A. Gestión de tareas

A.1- Implementar tres tareas que no posean llamados a funciones de la API de FreeRTOS con puntos de re-planificación con la misma prioridad. Una de las tareas deberá monitorear el valor mínimo utilizado del stack de cada tarea.

Valide el funcionamiento del mecanismo de detección.

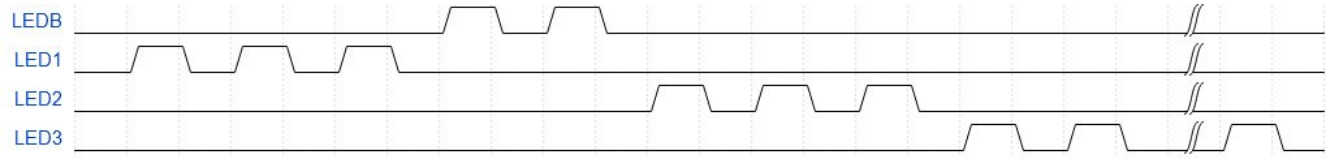
A.2- Para el caso de A.1, utilice el IDLE hook para la validación. ¿ Qué ocurre ?

A.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):

LED1 LED1 LED1 LEDB LEDB LED2 LED2 LED2 y posterior a esta secuencia, permaneces titilando a LED3

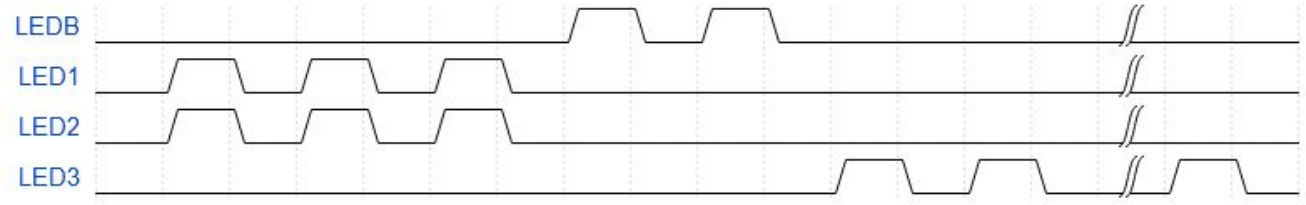


Solo la tarea D podrá destruir las otras, cuando comience a operar.

A.4- Partiendo del ejercicio A.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.



Analice el funcionamiento.

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

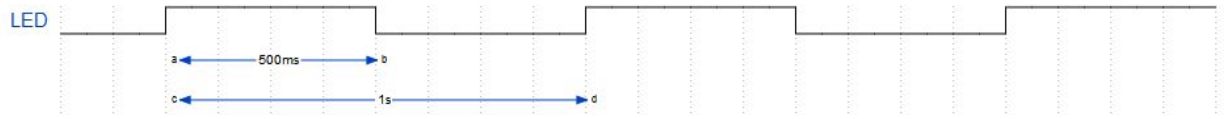
¿Qué sucedió?

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

B. Temporización

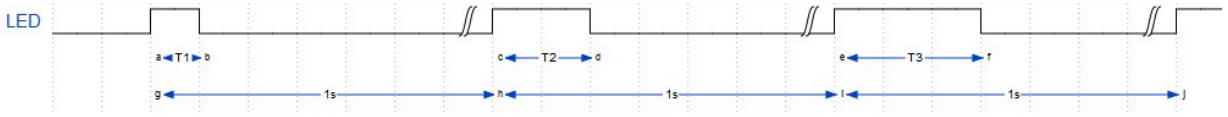
B.1- Demoras fijas

Implementar una tarea que encienda un LED durante 500 ms cada T = 1 seg.



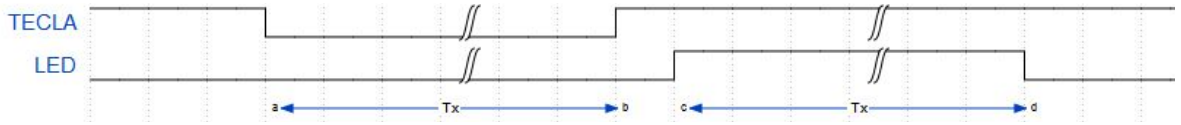
B.2- Periodos fijos

Implementar una tarea que genere una onda cuadrada (y que encienda un LED) con periodo de 1 seg y ciclos de actividad incrementándose T1 = 100 ms, T2 = 200 ms, T3 = 300 ms, .. T9 = 900ms



