Facultad Regional Tucumán

Departamento de Electrónica

Técnicas Digitales II

Informe

Actividad de Formación Practica N°4

Tema: Anti-rebote desarrollado por MEF. Aplicaciones en STM32CubeIDE. Programación de Microcontroladores

Profesor:
Ing. Rubén Darío Mansilla

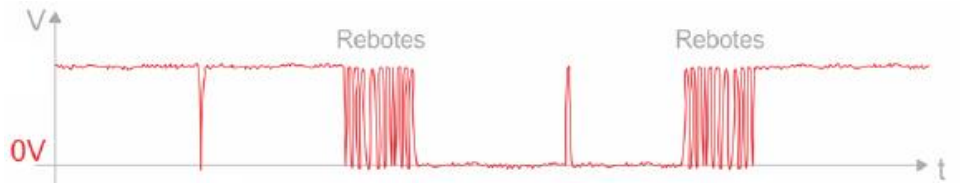
ATTP:
Ing. Lucas Abdala

Grupo 8:
Gao Luciano, Mitre Emilio, Jorrat Tomas.

Introducción: Implementación de funciones Antirebote utilizando máquinas de estado finito (MEF) con el entorno de desarrollo STM32CubeIDE.

En sistemas embebidos, el manejo eficiente de las entradas digitales, como botones o interruptores, es crucial para garantizar la estabilidad y confiabilidad de una aplicación. Un problema común al leer el estado de un botón es el fenómeno conocido como "rebote" (debounce), el cual ocurre cuando una pulsación genera múltiples transiciones de estado debido a la naturaleza mecánica del botón. Este comportamiento puede causar lecturas erróneas y resultados no deseados en el software.

Para resolver este problema, las funciones antirebote implementadas mediante Máquinas de Estado Finito (MEF) ofrecen una solución robusta y eficiente. Una MEF divide el proceso de gestión del botón en estados definidos (por ejemplo, BUTTON_UP, BUTTON_FALLING, BUTTON_DOWN, BUTTON_RISING) y establece transiciones claras entre ellos basadas en condiciones específicas y temporizadores. Este enfoque permite filtrar de manera determinista los rebotes al validar que una transición entre estados sea consistente durante un tiempo definido.



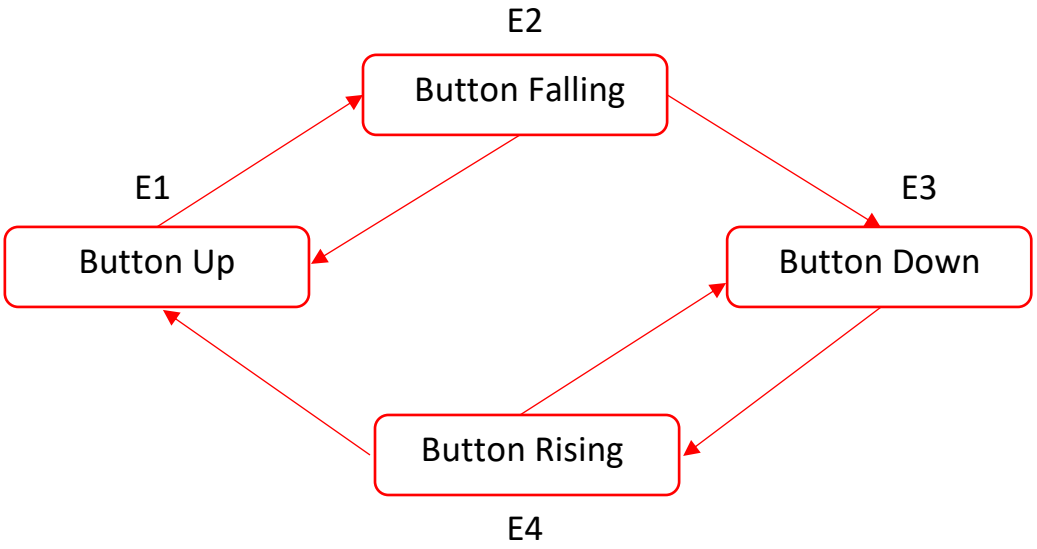
Ventajas del uso de MEF para funciones antirebote:

- **Flexibilidad:** Las MEF permiten modelar una amplia variedad de comportamientos, adaptándose a diferentes escenarios y requisitos.
- **Fiabilidad:** Al definir estados claros y transiciones precisas, se reduce la posibilidad de errores en la detección de pulsaciones.
- **Facilidad de mantenimiento:** La estructura modular de las MEF facilita la comprensión y modificación del código.
- **Determinismo:** El comportamiento de la MEF es predecible y reproducible, lo que facilita la depuración y el análisis del sistema.
- **Escalabilidad:** Las MEF pueden ser fácilmente extendidas para gestionar múltiples botones o eventos.

