Práctica 0

Introducción y repaso

1. Manipulación de bits

Cree una función para “prender o apagar” bits de una variable. La función debe recibir una variable entera sin signo de 32 bits por referencia, un entero que indique el número de bit que debe modificarse de la variable, y otro entero que indique si debe ponerse en 1 o 0. En caso de no poder realizar la operación (porque se pasó un argumento erróneo), devuelva 0.

2. Manejo de estructuras

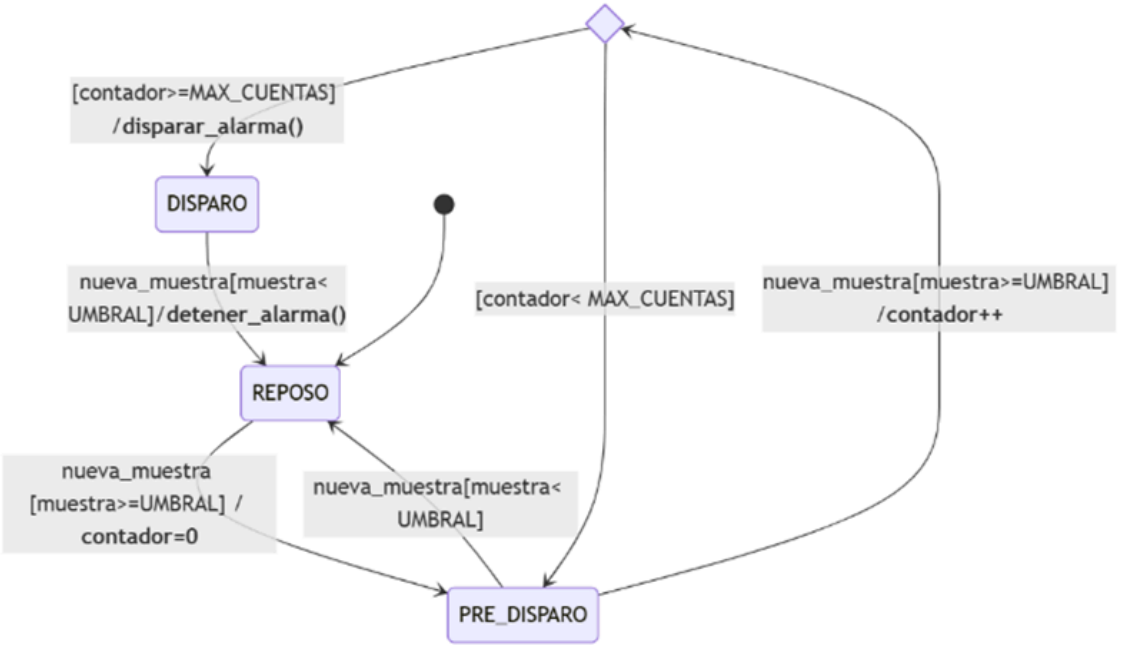
Cree un conjunto de variables y funciones que permita manipular los valores de un puerto de un periférico de entrada-salida de propósito general (GPIO) hipotético de un microcontrolador de 8 bits, que posee un registro donde se configura si cada pin es de entrada o salida, otro donde escribir valores que se quieran para los pines de salida y otro otro donde leer valores de los pines configurados como entradas. Para ello, siga los siguientes pasos:

a. Declare una estructura gpio_t con campos de tipo uint8_t reg_salida, reg_entrada, y reg_direccion. Declare una variable de este tipo.
b. Declare un enum gpio_result_e con valores gpio_ok y gpio_error
c. Cree una función gpio_configurar_direccion que sea capaz de recibir una estructura gpio_t por referencia, junto con un número de bit y un valor de 0 o 1 para definirlo como entrada o salida asignando ese valor a la posición correspondiente del campo reg_direccion. La función devolverá gpio_ok si el rango del número de bit es correcto o gpio_error en caso contrario.
d. Cree una función gpio_leer que reciba una estructura gpio_t por referencia y un número de bit y devuelva, por referencia, el valor de ese bit. Si el rango es erróneo o el bit es de salida, debe devolver gpio_error, caso contrario, gpio_ok.
e. Cree una función gpio_escribir completando la funcionalidad.
f. Cree dos variables de tipo gpio_t de nombre GPIOA y GPIOB y compruebe el funcionamiento de todos los aspectos del código creado.

