al estado de
                                    espera
                                break;
                        }
```

```
            break;
        case RELEASED://El boton fue presionado y liberado
            event=0;        //Se envia el evento al controlador del led
            antire=0;       //Se reinicia la variable del antirrebote
            boton_state=WAIT;//Se pasa al estado de espera
            break;
        case WAIT:              //Espera a detectar un touch
            switch (p){
            case(0)://Si el boton es presionado pase al estado detect
                    boton_state=DETECT;
                    break;
            case (1)://Si no esta presionado no haga nada
                    break;
            }
            break;
        default:

            break;

}
```

Controlador led: Es la función que administrará el led, esta tiene tres estados y cambiará entre ellos según 3 eventos que dependerán del tiempo en que se presione el botón de la tarjeta.

```
enum states {OFF,DIM,BRIGHT} led_state; //Estados del controlador led
enum events{tswm,tswp,NC} new_event;            //Eventos del controlador del led
```

Teniendo en cuenta esto, se crea una tabla de estados y eventos, la cual determinará que acciones tomar cuando se presenta un evento estando en un estado específico.

```
#define MAX_STATES 3
#define MAX_EVENTS 3
typedef void (*transition)();

transition state_table[MAX_STATES][MAX_EVENTS] = {
                {dimer,brill,error}, //OFF
                {brill, apag, error},//DIM
                {apag, dimer, error}//BRIGHT
};
```

Estados:

- OFF: Led apagado.
- DIM: Led dimerizado.
- BRIGHT: Led completamente encendido.

Eventos:

- tswm: Se presionó el botón en un tiempo menor al establecido, este tiempo se cuenta desde el último cambio de estado.
- tswp: Se presionó el botón en un tiempo mayor o igual al establecido, este tiempo se cuenta desde el último cambio de estado.
- NC: No se ha presionado el botón y por lo tanto nada cambia.

El tiempo establecido depende del estado en el que se encuentra el Led: son 20 segundos para el estado OFF y 4 segundos para el estado DIM y BRIGHT. Los eventos se determinan según 2 factores: el tiempo entre el cambio de estados y si se presionó o no el botón, si el botón si fue presionado (cam==0) entonces se mide el tiempo actual y se determina si es mayor o menor al limite, si no fue presionado entonces nada cambia.

```
if (event==0){  //Si se presiono el boton
        event=1;        //Limpie la bandera del boton
        con=HAL_GetTick();      //Lea el tiempo actual
        if(con<(t_led+tsw)){    //Si ha transcurrido menos de un tiempo limite
            entre
                new_event=tswm;       //cada cambio de estado, presente el
                    evento
          }
        else{
                 new_event=tswp;
```

```
            }
        }
        else{new_event=NC;}             //Nada cambio
```

Cuando se ha determinado el evento es momento de escoger las acciones que se llevarán a cabo: Se revisa que el evento y el estado actual estén dentro de los los creados para la máquina de estados, si cumple esto ejecuta la tabla de estados:

```
        if ((new_event >= 0) && (new_event < MAX_EVENTS)        //El evento actual esta
              entre los eventos que se crearon
                    && (led_state >= 0) && (led_state < MAX_STATES)) {//El estado
                         actual esta entre los eventos que se crearon
        /* call the transition function */
                state_table[led_state][new_event]();//Se ejecuta la tabla de estados
                     usando el evento y estado actual
        }
        else {
        /* invalid event/state - handle appropriately */
    }
```

La tabla de estados tiene tantas filas como estados y tantas columnas como eventos, se puede asumir entonces que cada fila representa un estado y cada columna un evento. Se organizan de primera a última fila/columna según fueron declaradas, esto es: fila 1=OFF, fila 2=DIM, columna 1=tswm, columna 2=tswp. Cuando se sepa cuál es el evento y el estado actual, se ubica en la tabla que acciones se deben ejecutar.

```
transition state_table[MAX_STATES][MAX_EVENTS] = {
                {dimer,brill,error}, //OFF
                {brill, apag, error},//DIM
                {apag, dimer, error}//BRIGHT
};

void apag(){    //Accion de apagar el led
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);
        tsw=20000;      //Establece el tiempo limite en 20s
        led_state=OFF;
        t_led=HAL_GetTick();    //Guarda el tiempo actual
        dim=0;
}
void dimer(){   //Accion de dimerizar el led
        dim=1;
        led_state=DIM;
        t_led=HAL_GetTick();    //Guarda el tiempo actual
        tsw=4000;
}
void brill(){//Accion de encender el led
        dim=0;
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);//Se enciende el led
        led_state=BRIGHT;               //Se pasa al estado BRIGHT
        t_led=HAL_GetTick();
        tsw=4000;
}
void error(){dimerizar(dim);}//Si no ha cambiado nada, revise si debe dimerizar
      el led
```

Todas la acciones (con excepción de error()) almacenan el tiempo actual para la próxima comparación de la máquina de estados.

- apag(): Apaga el led, cambia el estado actual a OFF y establece el tiempo limite en 20 segundos.
- dimer(): Salida del led dimerizada y pasa a estado DIM, establece el tiempo limite en 4 segundos.
- brill(): Enciende completamente el led, pasa al estado BRIGHT y establece el tiempo limite en 4 segundos.
- error(): No ocurrió nada así que no se cambia de estado, si esta en estado dimerizado se ejecuta dimerizar.

### 1.3.4 fsm.h

En este archivo se declaran las funciones que se crearon en el .c, es este el que se va a incluir el el main.c para poder usar las funciones de fsm en el archivo principal.

