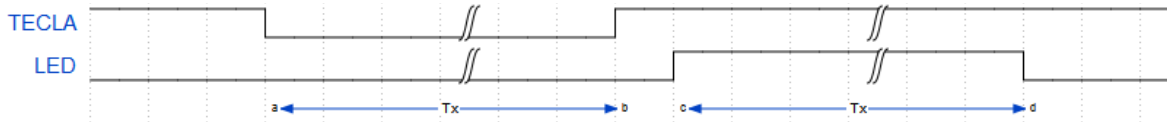


A. Baremetal cooperativo con Pont.

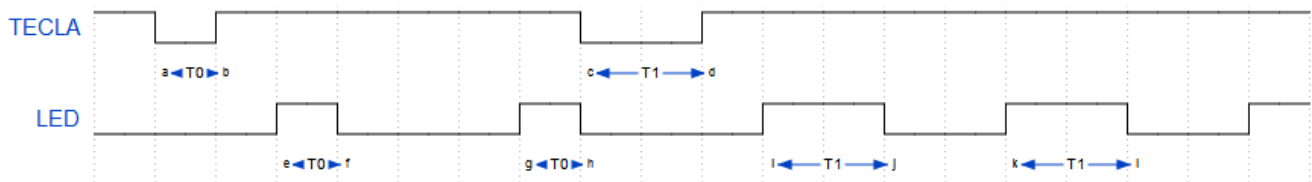
A.1 Empleando dos tareas, medir el tiempo de pulsación de un botón utilizando un algoritmos anti-rebote. Luego destellar un led durante el tiempo medido. Ayuda: Se puede consultar el contador de ticks para obtener el tiempo del sistema (en ticks) al inicio y al fin del mismo. En este caso hay que prever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

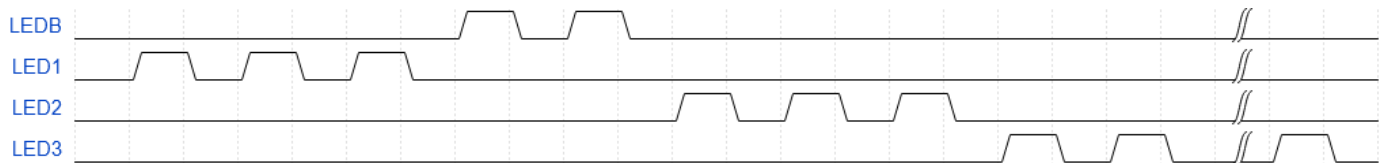
Tarea A: Prioridad IDLE + 4 - LED asociado LEDB

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:

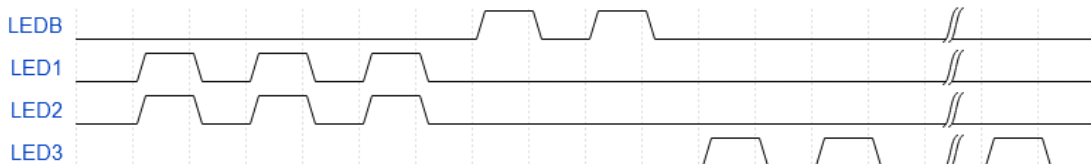
Tarea A - Prioridad IDLE + 4 - LED asociado LEDB

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`

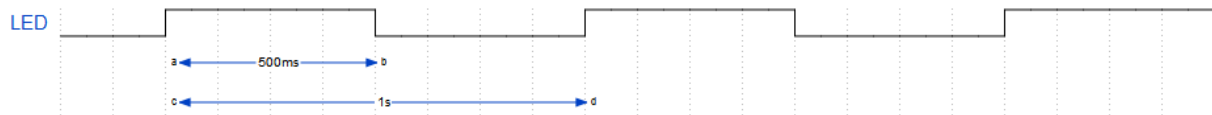
¿Qué sucedió?

