Universidad Tecnológica Nacional - Facultad Regional Tucumán

Carrera: Ingeniería Electrónica **Asignatura: Técnicas Digitales II**

Actividad de Formación Práctica Nº5 - LAB - Año 2025

Tema: Programación de microcontroladores: anti-rebote desarrollado por MEF para garantizar detección positiva de pulsadores mecánicos, aplicación en STM32CubeIDE, programación de microcontroladores.

Tipo de Actividad de Formación Práctica: Laboratorio

Unidad Temática 2.- Arquitectura de un microprocesador.
Unidad Temática 6.- Microprocesadores.

Trabajo de Laboratorio: Implementación de un driver de funciones anti-rebote (debounce) mediante MEF, aplicado en proyectos con STM32CubeIDE y placas de desarrollo STM32-NUCLEO-F4XX.

1. Alcance

El objetivo de este trabajo de laboratorio es que los estudiantes logren:

- Aplicar el concepto de Máquina de Estados Finitos (MEF) y el uso de drivers para implementar funciones anti-rebotes por software en STM32CubeIDE.
- Implementar un driver de funciones debounce y aplicarlo en proyectos previos.
- Integrar el driver en aplicaciones existentes, modificando el código para utilizar retardos no bloqueantes y funciones estructuradas.
- Presentar el trabajo en laboratorio con funcionamiento correcto y defensa técnica de lo realizado.

2. Elementos / Instrumental a Utilizar

- Placa de desarrollo STM32-NUCLEO-F4XX (modelos sugeridos: F401-RE, F412-ZG, F413-ZH, F429-ZI).
- Pulsadores mecánicos conectados a entradas digitales.
- LEDs para pruebas de encendido y apagado controlados por pulsadores.
- Computadora con software de desarrollo integrado STM32CubeIDE instalada.
- Software de Programación y *Debugging*: STM32CubeIDE para programar y depurar el código.
- Repositorio GitHub grupal para subir las aplicaciones desarrolladas.
- Documentación técnica del fabricante y material de la cátedra sobre MEF y debounce.

3. Actividades

a- Implementación del driver:

  - Partir de un proyecto base de la práctica anterior sobre programación de microcontroladores e implementar en main.c y main.h las funciones necesarias para el sistema anti-rebote con un retardo no bloqueante de 40 ms.
  - Definir el tipo de datos *debounceState_t* y las funciones **debounceFSM_init()**, **debounceFSM_update()**, **buttonPressed()** y **buttonReleased()**.
  - Configurar **buttonPressed()** para invertir el estado del LED1 y **buttonReleased()** para invertir el estado del LED3.

b- Implementación modular:

- Crear el módulo API debounce en /Drivers/API/src y /Drivers/API/inc con sus funciones públicas y privadas.
- Agregar la función *bool_t* **readKey()** que detecta flancos descendentes y reinicia el estado al ser leída.

c- Integración en aplicaciones previas:

- Modificar las Apps de la práctica anterior para usar el driver de debounce y retardos no bloqueantes del módulo API delay.

d- Pruebas y verificación:

- Compilar, depurar y probar en laboratorio cada aplicación modificada.
- Evaluar la mejora en el comportamiento de las aplicaciones con el uso del driver debounce.

e- Informe de laboratorio:

El informe debe presentar los siguientes puntos:

- Introducción al uso de debounce por MEF y ventajas en aplicaciones comerciales.
- Lista de aplicaciones modificadas, autores y observaciones.
- *Link al repositorio grupal con el código final*.

4. Duración: 5 Hs.

ESTUDIANTE: Roqué Luciano J. …… Luciano J. Roqué

ESTUDIANTE: Cruz Rodolfo Martin……

FECHA DE INICIO: __/__/____ FECHA DE PRESENTACIÓN: 17/10/25

CONFORMIDAD DEL DOCENTE:……...

Universidad Tecnológica Nacional - Facultad Regional Tucumán

Carrera: Ingeniería Electrónica Asignatura: Técnicas Digitales II

Actividad de Formación Práctica Nº5 - LAB - Año 2025

Informe de Máquinas de Estado Finito (MEF)

En esta práctica se trabajó con la implementación de un sistema antirrebote por software utilizando Máquinas de Estado Finitas (MEF). Un pulsador mecánico no cambia su estado de manera limpia cuando se presiona o libera, sino que presenta pequeños rebotes eléctricos que pueden generar múltiples detecciones falsas si no se los filtra. Para resolver esto se implementó una MEF que detecta transiciones válidas entre los estados BUTTON_UP, BUTTON_FALLING, BUTTON_DOWN y BUTTON_RISING, usando un retardo no bloqueante de 40 ms.
La MEF permite controlar el flujo de la lógica de forma ordenada y predecible, evitando lecturas erróneas del pulsador. Entre las ventajas en aplicaciones comerciales, se destaca:

- Mayor fiabilidad en la detección de eventos.
- Evita errores por rebotes que podrían afectar procesos críticos.
- Permite modularidad: el mismo driver puede reutilizarse en distintas aplicaciones.
- Mejora la escalabilidad y mantenibilidad del código.