```
#ifndef INC_FSM_H_
#define INC_FSM_H_
#include "stm32l4xx_hal.h"

void BotonHandler(void);
void LedHandler(void);
void dimerizar(int);
#endif /* INC_FSM_H_ */
```

### 1.3.5 main.h

En este archivo se incluyen el .h de las máquinas de estados y un .h generado por el HAL con multiples funciones, como por ejemplo GetTick().

```
#include "stm32l4xx_hal.h"
#include "fsm.h"
```

## 2 Antirrebote

Para evitar el rebote causado por el botón de la tarjeta se implementa un antirrebote por software, así evitando que se generen más eventos de presión del botón de los que realmente ocurrieron. Para comprobar su funcionamiento cada vez que se presiona el botón y se libera, aumentamos en 1 una variable, sin importar cuantas veces se haga el número de veces que se presionó el botón debe ser igual al valor en el contador.
Para comparar el funcionamiento del código con antirrebote y si antirrebote, se va a ejecutar ambas versiones y buscar las diferencias en el resultado.

### 2.0.1 Con antirrebote

Presionar 10 veces:

| Expression | Type | Value |
|---|---|---|
| (x)= cont | uint32_t | 10 |

Presionar 50 veces:

| Expression | Type | Value |
|---|---|---|
| (x)= cont | uint32_t | 50 |

Presionar 150 veces:

| Expression | Type | Value |
|---|---|---|
| (x)= cont | uint32_t | 150 |

### 2.0.2 Sin antirrebote

Presionar 10 veces:

| Expression | Type | Value |
|---|---|---|
| (x)= cont | uint32_t | 10 |

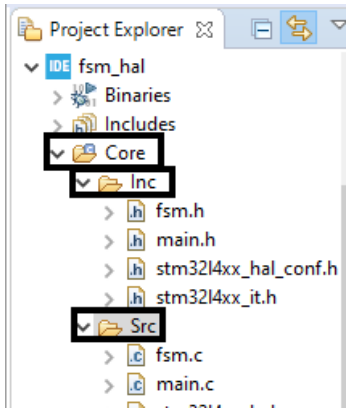
Presionar 100 veces:

| Expression | Type | Value |
|---|---|---|
| (x)= cont | uint32_t | 101 |

Como podemos ver, cuando el código tiene antirrebote el contador siempre sigue al número de veces que sea presionado el botón, por otro lado cuando no se usa integra un antirrebote el contador va a ser mayor que el número de veces que se presione, lo cual provocará que el programa ejecute más eventos de los que debería.

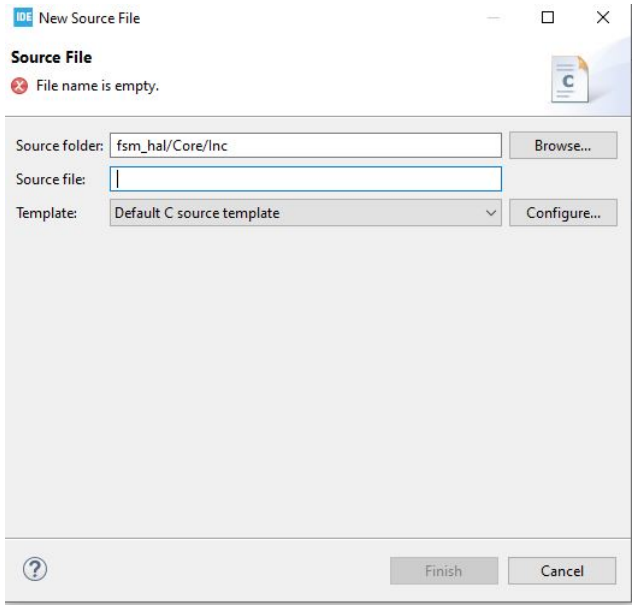
## 3 Creación de nuevos archivos .c y .h

En caso de crear nuevas máquinas de estado en otros archivos siguiendo la estructura de este ejemplo, se deben crear nuevos archivos .c y .h, lo cual se hace de la siguiente manera:
Primero se selecciona la carpeta en donde se quieren crear en el Project Explorer, para este caso se debe seleccionar (click izquierdo) la carpeta Src para los .c e Inc para los .h, ambas carpetas están dentro de la carpeta Core:




Después se presiona el botón de crear un nuevo archivo fuente y se escoge si va a ser .c o .h:


Lo que sigue es igual para ambos tipos de archivo, se abre una ventana con la dirección donde se va a crear el archivo y un espacio para ingresar el nombre, es importante recordar que es necesario agregar .h o .c al final del nombre para que se cree correctamente.




Cuando se haya hecho esto se presionas Finish y el archivo es creado.

# 4 Referencias

+ PM0214 Programming manual, STM32 Cortex-M4 MCUs and MPUs programming manual.
+ RM0351 Reference manual, STM32L4x5 and STM32L4x6 advanced Arm-based 32-bit MCUs.
+ STM32l476rg datasheet.
+Time-optimal Real-Time Test Case Generation using UPPAAL
+Código en Github: fsm_hal