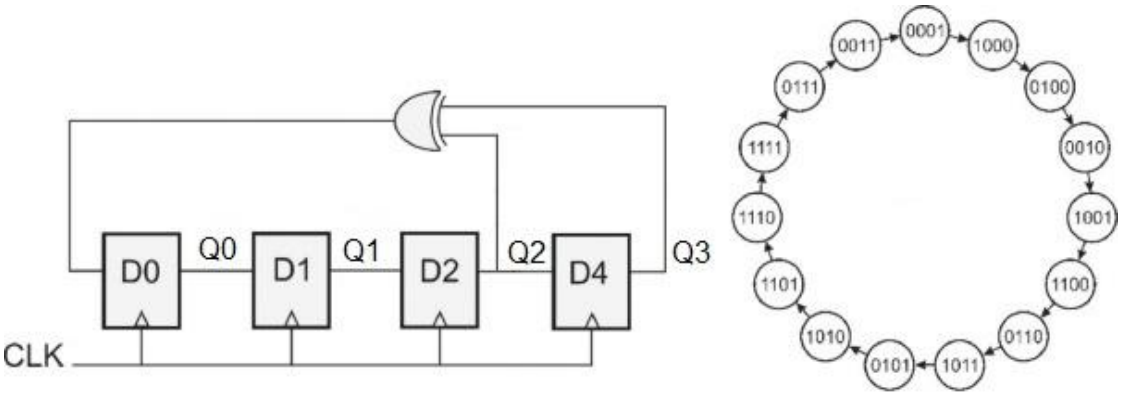
# *Ejercicios FreeRTOS*

## *1. Números pseudoaleatorios (PRNG)*

Los generadores de números pseudoaleatorios son algoritmos que dan una muy buena aproximación a un conjunto aleatorio de números. La sucesión no es exactamente aleatoria en el sentido de que queda completamente determinada por un conjunto relativamente pequeño de valores iniciales, llamados el estado o semilla del PRNG.
La mayoría de los generadores de números pseudoaleatorios producen números con distribución uniforme. Entre las propiedades deseables de un PRNG están la de tener un período largo o lo más largo posible y una buena distribución (es decir lo más uniforme posible).
Hay muchos algoritmos para llevar adelante esta tarea: Blum Blum Shub, Mersenne Twister o WELL entre otros con diferentes características.
Conceptualmente se puede llegar a un PRNG partiendo de un contador LFSR de muchos bits (para mantener un período largo).




***LFSR de 4 bits***

Para esta práctica vamos a utilizar un subtipo de los LFSR denominados ***xorshift*** publicados por George Marsaglia en 2003. Se caracterizan por tener buenas cualidades de aleatoriedad con un código rápido y sencillo. El PRNG a implementar es el xorshift de período $2^{32}$-1 que se encuentra en el comienzo de la página 4 (función xor) del documento https://www.jstatsoft.org/article/view/v008i14.

```
uint32_t xor32(void) {
    static uint32_t y = 2463534242; /*semilla*/
    y^= y<<13;
    y^= y>>17;
    y^= y<<5;
    return y;
}
```

***PRNG a utilizar en el TP***

Puede verificar el funcionamiento del generador en https://onlinegdb.com/Gj-5ElWyK.

## 2. Generar tareas.

Dado el programa que hace parpadear dos leds controlado por dos pulsadores https://drive.google.com/file/d/1a4wqPmjjCFmxpqyyVPMh3D80B5nJ1mqT/view?usp=sharing Se pide:

- Haga una copia de este programa en un workspace vacío.
- Implemente la misma funcionalidad del programa pero con FreeRTOS. Es decir, el led de la placa y los dos leds adicionales (B8 y B9) que parpadeén con un período de 0.25s
- Utilizar los pulsadores en B0 y B1 para controlar los leds de B8 y B9. Sí los pulsadores están abiertos los leds parpadean, si están presionados, no.
- Se sugiere utilizar tres tareas (una para el led de la placa que parpadea libremente y dos más para los leds de B8 y B9).

## 3. Generar tareas (2).

Implementar el programa del punto anterior, pero utilizando código único para las tareas de los leds B8 y B9.