Aplicaciones desarrolladas:

Nombre: App_1_1_Grupo_4_2025_Driver_Antirrebote

Autor de la modificación: Cruz Rodolfo Martín

Observaciones:

Este trabajo se basó en la implementación de una Máquina de Estados Finitos (MEF) para gestionar el comportamiento del botón de usuario y el encendido de los LEDs.
La MEF permite visualizar los cambios de estado a través de los LEDs verde (LED1) y rojo (LED3). La implementación presentó cierto desafío, ya que fue necesario modificar el archivo main.c para integrar los drivers del módulo API_debounce, donde se encuentra la lógica principal de la MEF.
El main.c debía llamar a las funciones del driver sin alterar las funcionalidades previamente desarrolladas en los ejercicios anteriores, como la secuencia de encendido de LEDs y el manejo del retardo no bloqueante.
Este proceso requirió tiempo adicional, especialmente al inicio, ya que los primeros intentos modificaban accidentalmente la secuencia original de LEDs, generando un comportamiento distinto al esperado.

Recomendación:

- Es importante documentar los cambios realizados y comprobar la correcta transición de estados mediante pruebas prácticas, lo que facilita la detección de errores y asegura un comportamiento estable del Sistema.

Nombre: App_1_2_Grupo_4_2025_Driver_Antirrebote

Autor de la modificación: Roqué Luciano

Observaciones:

La aplicación original consistía en recorrer un vector de tres LEDs en un único sentido, encendiéndolos secuencialmente. La modificación introducida consistió en utilizar el nuevo driver antirrebote con MEF para

invertir el sentido de recorrido cada vez que se presiona el pulsador. Además, se mantuvo la indicación visual de cambio de estado a través de la inversión del LED1 al detectar un flanco descendente.
Durante el desarrollo surgieron algunos problemas relacionados con el comportamiento inestable del sistema al detectar el cambio de estado. Esto se solucionó al eliminar la lógica de detección directa en el while y trasladarla a la función buttonPressed(), evitando así lecturas falsas y simplificando el flujo del programa. También se mejoró la organización del código al usar etiquetas para los LEDs y tiempos, lo que facilita la modificación y el mantenimiento futuro.

Recomendaciones:

- Mantener la lógica de eventos separada de la lógica de ejecución principal mejora la claridad y permite reutilizar el driver en otras aplicaciones.
- Al trabajar con secuencias de LEDs, es útil reiniciar los estados y delays al detectar cambios para evitar comportamientos inesperados.

Aplicaciones desarrolladas:

Nombre: App_1_3_Grupo_4_2025_Driver_Antirrebote

Autor de la modificación: Alan Cancino

Observaciones:
Siguiendo con la implementación de drivers en la app 3, agregamos el control de estados al pulsador, lo que nos permite identificar 4 diferentes estados. Se continua con el camino marcado por la catedra de código modular y portable.

Se siguio la guía de desarrollo de la catedra por la que no hubo mayores contratiempos, solamente se debió de implementar reinicios de estado para que no surgieran parpadeos erráticos en los LEDs

Recomendaciones:

- Utilizacion de etiquetas claras y descripción breve de funciones o líneas clave

Aplicaciones desarrolladas:

Nombre: App_1_4_Grupo_4_2025_Driver_Antirrebote

Autor de la modificación: Cusi Alejandro

Observaciones:
Continuando con las modificaciones de la app 4 original, esta vez agregándole drivers de MEF para la lectura de estados de pulsadores. Se intento seguir con las consignas de un código limpio y entendible a simple vista.

Al principio se realizo un Driver aceptable desde su funcionamiento, pero se cometio el error de utilizar drivers anteriores para simplificar este último, lo que va en contra de la portabilidad del código y de los drivers desarrollados. Luego de caer en cuenta de dicho error se procedio a volver a la metodología correcta.

Universidad Tecnológica Nacional - Facultad Regional Tucumán

Carrera: Ingeniería Electrónica **Asignatura: Técnicas Digitales II**

Actividad de Formación Práctica Nº5 - LAB - Año 2025

Recomendaciones:

- Probar el correcto funcionamiento del driver desarrollado en un proyecto de prueba, es decir, que nos permita testear cada función en nuestro código de manera clara y concisa.

Grupo 4 : Cancino Alan Maximiliano
Cruz Martín Rodolfo
Cusi Alejandro Daniel
Roqué Luciano Javier

Actividad de Formación Practica N°5

Repositorio Grupal: Grupo-4-TDII-2025