ELECTRÓNICA DIGITAL 3

TALLER PRÁCTICO

Ejercicios propuestos

Autor: David Trujillo

Módulo 1 – GPIO y PINSEL

1. Identificación de Registros PINSEL y PINMODE

El objetivo de este ejercicio es deducir la regla general para la identificación y manipulación de los registros de selección de función de los pines de la LPC1769.

Consigna:

- Deduzca una fórmula para calcular el número de registro (0, 1, 2, etc.) para un pin **Px.y** (donde x es el puerto e y es el número de pin).
- Una vez identificado el registro, deduzca una segunda fórmula para calcular la posición de los dos bits que controlan ese pin.
- Utilice sus fórmulas para determinar qué registro y qué par de bits se deben configurar para el pin **P1.8**.

2. Utilización del LED RGB Integrado

Escriba un programa que haga parpadear un color en el LED RGB integrado en la placa.

- **Pines:** Rojo (P0.22), Verde (P3.25), Azul (P3.26). Los LEDs son activos por bajo (se encienden con un 0 lógico).
- **Condiciones:**
  - Modularice el código utilizando funciones como confiGPIO() y delay().
  - Implemente un retardo por software (con un bucle for).
  - Defina el tiempo del retardo con una macro (#define).

3. Secuencias de Colores con el LED RGB

Escriba un programa que alterne entre dos o más secuencias de colores en el LED RGB. El tiempo de duración de cada secuencia debe ser considerablemente mayor que el retardo entre los colores dentro de la misma secuencia.

- **Ejemplo:**
  - secuencia_A: rojo -> verde -> amarillo (con un retardo corto entre colores).
  - secuencia_B: azul -> magenta -> blanco (con un retardo corto entre colores).
  - La transición de secuencia_A a secuencia_B debe ocurrir después de un tiempo largo (ej. varios segundos).

4. Contador Hexadecimal Automático de 1 Dígito

Utilice 7 pines GPIO para controlar un display de 7 segmentos. El programa debe mostrar de manera cíclica y automática los 16 dígitos hexadecimales (0-F) de forma ascendente.

5. Identificación de Nivel (Control de LED con Pulsador)

Escriba un programa que identifique la presión de un pulsador conectado a un pin GPIO configurado como entrada. Utiliza otro pin de salida para encender un LED solo mientras el botón está presionado.

6. Contador de Bits de un Puerto

Escriba un programa que lea el estado de todos los pines disponibles del Puerto 0 y cuente cuántos de ellos están en un nivel alto (1 lógico). El resultado debe mostrarse en binario utilizando 5 LEDs conectados a los pines menos significativos del Puerto 2.

7. Contador Hexadecimal con Botón

Similar al ejercicio 4, pero el contador solo debe avanzar un dígito cuando se presiona un botón.

- **Consideración:** Implemente un método de *debouncing* (antirrebote) por software para evitar múltiples conteos por una sola pulsación.

8. Decodificador binario a 7 Segmentos

Escriba un programa que lea un valor de 4 bits en formato binario (0-F) de 4 pines de entrada. Este valor debe ser decodificado y mostrado en un display de 7 segmentos.

9. ALU de 4 Bits

Utilice dos grupos de 4 pines de entrada (A_in, B_in) y un interruptor para seleccionar una operación.

- El interruptor (conectado a un pin) debe decidir si los valores de A_in y B_in se **suman** o se **restan**.
- El valor absoluto del resultado debe mostrarse en 4 LEDs de salida.
- Utilice un LED adicional para mostrar si se genera un overflow en la suma o si el resultado de la resta es negativo

10. Promedio Móvil

Escriba un programa que realice un promedio móvil con los últimos 8 datos leídos del puerto 0.

- **Entrada:** Los 8 bits menos significativos del Puerto 0, leídos cada cierto tiempo (usando un *delay*).
- **Salida:** El resultado del promedio, también de 8 bits, debe mostrarse en los 8 pines más bajos del Puerto 2.
- **Consideración:** Un promedio móvil se calcula sumando los últimos n valores y dividiendo por n. En este caso, n=8.

Módulo 2 – Interrupciones

1. Toggling de un LED con Interrupción

Escriba un programa muy sencillo que utilice un pin del Puerto 2 como interrupción externa (EINT0). Cada vez que se active la interrupción (por flanco de bajada), un LED conectado a otro pin debe cambiar de estado (encenderse si estaba apagado, y apagarse si estaba encendido).

