Práctica 4

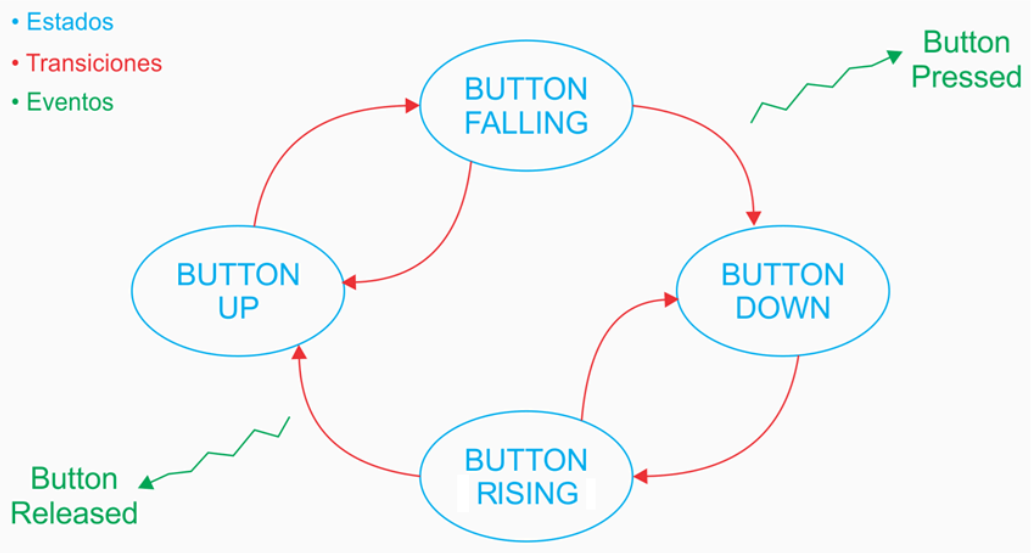
Objetivo:

Implementar un MEF para trabajar con anti-rebotes por software.

Punto 1

Crear un nuevo proyecto como copia del proyecto realizado para la práctica 3.

Implementar una MEF anti-rebote que permita leer el estado del pulsador de la placa NUCLEO-F4 y generar acciones o eventos ante un flanco descendente o ascendente, de acuerdo al siguiente diagrama:



El estado inicial de la MEF debe ser BUTTON_UP.

Implementar dentro de main.c, las funciones:

```
void debounceFSM_init();        // debe cargar el estado inicial
void debounceFSM_update();      // debe leer las entradas, resolver la lógica de
                                // transición de estados y actualizar las salidas
void buttonPressed();           // debe encender el LED
void buttonReleased();          // debe apagar el LED
```

El tiempo de anti-rebote debe ser de 40 ms con un retardo no bloqueante como los implementados en la práctica 3.

La función debounceFSM_update() debe llamarse periódicamente.

```
typedef enum{
	BUTTON_UP,
	BUTTON_FALLING,
	BUTTON_DOWN,
	BUTTON_RAISING,
} debounceState_t
```

Punto 2

Implementar un módulo de software en un archivo fuente **API_debounce.c** con su correspondiente archivo de cabecera **API_debounce.h** y ubicarlos en el proyecto dentro de las carpetas /drivers/API/src y /drivers/API/inc, respectivamente.

En API_debounce.h se deben ubicar los prototipos de las funciones públicas y las declaraciones:

```
void debounceFSM_init();
void debounceFSM_update();

/* La función readKey debe leer una variable interna del módulo y devolver true o false si la tecla fue presionada.  Si devuelve true, debe resetear (poner en false) el estado de la variable.*/
bool_t readKey();
```

En API_debounce.c se deben ubicar las declaraciones privadas, los prototipos de las funciones privadas y la implementación de todas las funciones del módulo, privadas y públicas:

La declaración de debounceState_t debe ser privada en el archivo .c y la variable de estado de tipo debounceState_t debe ser global privada (con static).

Declarar en API_debounce.c una variable tipo bool_t global privada que se ponga en true cuando ocurre un flanco descendente y se ponga en false cuando se llame a la función readKey();

Implementar un programa que cambie la frecuencia de parpadeo del LED entre 100 ms y 500 ms cada vez que se presione la tecla. El programa debe usar las funciones anti-rebote del módulo API_debounce y los retardos no bloqueantes del módulo API_delay y la función readKey.

Para pensar luego de resolver el ejercicio:

- ¿Es adecuado el control de los parámetros pasados por el usuario que se hace en las funciones implementadas? ¿Se controla que sean valores válidos? ¿Se controla que estén dentro de los rangos correctos?

- ¿Se nota una mejora en la detección de las pulsaciones respecto a la práctica 0? ¿Se pierden pulsaciones? ¿Hay falsos positivos?

- ¿Es adecuada la temporización con la que se llama a debounceFSM_update()? ¿Y a readKey()? ¿Qué pasaría si se llamara con un tiempo mucho más grande? ¿Y mucho más corto?