UNIVERSIDAD TECNOLOGICA NACIONAL

FACULTAD REGIONAL TUCUMAN

Actividad de Formación Práctica 4

Anti-rebote desarrollado por MEF para garantizar detección positiva de pulsadores mecánicos, aplicación en STM32CubeIDE, programación de microcontroladores.

Integrantes:

Castro Oscar Martin

Décima Enrique Emanuel

Ortiz Nicolas Agustín

Introducción

¿Qué es el rebote mecánico de los botones?

El rebote mecánico ocurre cuando un botón o interruptor mecánico se presiona o suelta. Durante el proceso, los contactos metálicos dentro del botón pueden vibrar y abrirse/cerrarse varias veces antes de asentarse en una posición estable. Esto genera múltiples pulsos eléctricos en lugar de uno solo, lo que puede ser interpretado erróneamente por un microcontrolador como varias pulsaciones. Esto puede causar lecturas erróneas, errores en caso de que realicemos un conteo mediante la activación del botón, y también algunos comportamientos erráticos de nuestra aplicación.

Para mitigar este problema hay soluciones que se implementan mediante hardware:

- Condensador en paralelo
- Circuito Schmitt Trigger
- Uso de Flip-Flop

Pero estas soluciones nos significan un costo adicional y aumentan la complejidad del proyecto, por lo que se opta por una solución mediante software.

Solución al rebote mecánico mediante el uso de MEF

Una de esas soluciones es el uso de una máquina de estados finita con 4 estados:

- **Button Up:** El botón esta sin presionar.
- **Button Falling:** el botón esta en su flanco de bajada, justo en el instante de ser presionado.
- **Button Down:** el botón esta presionado.
- **Button Rising:** el botón esta en su flanco de subida, en el instante en el que se suelta.

Ventajas del uso de MEF para controlar el rebote de botones

- Una máquina de estados garantiza que el botón pase por transiciones claras y controladas entre estados. Esto evita errores al gestionar los cambios de estado solo cuando son estables.
- Permite manejar múltiples botones y eventos complejos con facilidad, ya que cada botón puede tener su propia máquina de estados.
- A diferencia del uso de delays, las máquinas de estados permiten seguir ejecutando otras tareas mientras se espera a que el botón se estabilice. Esto es ideal en sistemas multitarea o de tiempo real.
- Una máquina de estados es fácil de extender para manejar más botones o para integrar más estados y transiciones sin afectar la lógica global del sistema.
- Es posible implementar funcionalidades avanzadas como:
    a. Pulsaciones largas
    b. Doble clic
    c. Combinaciones de botones

Creación de la librería debounce

En primera instancia, creamos la maquina de estados, esta se basa en una enumeración de los estados:

1- BUTTON_UP
2- BUTTON_FALLING
3- BUTTON_DOWN
4- BUTTON_RISING

Esta maquina de estados se llamará luego, mediante el tipo definido *debounceState_t.*

Luego declaramos la variable *debounce_timer* que guardara el tiempo entre lecturas que se establece en este caso en 40ms debido al tiempo de respuesta promedio de una persona en presionar dos veces un botón.

Instanciamos la clase mediante *estado_btn* y creamos una variable booleana llamada *lectura_btn* que nos devolverá la lectura final del botón.