- **Pines:** Utiliza **P2.10** para el botón (EINT0) y **P0.22** para el LED.

2. Contador Hexadecimal con Botón

Realice el ejercicio del contador hexadecimal (1.7), pero en esta ocasión, el incremento de la cuenta debe ser manejado por una interrupción por flanco de subida en un pin GPIO.

- **Pines:** Utilice un pin del Puerto 0 como entrada para el pulsador y 7 pines del Puerto 2 para el display de 7 segmentos.
- **Condiciones:**
    - Configure el pin de entrada para que genere una interrupción solo cuando se detecte un flanco de subida (paso de 0 a 1).
    - La lógica de incremento del contador debe residir en el **Handler de la Interrupción (ISR)**.

3. Contador Reversible con Interrupción Externa

Realice un contador hexadecimal automático (1.4), pero esta vez utilice una interrupción externa para controlar el sentido de la cuenta.

- **Pines:** Utilice un display de 7 segmentos conectado a un puerto de su elección. Un pulsador, conectado a un pin configurado como interrupción externa (EINT0, EINT1 o EINT2), controlará la dirección.
- **Lógica:**
    - Inicialmente, el contador debe avanzar de forma automática (ascendente).
    - Cuando el botón se mantenga presionado, el contador debe detenerse. Al soltar el botón, el sentido de la cuenta debe invertirse (descendente) a partir del valor actual.
    - Al volver a presionar el botón y soltarlo, la cuenta debe volver a ser ascendente.

4. Múltiples Interrupciones

Cree un programa que use dos fuentes de interrupción para controlar secuencias de LEDs diferentes.

- **Configuración:**
    - Configure el pin **P0.0** para generar una interrupción por flanco de subida.
    - Configure una interrupción externa (EINT1) en el pin **P2.11** por flanco de bajada.

- **Lógica:**
    - Cuando se detecta la interrupción del pin **P0.0**, se debe ejecutar una secuencia de 4 LEDs o 4 números si está utilizando un display de 7 segmentos
    - Cuando se detecta la interrupción externa del pin **P2.11**, se debe ejecutar una secuencia de parpadeo diferente.
- **Condiciones:** Configure la interrupción externa (EINT1) para que tenga una mayor prioridad que la interrupción del GPIO del Puerto 0.

5. Secuencia con Interrupción por Botón

Escriba un programa que ejecute una secuencia de 8 LEDs de forma automática. Cuando se presione un botón conectado a una entrada GPIO, la secuencia debe pausarse. Al volver a presionar el botón, la secuencia debe reanudarse.

- **Condiciones:**
    - Utilice una interrupción por flanco de bajada en el pin del pulsador.

6. Prioridades de Interrupción

- **Consigna:**
    - Si tiene dos interrupciones, una de la interrupción externa EINT2 y otra del GPIO del Puerto 2, y ambas ocurren al mismo tiempo, ¿cómo configuraría el NVIC para asegurar que la EINT2 se atienda siempre primero?
    - Qué registros del NVIC son necesarios para modificar la prioridad de una interrupción y explique todas las situaciones posibles que se pueden dar entre dos interrupciones con distinta prioridad y cómo maneja el NVIC cada una de las situaciones.
    - Escriba un pequeño fragmento de código que ilustre la configuración de estas prioridades.
    - ¿Qué prioridad tienen las interrupciones después de un reset?

Módulo 3 – SysTick

1. Interrupciones vs excepciones

- Describa la diferencia entre una interrupción y una excepción.
- ¿Las excepciones tienen prioridades configurables?
- ¿Se puede deshabilitar una excepción?

2. Parpadeo de LED con SysTick

Escriba una rutina simple para configurar e iniciar un Systick que pueda interrumpir cada 10 ms. Escriba también una rutina mínima del handler.

3. Parpadeo de LED con SysTick

Escriba un programa que haga parpadear un LED cada 500 ms utilizando el SysTick Timer.

- **Pines**: Utilice el pin P0.22 para el LED (activo por bajo).
- **Condiciones**:
  - Configure el SysTick para generar una interrupción cada 500 ms.
  - El programa principal (main) debe estar en un bucle infinito vacío, demostrando que la tarea de parpadeo se gestiona completamente por la interrupción.

4. Contador Hexadecimal Automático con SysTick

Reemplace el delay por software del ejercicio 1.4 por una interrupción del SysTick. El contador hexadecimal automático debe avanzar un dígito cada 1 segundo.