3. Código en archivos separados

Convierta el código del ejercicio anterior en una librería. Pase las declaraciones de tipos y funciones a archivos .h y .c. El archivo .h debe contener guardas de inclusión adecuadas y el mismo código del inciso f debe seguir funcionando al incluir la librería.

4. Implemente en C la siguiente máquina de estados:



La máquina procesa valores de temperatura medidos consecutivamente y dispara una alarma por alta temperatura si el valor de las últimas MAX_CUENTAS temperaturas es mayor a un UMBRAL determinado. Cualquier muestra por debajo del umbral llevará el sistema a su estado de REPOSO y desactivará la alarma en caso de estar en estado de DISPARO.

a) Identifique: estados, pseudoestados, eventos, acciones y condiciones. Defina un nuevo tipo de dato enumerado para los estados llamado **al_estado_e**. (¿Es necesario definir una enumeración también para los eventos?) e implemente una primera versión de la máquina dentro de un esquema como el siguiente:

```
float muestras[NUM_MUESTRAS] =
    {31,32,33,34,35,36,37,38,39,40,41,42,43,42,41,40,39,38,37,36,35,37,38,20};

for(int i=0;i<NUM_MUESTRAS;i++){

   …

}
```

Tanto el estado como el contador de muestras consecutivas por encima del umbral serán variables globales. UMBRAL y MAX_CUENTAS serán constantes.
Agregue una función **al_debug(...)** a la que en cada ciclo de ejecución de la máquina se le puedan pasar los valores para que imprima una línea con el valor de entrada, el estado, y el valor de cuentas.

b) Cree un archivo *alarma.h* (con guardas de inclusión apropiadas) y otro *alarma.c* para trasladar la funcionalidad de la alarma, basada en la máquina de estados, a una librería. Se busca que la alarma provea una interfaz sencilla, basada en la función

```
al_estados_e  al_nueva_muestra(float m);
```

Implemente dicha función para que, cada vez que se la invoque, procese el evento correspondiente en la máquina de estados. *alarma.h* solo expondrá el tipo de dato para los estados, las funciones que componen la interfaz pública, y defines para configurar la librería, mientras que el resto de las funciones y variables globales no estarán disponibles a los otros módulos. Un define permitirá configurar si se imprime o no la información de debug.

c) **Opcional:** Cambie las funciones de disparo que implementan las acciones de la máquina por punteros a funciones para independizar la librería por completo de funciones específicas como printf.

5. Dibuje el esquemático de un circuito para conectar un pulsador normal abierto, que en reposo esté a 0 V y al pulsarlo a 3.3 V. Realice una simulación temporal del circuito en cualquier software de simulación (circuitlab.com, TINA TI (Texas Instruments), LTSpice (Analog Devices)) reemplazando el pulsador por una llave controlada por tiempo. Implemente el circuito y realice una medida utilizando el osciloscopio. Registre si hay rebotes y la amplitud de la perturbación resultante.

6. Se desea generar un valor de tensión constante utilizando una salida PWM de 10 kHz, 50% de ciclo de trabajo y 0 a 3.3 V de rango.
¿Qué tensión se espera ver a la salida?
Calcule dos filtros pasabajos RC con frecuencia de corte de (i) 160 Hz y (ii) 1.6 Hz y realice para cada uno una simulación de AC para evaluar si el cálculo fue correcto y una simulación temporal para observar la forma de onda de salida. Registre la amplitud del “ruido” resultante en cada caso.

7. Encuentre una hoja de datos para un LED rojo de 3 mm de montaje through-hole.Si quiere usarlo con un margen de seguridad del 80% de su disipación de potencia máxima en un sistema de 3.3 V ¿qué resistencia debe usar?. ¿Cambia si el LED tiene encapsulado SMD 0805? ¿Y si es verde? Cite las hojas de datos utilizadas.