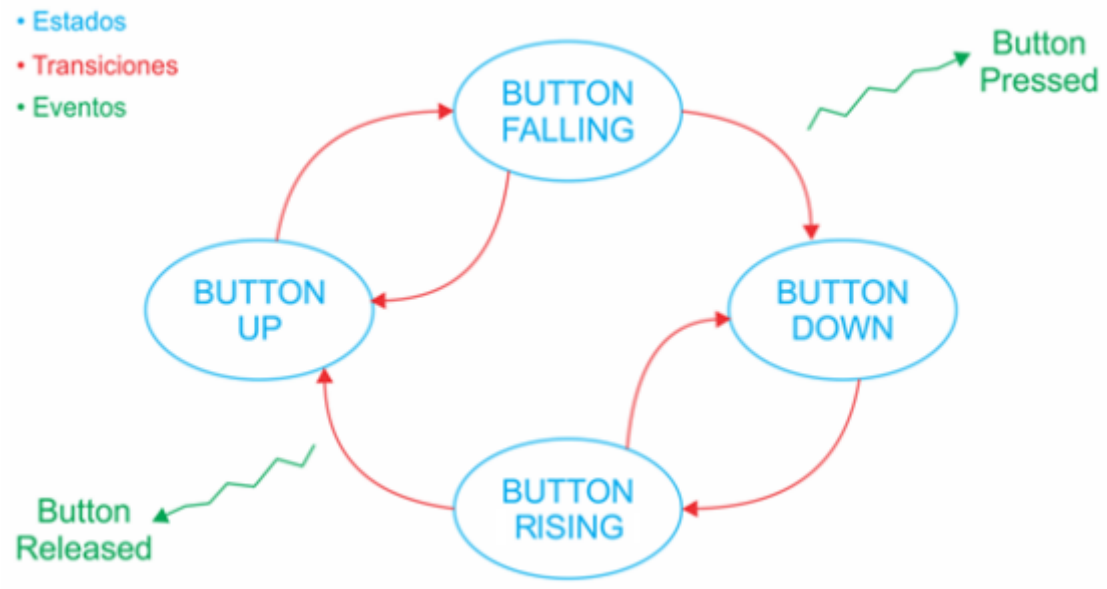
debounceFSM_Init

Esta función inicializa la maquina de estados, iniciando en el estado BUTTON_UP, asignando a la variable *debounce_timer* la duración de 40ms, apagando todos los leds, y estableciendo *lectura_btn* en *false.*

```
 *  Created on: Nov 6, 2024
 *      Author: Grupo 3_TD2     Enrique Emanuel Decima, Castro Oscar Martin, Ortiz Nicolas Agustin
 */

#include "main.h"
#include "API_debounce.h"
#include "API_Delay.h"

typedef enum {
        BUTTON_UP ,
        BUTTON_FALLING ,
        BUTTON_DOWN ,
        BUTTON_RISING
} debounceState_t;

delay_t debounce_timer;         //Delay del antirrebote
debounceState_t estado_btn;     //Instancicion de la maquina de estado
bool_t lectura_btn;                     //Lectura del boton a devolver al usuario


/*
 * @brief Inicializacion de la MEF, establece la duracion del antirebote
 * @param None
 * @retval None
 */
void debounceFSM_Init(){
        delayInit(&debounce_timer, 40);
        estado_btn = BUTTON_UP;
        lectura_btn = false;
        writeLedOff_GPIO(LD1_Pin);
        writeLedOff_GPIO(LD2_Pin);
        writeLedOff_GPIO(LD3_Pin);
}
```

debounceFSM_Update

Esta función controla la maquina de estados en si, recibe el estado mediante *estado_btn*, lee continuamente la señal de entrada del botón mediante *readButton_GPIO()*, y pregunta si ha transcurrido el periodo de 40ms, con esos datos decide cual es el estado siguiente de acuerdo a la siguiente representación:




```c
/*
 * @brief Actualizacion de los estados
 * @param None
 * @retval None
 */
void debounceFSM_Update(){
	switch(estado_btn){
	case BUTTON_UP:
		if(readButton_GPIO()){
			estado_btn = BUTTON_FALLING;
		}
		break;
	case BUTTON_FALLING:
		if(delayRead(&debounce_timer)){
			if(readButton_GPIO()){
				lectura_btn = true;
				estado_btn = BUTTON_DOWN;
			}else{
				estado_btn = BUTTON_UP;
			}
		}
		break;
	case BUTTON_DOWN:
		if(!readButton_GPIO()){
			estado_btn = BUTTON_RISING;
		}
		break;
	case BUTTON_RISING:
		if(delayRead(&debounce_timer)){
			if(!readButton_GPIO()){
				estado_btn = BUTTON_UP;
			}else{
				estado_btn = BUTTON_DOWN;
			}
		}
		break;
	default:
		Error_Handler();
		break;
	}
}
```

readKey

Esta función nos devuelve la lectura final del botón mediante una variable booleana que se activa exactamente en la transición entre BUTTON_FALLING y BUTTON_DOWN, y que se desactiva cada vez que se llama (para evitar dobles lecturas).

```
80  /*
81   * @brief Devuelve la lectura del boton
82   * @param None
83   * @retval bool_t :     true-> boton pulsado ; false -> boton soltado
84   */
85  bool_t readKey(){
86          if(lectura_btn){
87                  lectura_btn = false;
88                  return true;
89          }else{
90                  return false;
91          }
92  }
```

Enlaces

Repositorio grupal

EmanuelDecima/Grupo_3_TDII_2024

AFP 4

Grupo_3_TDII_2024/AFP_4_TDII_2024 at main · EmanuelDecima/Grupo_3_TDII_2024

App 4.1

Grupo_3_TDII_2024/AFP_4_TDII_2024/AFP_4_Grupo_3_App_4.1 at main · EmanuelDecima/Grupo_3_TDII_2024

App 4.2

Grupo_3_TDII_2024/AFP_4_TDII_2024/AFP_4_Grupo_3_App_4.2 at main · EmanuelDecima/Grupo_3_TDII_2024

App 4.3

Grupo_3_TDII_2024/AFP_4_TDII_2024/AFP_4_Grupo_3_App_4.3 at main · EmanuelDecima/Grupo_3_TDII_2024

App 4.4

Grupo_3_TDII_2024/AFP_4_TDII_2024/AFP_4_Grupo_3_App_4.4 at main · EmanuelDecima/Grupo_3_TDII_2024