Funcionamiento general del algoritmo:

El algoritmo propuesto se basa en una MEF que define cuatro estados posibles para el botón:

1. **BUTTON_UP:** El botón está en reposo.
2. **BUTTON_FALLING:** Se ha detectado un flanco descendente, se inicia un temporizador para filtrar rebotes.
3. **BUTTON_DOWN:** El botón se considera presionado de forma válida.
4. **BUTTON_RISING:** Se ha detectado un flanco ascendente, se inicia un temporizador para filtrar rebotes.



API_Debounce:

Las funciones para implementar la MEF y lograr evitar los problemas debido al rebote estarán en la API_Debounce.
Dentro del archivo API_Debounce.h encontraremos las declaraciones de las funciones y las variables utilizadas, además de la estructura de enumeración para la MEF (debounceState_t). Las funciones utilizadas y que se desarrollan dentro de API_Debounce.c son:

- debounceFSM_init: Se encarga de inicializar la función con los valores predeterminados. Configura el estado inicial de la MEF como BUTTON_UP, por un lado, indicando donde comenzara la MEF en la función debounceFSM_update y que el botón es activo bajo. Inicializa un temporizador DEBOUNCE_DELAY que se utiliza para filtrar los rebotes mecánicos, tanto los producidos en los flancos descendentes como los ascendentes. Por último, apaga todos los leds de la placa de desarrollo por precaución.
- debounceFSM_update: Esta función recibe como parámetro el estado del botón que se desea utilizar y se actualiza de forma periódica dentro del programa principal. Dentro de esta función se

implementa la máquina de estado y según el estado en que se encuentre podremos utilizar una variable auxiliar, en este caso implementamos la variable booleana keyPressed, la cual indicara si se presionó o no el botón.
Se utiliza un delay de 40ms en cada estado para evitar falsas lecturas debido al rebote mecánico del botón.

- ReadKey: Función encargada de verificar si el pulsador fue presionado. Devuelve una variable booleana keyPress. Si el valor de retorno es true, se entiende que hubo un evento de pulsación válido. Por otro lado, reinicia el estado interno de la variable keyPressed, para evitar lecturas erróneas/repetitivas de la misma pulsación. Además, también se reinicia el estado de la variable que detecta un flanco descendente por la misma razón.

Aplicaciones desarrolladas:

Se implemento el driver API_debounce, el cual cumple la función de anti-rebote de un pulsador, los cambios en las aplicaciones se realizaron de modo que se inicialicen las funciones correspondientes de la API (debounceFSM_init) y con la función debounceFSM_update mediante el uso de máquina de estado se evitan las pulsaciones no deseadas. La función readKey será la utilizada para evaluar la condición de si se presionó o no el botón.

App 4.0: Como primera implementación de las funciones debounce, se utilizaron las funciones propuestas en el enunciado de la practicas, buttonPressed y buttonReleased las cuales según se detecte un flanco descendente del botón alternara el estado del LED1 (Verde) y al detectar un ascendente del botón se alterna el estado del LED3 (Rojo).
Permalink: main.c

App 4.1: En esta aplicación implementando API_debounce, se alternará entre 2 estados cuando se presione el botón integrado de la placa. Como estado inicial se tiene el encendido y apagado en secuencia de los leds cada 200ms (verde, azul y rojo) y al presionar el botón alterna al otro estado donde solo parpadea cada 200ms el led azul (LED2).
Para detectar que el botón fue presionado y ejecutar el cambio de estado o secuencia, se utilizó la función readKey como condición de un “if” para realizar el cambio de secuencia y restablecer las condiciones iniciales necesarias en cada una.
Permalink: main.c

App 4.2: En esta aplicación se debe alternar entre dos secuencias cuando se presiona el botón, por lo que se implementó de igual manera que en la aplicación 4.1 una condición if(readKey) para alternar entre las secuencias dentro de un switch.
La primera secuencia enciende y apaga los leds cada 200ms en orden (verde, azul y rojo), la segunda secuencia hará lo mismo, pero en orden inverso (rojo, azul y verde).
Permalink: main.c

App 4.3: De igual manera que en las aplicaciones anteriores se implementó una condición con la función readKey para evaluar si se presionó el botón o no y cambiar entre 4 secuencias distintas para el encendido de los leds. En esta

aplicación fue donde más se pudo notar el efecto del driver anti-rebote, ya que en versiones anteriores era común que se saltee una o más secuencias al presionar el botón ya que se detectaba como si esta se presionó muchas veces.
Permalink: main.c

App 4.4: Al igual que en la aplicación 4.3 se alterna entre cuatro secuencias donde cada una hace parpadear los leds en simultánea, pero con diferentes intervalos de tiempo. La implementación de la API_debounce se realizó de la misma forma que en las otras aplicaciones.
Permalink: main.c

En general para todas las aplicaciones no hubo grandes dificultades para realizar la implementación del driver, sino más bien en la creación de este, entendiendo como es el funcionamiento de la máquina de estado y la lógica para resolver los diferentes estados de la misma evitando los rebotes mecánicos al detectar el estado de un pulsador.

Repositorio Grupo 8: AFP_4_Grupo_8_2024