- **Pines**: 7 pines GPIO para el display de 7 segmentos.
- **Condiciones**:
  - Configure el SysTick para generar una interrupción cada 1 segundo.
  - La función principal solo debe encargarse de la inicialización y el bucle infinito. La actualización del display debe hacerse completamente en el handler de la interrupción

5. Múltiples Tareas Temporizadas

Escriba un programa que utilice el SysTick para gestionar dos tareas con diferentes periodos de tiempo.

- **Tareas**:
  - Un LED debe parpadear cada 500 ms.
  - Una secuencia de 4 LEDs debe avanzar un paso cada 200 ms.

6. Secuencia con SysTick y Botón de Inicio/Pausa

Combine el uso del SysTick y una interrupción externa para controlar una secuencia de LEDs.

- **Lógica:**
    - Un pulsador conectado a una interrupción externa (EINT0) iniciará o pausará una secuencia de 8 LEDs que avanza automáticamente cada 250 ms (controlado por SysTick).
    - El pulsador debe funcionar como un *toggle*, es decir, la primera pulsación inicia la secuencia y la segunda la pausa.

7. Contador con Interrupción Externa y Reset con SysTick

Escriba un programa que utilice una interrupción externa (EINT1) para incrementar un contador binario de 4 bits que se muestra en 4 LEDs. El contador debe resetearse a cero automáticamente cada 2 segundos, utilizando el SysTick.

- **Pines**: Pulsador en EINT1, 4 LEDs en un puerto GPIO.
- **Lógica**:
    - La interrupción externa (EINT1) incrementa el contador.
    - El SysTick se configura para generar un reset de la cuenta cada 2 segundos.

8. Semáforo con prioridad para peatones

Desarrolle un programa que implemente el control de un semáforo para una intersección de dos sentidos, utilizando el temporizador SysTick para gestionar los tiempos de la secuencia.

Lógica de los semáforos

La secuencia de los semáforos debe estar sincronizada y respetar los siguientes tiempos

ROJO (25 s)	**VERDE (25 s)**
ROJO (5 s)	**AMARILLO (5 s)**
VERDE (25 s)	**ROJO (25 s)**
AMARILLO (5 s)	**ROJO (5 s)**

Funcionalidad para Peatones:

- Implemente un pulsador para que un peatón solicite el cruce.
- Cuando se presione el botón, el semáforo debe ir al estado en el cual el semáforo de los autos está en amarillo y el del peatón en rojo.
- Después de 5 segundos en amarillo, el semáforo principal se pondrá en rojo, permitiendo que el peatón cruce.
- Una vez que se ha activado la solicitud del peatón, el ciclo debe continuar desde ese punto de manera normal, sin alterar los tiempos futuros de la secuencia principal.

Condiciones:

- Utilice SysTick para controlar todos los tiempos de las secuencias.
- El botón del peatón debe estar conectado a un pin GPIO configurado con interrupción externa para responder inmediatamente.
- El programa principal (main) debe estar en un bucle infinito vacío, demostrando que toda la lógica es gestionada por las interrupciones del SysTick y la interrupción externa.

Módulo 4 – Timers

1. Conceptos de Match y Capture

- Describa brevemente las funcionalidades de Match y Capture de los temporizadores de la LPC1769.
- Proporcione ejemplos de uso práctico para cada una de estas funciones.
- Explique en qué situaciones un temporizador genera interrupciones y qué registros están involucrados.

2. Toggle de LED con Match

Desarrolle un programa que configure un **Timer** para hacer parpadear un LED cada 1 segundo, utilizando la función de **Match** para alternar el estado del pin de salida.

Condiciones:

- Utilice el Timer0 de la LPC1769.
- Conecte el LED al pin de match MR0.0 del Timer0.

3. Generación de Onda Cuadrada con Frecuencia Variable

Escriba un programa que utilice un Timer para generar una onda cuadrada de 1 Hz. Agregue un botón que, al ser presionado, duplique la frecuencia de la onda. La frecuencia máxima debe ser de 1024 Hz, y la siguiente presión del botón debe reiniciar la frecuencia a 1 Hz.

Condiciones:

- Utilice un Timer de su elección y configure su canal de Match para generar la onda.
- El botón debe estar conectado a un pin GPIO configurado con interrupción externa.
- La frecuencia debe ser modificada en el Handler de la interrupción.