B.3- Medir tiempo transcurrido

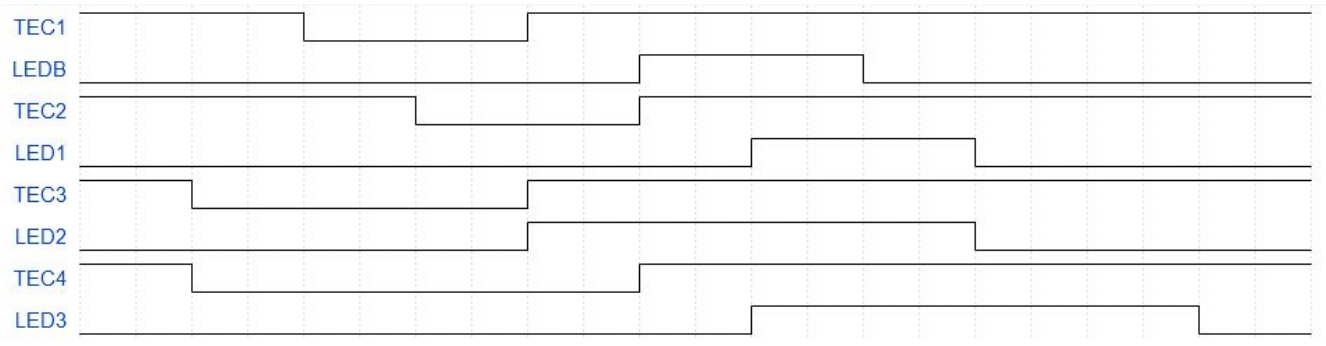
Medir el tiempo de pulsación de un botón utilizando un algoritmos anti-rebote. Luego destellar un led durante el tiempo medido. Ayuda: Se puede consultar el contador de ticks del RTOS para obtener el tiempo del sistema (en ticks) al inicio y al fin del mismo. En este caso hay que prever que esta variable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



B.4- Medir tiempo transcurrido en múltiples teclas

Rehacer el ejercicio B.3 para múltiples teclas independientes, asociadas cada una a un led distinto. Por ejemplo:

- TEC1 -> LEDB
- TEC2 -> LED1
- TEC3 -> LED2
- TEC4 -> LED3



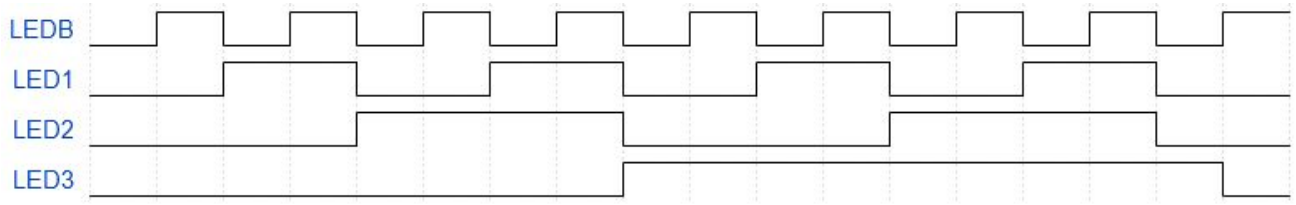
B.5- Medir tiempo transcurrido (utilizando tarea one-shot)

Rehacer el ejercicio B.3 pero la tarea asociada al led debe ser one-shot. Es decir, al presionar la tecla se deberá crear una tarea_led que encienda el led correspondiente, luego se apague y la tarea se autodestruya, liberando la memoria asociada a la misma.

B.6- Demoras fijas (múltiples leds y tick rate modificado)

Rehacer el ejercicio B.1 para múltiples leds, donde cada led deberá tener el doble de tiempo encendido que el anterior. Es decir:

- LEDB -> 500ms con T = 1s
- LED1 -> 1s con T = 2s
- LED2 -> 2s con T = 4s
- LED3 -> 4s con T = 8s



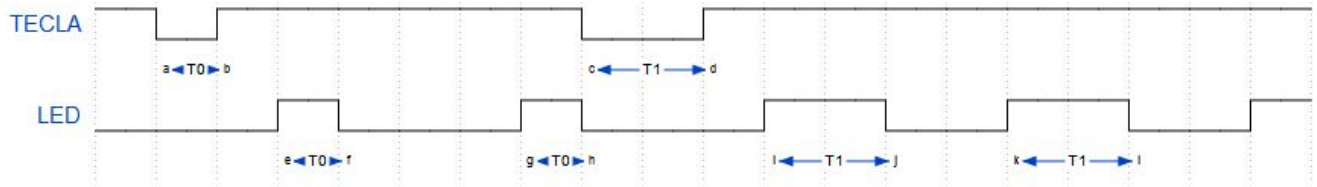
Modificar el tickrate del archivo de configuración, aumentando y disminuyendo su valor ¿Qué cambios notan?

B.7- Ejercicio integrador (ENTREGA OBLIGATORIA)

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, protegiendo sus operaciones dentro de una sección crítica.



B.8- Ejercicio integrador (OPCIONAL)

Incluir en el ejercicio B.7 la posibilidad de utilizar todas las teclas y leds.

B.9. Transmisión por UART

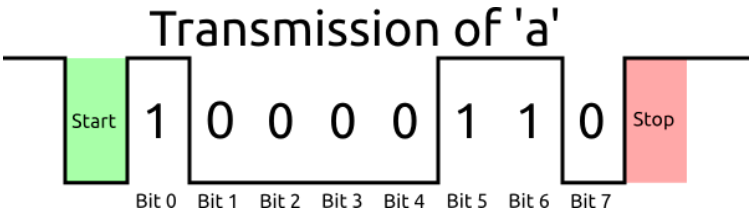
En una cierta aplicación todas las UARTs del microcontrolador están ocupadas. Se desea comunicar a través de una nueva comunicación asincrónica a muy baja tasa de transmisión (< 500 bps) con otro periférico.

