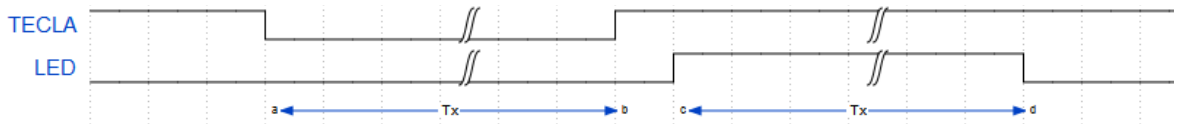
A. Baremetal cooperativo con Pont.

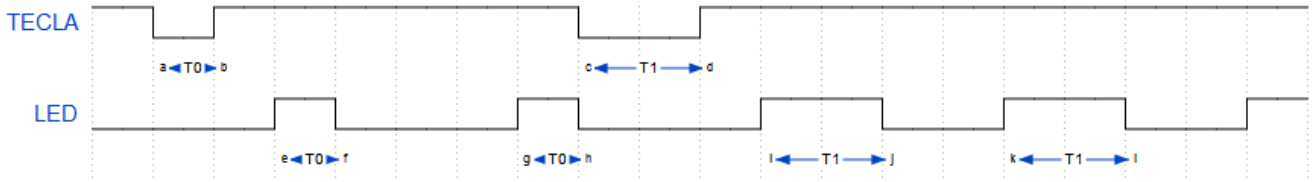
A.1 Empleando dos tareas, medir el tiempo de pulsación de un botón utilizando un algoritmos anti-rebote. Luego destellar un led durante el tiempo medido. Ayuda: Se puede consultar el contador de ticks para obtener el tiempo del sistema (en ticks) al inicio y al fin del mismo. En este caso hay que prever que esta variable puede desbordar.



A.2 Escribir un programa con dos tareas:

- Tarea 1: Medirá el tiempo de pulsación de un botón, aplicando anti-rebote.
- Tarea 2: Destellará un led con un período fijo de 1 seg, y tomando como tiempo de activación el último tiempo medido.

El tiempo medido se puede comunicar entre tareas a través de una variable global.



A.3 Extender A.1 para que se usen 4 teclas contra 4 LEDS

A.4 UART RX
Utilizando el ejemplo *firmware_v3\examples\c\sapi\uart\rx_interrupt* implemente un programa con una tarea:

- Configura el driver de sapi para que puedan utilizarse interrupciones para recibir datos via UART.
- Esperara un paquete que comience con '>' y finalice con '<'. Al recibirlo:
    - Una tarea deberá encender un led testigo.
    - Pasados 1 segundo de la recepción, deberá enviarse (usando printf) el texto "Recibido".

B. Gestión de tareas cooperativas en FreeRTOS

NOTA: Para implementar los delays empleados para periodizar a los LEDs **NO** utilizar vTaskDelay. Para ello, utilizar una función "delay" empleando ciclos for. La intención de esta sección de la guía es que las tareas **NUNCA** transiten por el estado Blocked.

B.1- Implementar tres tareas en FreeRTOS:

Tarea A: Encienda periódicamente el LED rojo.
Tarea B: Encienda periódicamente el LED azul.
Tarea C: Monitoree el valor mínimo utilizado del stack de cada tarea.

B.2- Para el caso de B.1, utilice el IDLE hook para implementar el monitoreo. ¿ Qué ocurre ?

B.3- Implemente un sistema de 4 tareas:

Tarea A: Prioridad IDLE + 4 - LED asociado LEDB
Tarea B: Prioridad IDLE + 3 - LED asociado LED1
Tarea C: Prioridad IDLE + 2 - LED asociado LED2
Tarea D: Prioridad IDLE + 1 - LED asociado LED3

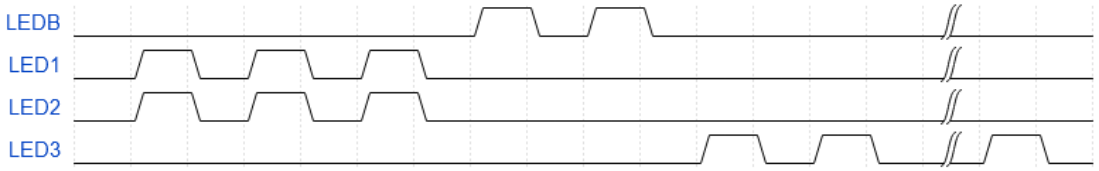
Arrancando solamente la tarea A antes de comenzar el scheduler, genere la siguiente secuencia de encendido y apagado (500ms/500ms):

Solo la tarea D podrá destruir las otras, cuando comience a operar, dejando titilando a LED3

B.4- Partiendo del ejercicio B.3 implemente un sistema de 4 tareas:

Tarea A - Prioridad IDLE + 4 - LED asociado LEDB
Tarea B - Prioridad IDLE + 2 - LED asociado LED1
Tarea C - Prioridad IDLE + 2 - LED asociado LED2
Tarea D - Prioridad IDLE + 1 - LED asociado LED3

Valide que la secuencia es la siguiente.



Las tareas B y C DEBEN tener un código fuente equivalente (salvando la parte en donde se accede al LED).

Ahora, configure en *freertosconfig.h*: `#define configUSE_TIME_SLICING 0`

¿Qué sucedió?

Proponga una manera de contrarrestar el efecto sin tocar la configuración mencionada (no utilice la Suspend/Resume para solucionarlo)