4. PWM con Duty Cycle Variable

Cree un generador de PWM donde el ciclo de trabajo se controle con un botón. El PWM debe iniciar con un 50% de ciclo de trabajo. Cada vez que se presione el botón, el ciclo de trabajo debe incrementarse en un 10%. Al alcanzar el 100%, la siguiente pulsación debe reiniciar el ciclo de trabajo a 0%.

Condiciones:

- Utilice el Timer2 para generar la señal PWM.
- Conecte el botón a un pin GPIO con interrupción externa para controlar el incremento del ciclo de trabajo.
- El periodo de la onda debe ser fijo, mientras que el ancho del pulso debe ser ajustable.

5. Medición de Tiempo entre Eventos

Desarrolle un programa que mida el tiempo que un botón se mantiene presionado. Para ello, utilice la función de Capture en el pin donde se conecta el botón. El tiempo de presión, en segundos, debe mostrarse en formato binario utilizando 4 LEDs.

Condiciones:

- Utilice el Timer3 y un canal de Capture.
- El botón debe estar conectado al pin asociado al canal de capture.
- Configure la captura para que se active en el flanco de bajada del botón.
- La lectura del tiempo se debe realizar en el Handler de la interrupción de captura.

6. Semáforo con Prioridad para Peatones

Realice el mismo ejercicio del semáforo con prioridad para peatones (ejercicio 3.8), pero esta vez utilizando uno de los Timers de la LPC1769 para controlar los tiempos de la secuencia, en lugar del SysTick.

Condiciones:

- Utilice un Timer para gestionar los tiempos de la secuencia.
- Configure la interrupción del temporizador para actualizar el estado del semáforo.
- El botón de peatón debe estar conectado a una interrupción externa.

7. Medición de Frecuencia

Escriba un programa que pueda medir la frecuencia de una señal externa. Utilice la función de Capture para determinar el período de la señal y, a partir de este valor, calcular la frecuencia. Los resultados deben mostrarse en displays de 7 segmentos.

Condiciones:

- Utilice uno de los Timers y su canal de Capture.
- Configure el canal de Capture para que se active solo en el flanco de subida de la señal externa.
- La señal de entrada del generador de funciones debe ser conectada al pin de Capture.
- Utilice displays de 7 segmentos para mostrar la frecuencia en Hz.

Módulo 5 – ADC

1. Lectura de Nivel Analógico y Visualización en LED

Escriba un programa que configure un canal del ADC para leer un voltaje variable.

- **Pines:** Utilice el pin **AD0.0** (P0.23) para la entrada analógica (conectado, por ejemplo, a un potenciómetro).
- **Salida:** El valor de 10 bits leído debe ser escalado a un valor de 1 bit para encender o apagar un LED. El LED debe encenderse si la lectura supera la mitad del rango (512).

Condiciones:

- Configure el ADC para disparar con el MAT0.1 cada 50 ms.
- No se deben usar interrupciones de timer.

2. Conversión Múltiple y Escaneo de Canal

Escriba un programa que lea tres canales analógicos distintos de forma secuencial.

- **Pines:** Utilice los pines **AD0.0** (P0.23), **AD0.1** (P0.24) y **AD0.2** (P0.25) como entradas.

Lógica:

- Utilice 4 LEDs para implementar un vúmetro por cada canal, de forma en que se encienda un LED adicional por cada 25% del nivel máximo.

Condiciones:

- Utilice un timer para disparar las conversiones de manera secuencial, con un intervalo entre conversión de 50 ms.

3. Control RGB mediante modulación PWM

Desarrolle un programa que permita al usuario controlar de forma independiente la intensidad (brillo) de los tres colores (Rojo, Verde, Azul) de un LED RGB, utilizando tres potenciómetros. La modulación de intensidad debe implementarse mediante PWM.

Pines:

- **Entradas ADC (Potenciómetros):** Utilice tres canales de ADC disponibles, por ejemplo: **AD0.0** (P0.23), **AD0.1** (P0.24) y **AD0.2** (P0.25).
- **Salidas PWM (LED RGB):** Utilice los pines de Match del Timer 2 (que soporta hasta 4 canales de Match) para generar las señales PWM. Asigne MR2.0, MR2.1 y MR2.2 para controlar los pines del LED RGB (Rojo, Verde, Azul).