## 4. Prioridades.

Implementar una aplicación que genere tres tareas de FreeRTOS. ***tarea_pulsadores***, ***tarea_led1***, ***tarea_led2***. Que se comporten de la siguiente manera:

- Cuando el sistema inicia, ***tarea_led1*** y ***tarea_led2*** tienen prioridad 1 ambas y ***tarea_pulsadores*** tiene prioridad 3.
- ***tareas pulsadores*** funciona de la siguiente manera. Cuando se presiona el pulsador en B0 pone a ***tarea_led1*** con prioridad 2 y deja a ***tarea_led2*** con prioridad 1, si se vuelve a presionar B0 ***tarea_led1*** pasa a tener prioridad 1 y ***tarea_led2*** pasa a tener prioridad 2, si se presiona otra vez, vuelve a la primera condición y así indefinidamente. Cuando se presiona B1 se genera un número con la función xor() y limitarlo entre los valores 0 y 2. Sí es cero, ***tarea_led1*** y ***tarea_led2*** pasan a tener

prioridad 1. Sí es uno ***tarea_led2*** pasa a tener prioridad 2 dejando a ***tarea_led1*** con prioridad 1. Si es dos **tarea_led1** pasa a tener prioridad sobre **tarea_led2.**

- ***tarea_led1*** debe encender el led en el puerto B8 y apagar el led del puerto B9 sin bloquearse nunca.
- ***tarea_led2*** debe encender el led en el puerto B9 y apagar el led del puerto B8 sin bloquearse nunca.
- Implemente la solución.
- Justifique los patrones de leds según presiona B0.
- ¿Puede determinar qué tarea ganó el sorteo de B1? ¿Cómo?

## 5. Semáforos (mutua exclusión).

Implementar una aplicación que genere dos tareas de FreeRTOS. ***tarea_rojo***, ***tarea_verde***, Que se comporten de la siguiente manera:

- Cuando el sistema inicia, ***tarea_rojo*** y ***tarea_verde*** tienen prioridad 1.
- ***tarea_rojo***, utiliza el led de B8 y el pulsador de B0, mientras que ***tarea_verde*** el led de B9 y el pulsador de B1.
- Ambas tareas deben comportarse de la misma manera. Las tareas intentan ganar el semáforo de mutua exclusión ***sem_mutex*** y la tarea que lo gana enciende el led y lo mantiene encendido hasta que se presiona el pulsador asociado a la misma. Luego apaga el led, duerme 1ms y vuelve a intentar ganar el semáforo.
- Implemente la solución.
- Describa el funcionamiento del programa, ¿Es esperable?¿Cuántos leds pueden estar encendidos a la vez?

## 6. Semáforos (binario).

Implementar una aplicación que genere dos tareas de FreeRTOS. ***tarea_pulsador*** y ***tarea_led***, Que se comporten de la siguiente manera:

- Cuando el sistema inicia, ***tarea_pulsador*** y ***tarea_led*** tienen la misma prioridad. También crea el semáforo binario ***sem_led*** y lo bloquea antes de iniciar el sistema.
- ***tarea_led***, utiliza el led de B8 y espera que por ***sem_led*** infinitamente. Cuando lo consigue invierte el led de B8. ***No libera el semáforo sem_led.***
- ***tarea_pulsador***. Se encarga de hacer el antirrebote del pulsador B0, y mide el tiempo que se encuentra presionado (en ms), luego de que se libera el pulsador, se encarga de liberar periódicamente el semáforo ***sem_led*** el tiempo que estuvo presionado el pulsador. Si fue menor a 50ms lo hace cada 50ms y si es mayor a 1s lo hace cada 1s, en los tiempos intermedios los hace el tiempo que estuvo presionado el pulsador. Es recomendable para implementar esta tarea que la tarea se “duerma” siempre 1ms y por medio de una máquina de estados implementar el

comportamiento pedido. Puede ser útil la función xTaskGetTickCount() para resolver esta tarea https://www.freertos.org/a00021.html#xTaskGetTickCount.

## *7. Colas de mensajes.*

Implementar la aplicación del punto anterior con colas de mensajes.