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

C. Temporización

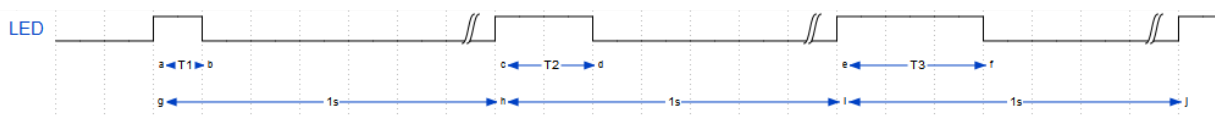
C.1- Demoras fijas

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



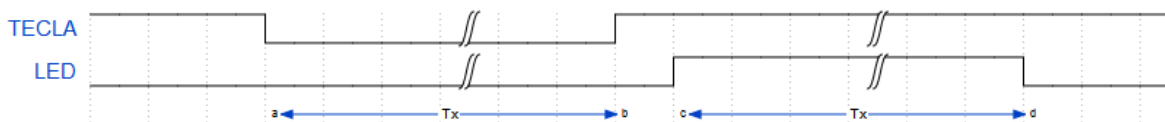
C.2- Periodos fijos

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, .. $T9 = 900$ ms



C.3- Medir tiempo transcurrido

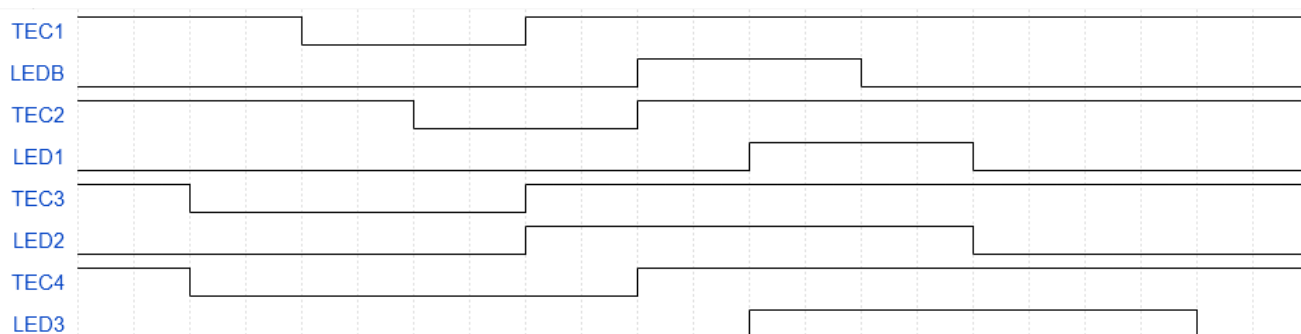
Medir el tiempo de pulsación de un botón utilizando un algoritmos anti-rebote. Luego destellar un led durante el tiempo medido. Ayuda: Se puede consultar el contador de ticks del RTOS para obtener el tiempo del sistema (en ticks) al inicio y al fin del mismo. En este caso hay que prever que esta variable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



C.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



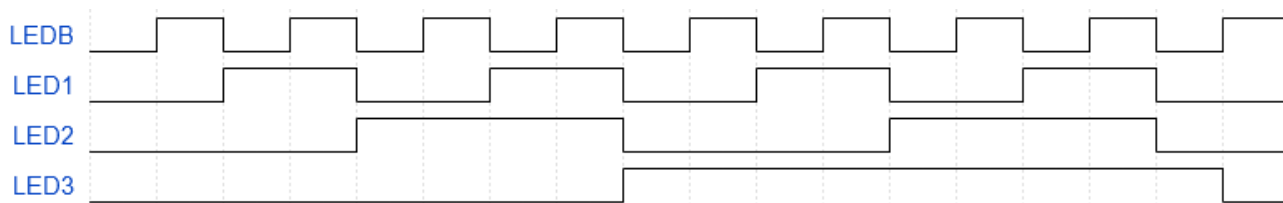
C.5- Medir tiempo transcurrido (utilizando tarea one-shot)

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.

C.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 $T = 8s$



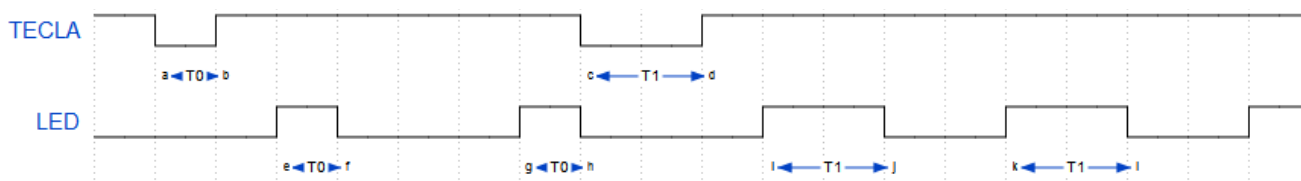
Modificar el tickrate del archivo de configuración, aumentando y disminuyendo su valor ¿Qué cambios notan?

C.7- Ejercicio integrador

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.



Ayuda: Usar el C.3 como base

C.8- Ejercicio integrador

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

Ayuda: Usar el C.4 como base

C.9. Transmisión por UART

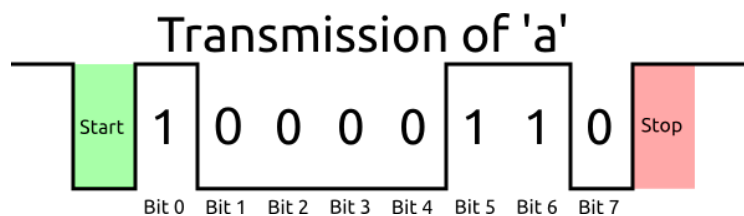
En una cierta aplicación todas las UARTs del microcontrolador están ocupadas. Se desea comunicar a través de una nueva comunicación asincrónica a muy baja tasa de transmisión (< 500 bps) con otro periférico.