Lógica:

1. Configure el ADC para escanear y convertir los tres canales de entrada de forma secuencial.
2. Configure el Timer 2 para una frecuencia PWM fija.
3. El valor de 10 bits de cada lectura de ADC debe ser mapeado y utilizado directamente para establecer el ciclo de trabajo (duty cycle) de la señal PWM correspondiente a cada color.
    - **ADC0.0** → Controla el Duty Cycle de **PWM (MR2.0)** - Color Rojo.
    - **ADC0.1** → Controla el Duty Cycle de **PWM (MR2.1)** - Color Verde.
    - **ADC0.2** → Controla el Duty Cycle de **PWM (MR2.2)** - Color Azul.

4. Muestreo de ADC Controlado por Evento y Temporizador (Sistema de Grabación por Demanda)

Desarrolle un programa que controle el muestreo de 100 muestras del ADC. La adquisición de datos es iniciada por una interrupción externa y cronometrada por un temporizador.

- **Pines:**
  - **Entrada ADC:** Utilice el pin **AD0.0** (P0.23) para la señal analógica.
  - **Botón de Control (E/S):** Utilice el pin **P2.10** configurado como interrupción externa **EINT0**.
  - **LED de Estado:** Utilice el pin **P0.22** para indicar el estado del sistema: encendido si el ADC está activo y muestreando, y apagado si está inactivo.

Lógica de Muestreo:

1. **Activación:** El primer flanco de **bajada** en el botón (EINT0) debe:
   - Encender el LED de Estado.
   - Configurar e iniciar un Timer para generar una interrupción o *match* cada 50 ms.
   - Habilitar **el ADC p**ara una sola conversión por evento de Timer.
2. **Muestreo:** Cada evento de *match* del Timer 1 debe disparar una nueva conversión del ADC.
3. **Almacenamiento:** El resultado de cada conversión del ADC debe ser leído por el CPU y almacenado secuencialmente en un *array* de 100 posiciones en la SRAM Bank 1.
4. **Finalización Normal:** Después de la muestra número 100, el Timer 1 y el ADC deben ser deshabilitados, y el LED de Estado debe apagarse.

Lógica de Control (Botón de Pausa/Reinicio):

1. Si se detecta un segundo flanco de bajada en el botón mientras la adquisición está en curso (antes de llegar a 100 muestras):
   - El muestreo debe detenerse inmediatamente.
   - El Timer 1 y el ADC deben ser deshabilitados.
   - El LED de Estado debe apagarse.
   - El sistema queda listo para un nuevo inicio.

Condiciones:

1. Asegure que las muestras de 10 bits del ADC se almacenen correctamente en un *array* de enteros de 16 o 32 bits, garantizando la correcta alineación de los bits de datos (0-11 o 0-12).
2. La gestión de la cuenta y el *toggling* del estado debe manejarse completamente dentro de la ISR de EINT0 y la ISR del Timer 1.

Módulo 6 – DAC

1. Cálculo de Parámetros de Generación de Onda

- **Consigna:** Deduzca una fórmula que relacione la frecuencia de la onda de salid**a** (F_{salida}), la cantidad de muestras utilizadas para definir esa onda ($N_{muestras}$), y la frecuencia máxima de actualización del registro DAC. Utilice esta fórmula para determinar si una configuración es viable.
- ¿Qué relevancia tiene el bit de bias en el DAC?.
- **Escenario Práctico:** Calcule la frecuencia máxima de la onda sinusoidal que se puede generar si se utiliza un *array* de 32 muestras y se asume que la frecuencia de muestreo del DAC es de **200 kHz**.
- **Condiciones:**
    - Muestre los cálculos matemáticos para la derivación de la fórmula y para la resolución del escenario.

2. Generación de Onda Sinusoidal con Bajo Consumo y Timer

- **Consigna:** Escriba un programa para generar una onda sinusoidal de baja frecuencia y demuestre el uso de la funcionalidad de bajo consumo del DAC.
- **Lógica:**
    1. Defina un array en la memoria con al menos 32 valores discretos que representen los puntos de una onda sinusoidal completa.
    2. Configure el DAC en modo de bajo consumo.
    3. Configure el Timer 1 para generar una interrupción que se active a una frecuencia constante (la frecuencia de muestreo).
- **Condiciones:**
    - La lógica de actualización del DAC (leer el siguiente valor del *array* y escribirlo en el registro DACR) debe residir completamente en el Handler de Interrupción del Timer 1.
    - La onda sinusoidal debe tener una frecuencia final de **50 Hz**.