Implementar una tarea que transmita bytes a cierta tasa de transmisión a través de un GPIO, con la configuración 8 bits de datos, sin paridad y 1 bit de stop.

Se recomienda realizar una librería (cuya API deba llamarse desde una tarea del Sistema operativo) con los métodos:

- **void sw_uart_sent(uint8_t byte_a_transmitir)**
- **void sw_uart_config(uint16_t baurdate)**

El método sw_uart_sent no debe ser bloqueante para el resto de las tareas.



B.10. Recepción por UART

Para la API escrita en el ejercicio anterior, implemente el método:

uint8_t sw_uart_receive(uint32_t timeout)

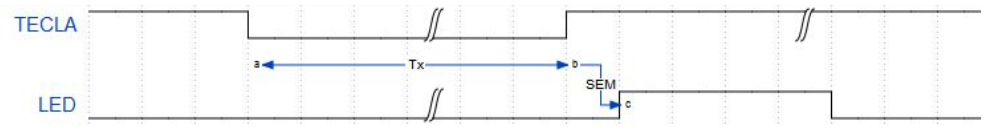
Deberá permitir recibir un byte en el formato que haya sido configurado. El parámetro timeout, deberá ser un valor en ticks en el cual la tarea que llame a este método, deje de esperar el bit de start.

C Sincronización de tareas mediante semáforos

C.1. Sincronizar dos tareas mediante un semáforo binario

Implementar dos tareas.

- Tarea 1: Medirá el tiempo de pulsación de un botón, aplicando anti-rebote. Liberará un semáforo al obtener la medición.
- Tarea 2: Esperará por el semáforo y destellará un LED al recibirlo.



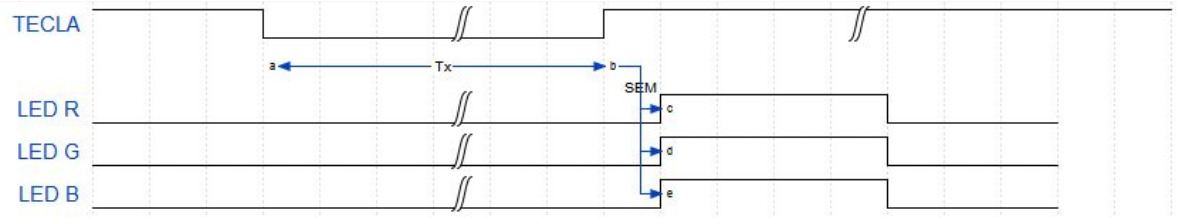
C.2. Sincronizar dos tareas mediante un semáforo binario

Repetir el ejercicio C.1 pero con múltiples teclas.

C.3. Sincronizar varias tareas

Implementar tres tareas:

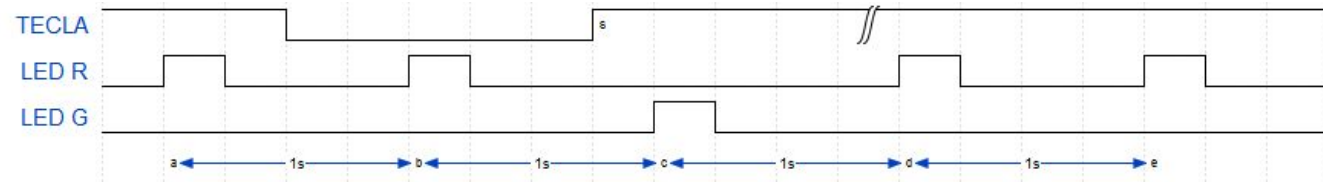
- Tarea 1: Medirá el tiempo de pulsación de un botón, aplicando anti-rebote. Liberará un semáforo al obtener la medición.
- Tarea 2,3,4: Tareas idénticas que destellarán un led (diferente) al recibir el semáforo.



C.4. Tiempo de bloqueo (ENTREGA OBLIGATORIA)

Implementar dos tareas:

- Tarea 1: Medirá el tiempo de pulsación de un botón, aplicando anti-rebote. Liberará un semáforo al obtener la medición.
- Tarea 2: Esperará el semáforo cada un segundo. Si recibe el semáforo se destellará el LED verde y si no recibe el semáforo destellará el LED rojo.



C.5. Cuenta de eventos con sincronización

Implementar dos tareas.

- Tarea 1: Medirá la pulsación de un botón, aplicando anti-rebote. Liberará un semáforo al liberar cada botón.
- Tarea 2: Destellará un LED 0,5 seg y lo mantendrá apagado 0,5 seg, cada vez que se pulse la tecla. Si durante el periodo se pulsan teclas, no se deberán perder esos eventos (con un límite de 3)