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 baudrate)`

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



C.10. Recepción por UART

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

`uint8_t sw_uart_receive(uint32_t timeout)`

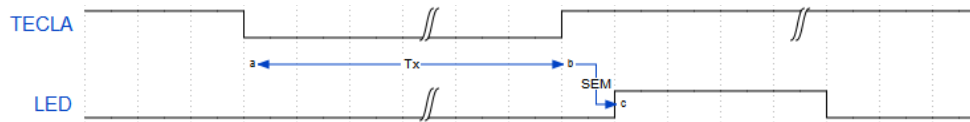
Deberá permitir recibir un byte en el formato que haya sido configurado. El parámetro `timeout`, deberá ser un valor en ticks en el cual la tarea que llame a este método, deje de esperar el bit de start.

D- Sincronización de tareas mediante semáforos

D.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.



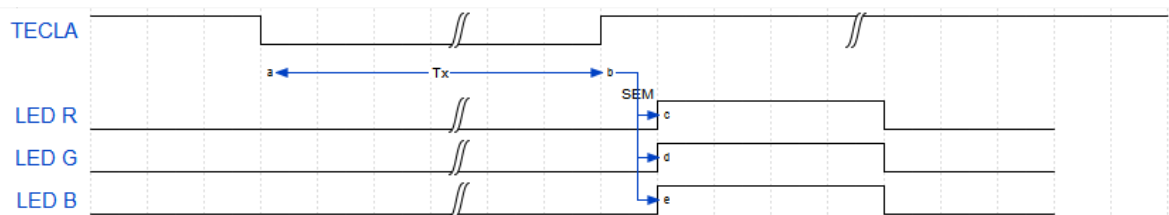
D.2. Sincronizar dos tareas mediante un semáforo binario

Repetir el ejercicio D.1 pero con múltiples teclas.

D.3. Sincronizar varias tareas

Implementar tres tareas:

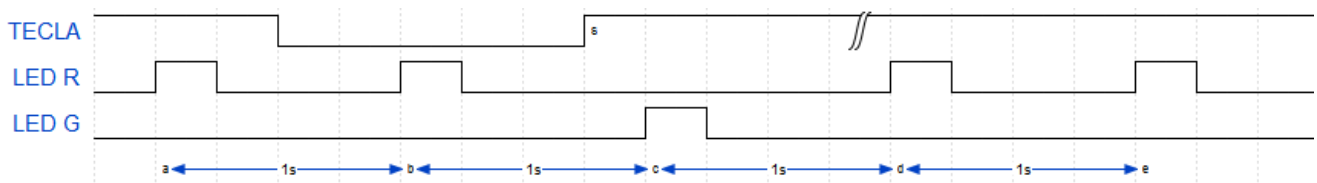
- Tarea 1: Medirá el tiempo de pulsación de un botón, aplicando anti-rebote. Liberará un semáforo al obtener la medición.
- Tarea 2,3,4: Tareas idénticas que destellarán un led (diferente) al recibir el semáforo.



D.4. Tiempo de bloqueo

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.



D.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)

D.6. Cuenta de eventos con sincronización

Implementar el ejercicio D.5 de forma modular.

E- Gestión de recursos compartidos

E.1 printf conflictivo

Escriba un programa que tenga configurada la UART USB en 9600 bps.

Agregue 4 tareas con la misma prioridad que impriman por la UART "Hola soy la Tarea X" de manera periódica, con 33, 55, 77, 200 ms de periodicidad, respectivamente.

Evalúe el resultado.

¿Cómo corrige el comportamiento?

E.2 El ejercicio C7 tiene errores !

El problema C7 ¿Está bien implementado?

Corregirlo para acceder correctamente a los recursos globales.

E.3 Más acceso concurrente

Agregue el funcionamiento de E.2, otra tecla funcional (TEC2) cuya acción al pulsarla borre la diferencia de tiempo de la TEC1 (que en E.2 definía el tiempo de encendido del LED).

E.4 Acceso a un módulo desde varias tareas

Escribir un programa tres tareas:

- Tarea 1: Medirá la pulsación de 2 teclas, aplicando anti-rebote. La tecla 1, incrementará un contador (C1) y la tecla 2 lo decrementará (como si fuera un control de volumen). Valor mínimo de C1 = 100. Valor máximo de C1 = 900. Valor inicial C1 = 500. Incrementos/Decrementos de 100 ms.
- Tarea 2: Destellará el LED0 con un período de C1 ms (duty cycle 50%).
- Tarea 3: Destellará el LED1 con un periodo fijo de 2 s y un tiempo de encendido de $2 \times C1$ ms. En cada ciclo, deberá decrementar C1 en 100.

El tiempo medido se puede comunicar entre tareas a través de una variable global, protegiendo sus operaciones dentro de una sección crítica.

E.5 Inversión de prioridades

Escribir un programa que explicita la problemática de la inversión de prioridades, según la filmilla de la presentación teórica titulada "¿Cómo resolver (MAL) el problema?"

E.6 Inversión de prioridades + UART

Basándose en E.5 y E1, desarrolle un sistema en donde se evidencie (y posteriormente se corrija) el problema de inversión de prioridades, cuando el recurso es la UART.