3. Modulación de Voltaje por Software (Rampa Asimétrica)

- **Consigna:** Desarrolle un programa que simule un "pseudo-contador" de 10 bits en la salida del DAC.
- **Lógica:**
    1. La salida del DAC debe aumentar gradualmente de 0% a 100% del Vref.
    2. Al llegar al 100% del Vref (valor máximo), la salida debe **disminuir bruscamente** a 0% y reiniciar el ciclo.
- **Condiciones:**
    - El tiempo de incremento (0% a 100%) debe ser de 2 segundos.
    - El tiempo de decremento (100% a 0%) debe ser instantáneo (por la escritura del nuevo valor).
    - La temporización debe ser controlada por el Timer 2 configurado para generar una interrupción o *match* a una frecuencia que te permita completar los 1024 pasos en 2 segundos.
    - Calcule la frecuencia de interrupción necesaria para lograr los 2 segundos exactos.

Módulo 7 – GPDMA

1. Cálculo de 1. Generación de Patrón PWM Múltiple por DMA (Memoria → GPIO)

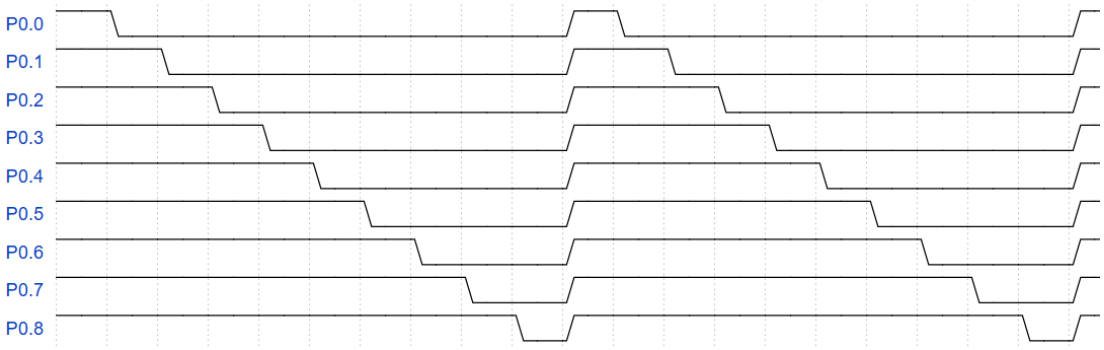
Consigna: Desarrolle un programa que utilice el DMA para generar un patrón de Modulación por Ancho de Pulso (PWM) en nueve pines de salida simultáneamente, cada uno con un ciclo de trabajo diferente.

Lógica:

- Cree un array en la memoria que contenga el patrón de salida digital deseado (donde cada bit representa el estado de un pin de salida en ese momento).
- El patrón en el array debe simular 9 ondas PWM con la misma frecuencia, pero con ciclos de trabajo incrementales: 10%, 20%, 30%, ..., hasta 90%.

Condiciones:

- El request de salida del DMA debe ser activado por un evento Match del Timer (e.g., MR0.0).
- La transferencia debe ser continua (utilizando el LLI en modo circular o configurando el TransferSize para el bucle).



2. Adquisición Cíclica de Alta Frecuencia con ADC (Periférico → Memoria)

Consigna: Desarrolle un sistema de adquisición de datos para una señal de alta frecuencia que garantice un muestreo continuo y cíclico.

Lógica:

- Configure el ADC para muestrear a una alta frecuencia, por ejemplo, 60 kHz, en modo burst o activado por un Timer.
- Configure un canal de DMA para transferir las muestras del ADC al SRAM Bank 1.
- El buffer de destino debe ser cíclico (circular).

3. Data Logging con Procesamiento y Visualización Lenta (Cadena de Periféricos)

Consigna: Cree un sistema de registro de datos que adquiera un lote de muestras del ADC a baja frecuencia, las procese, y muestre el resultado a través del DAC.

Lógica:

- Activación Lenta: Configure un Timer para generar una interrupción cada 1 minuto.
- Adquisición: Esta interrupción debe disparar al ADC para iniciar una transferencia de 20 datos a una ubicación en la SRAM Bank 0.

- Procesamiento: Configure el DMA para generar una interrupción al completar la transferencia de los 20 datos. El Handler de la Interrupción de DMA debe procesar estos 20 datos (e.g., calcular el promedio).
- Visualización: El resultado del procesamiento debe ser enviado al DAC y mantenerse en la salida durante 30 segundos.

Condiciones:

- El DMA debe manejar la transferencia de ADC → SRAM Bank 0.
- La lógica para el envío del resultado al DAC y la gestión del tiempo de visualización debe estar estrictamente separada de la adquisición de datos.