8. Luego de dimensionar un proyecto y experimentar con hardware disponible se determina que para un sistema embebido se necesita un microcontrolador con más de 50 kB de RAM, 500 kB de Flash, 4 timers de 16 bits y al menos 2 ADCs de 16 bits (no dos canales, sino 2 ADCs independientes). Debe costar menos de US$8 en volúmenes de 10.000 unidades. Se cuenta con herramientas de desarrollo para la familia STM32. Encuentre un microcontrolador apropiado.

Creación de proyectos con el IDE

Recomendaciones para el uso del kit de la cátedra

*Cada kit contiene un **microcontrolador STM32F103C8T6** soldado a una placa de desarrollo llamada "Blue Pill". La placa contiene al microcontrolador (µC) y elementos accesorios para lograr una funcionalidad mínima, incluyendo pines que dan acceso al puerto de programación y alimentación para el sistema. **Notar que el conector micro-USB no sirve para programar el µC**, sino que puede usarse en un futuro cuando se quiera implementar una conexión a través de ese puerto.*

***La programación y depuración del firmware del µC se realiza utilizando el herramienta ST-LINK V2** que se encuentra en cada kit, la cual viene acompañada de 4 cables para conectar a la placa de desarrollo (PD)*

*La conexión entre el programador ST-LINK y la PD es sencilla ya que la función de los pines se encuentra etiquetada en ambos dispositivos. Sin embargo, **los pines no se encuentran en el mismo orden** y debe prestarse atención al conexionado.*

*Utilice el **cable extensor USB** para conectar la PD más fácilmente a la PC y evitar ejercer fuerzas sobre los pines de conexión.*

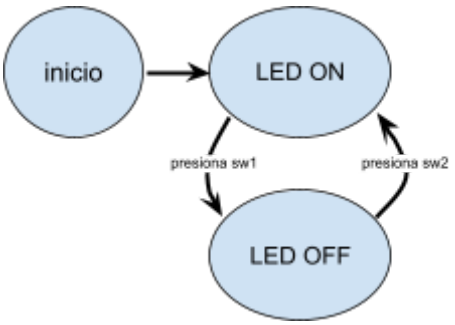
9. Cree un proyecto "Hola mundo" de sistemas embebidos. No avance al siguiente paso antes de realizar la comprobación pedida en la tabla:

Paso	Descripción	Check
1	Cree un proyecto en el STM32CubeIDE para el microcontrolador STM32F103C8T6 utilizando el asistente Cube MX. Configure el GPIO correspondiente al LED como salida. Genere el código y compruebe poder compilar el código y descargarlo	
2	Programe el código para que el LED permanezca siempre encendido. Verifique que se prende	
3	Programe el código para que el led destelle permaneciendo en cada período medio segundo encendido y dos segundos apagado. Utilice las funciones HAL_GPIO_WritePin para modificar el estado del LED y HAL_Delay para generar los retardos. Verifique su funcionamiento.	
4	A partir del esquemático de la placa, calcule la corriente que consume el LED cuando el GPIO está nivel en alto y en	

	bajo. Asuma que el LED encendido tiene una caída de 2V. Mida los valores.	

10. Agregando pulsadores

Paso	Descripción	Check
1	Seleccione dos pines en el esquemático de la placa tales que no estén conectados a ningún otro componente.	
2	Cree un nuevo proyecto como en el inciso anterior, configurando la salida para el led. Puede “copiar y pegar” el proyecto en el panel “Project Explorer” del IDE, pero debe cambiar manualmente el nombre al archivo .ioc para que coincida con el del nuevo proyecto y borrar el archivo .launch. Compruebe que puede compilar el proyecto.	
3	Configure dos GPIO como entradas. Configure uno con pull-up interno y el otro con pull-down interno. Lea el estado de los pines con la función HAL_GPIO_ReadPin y guarde en dos variables uint8_t globales SW1 y SW2. Utilizando el debug Compruebe que las variables toman los valores correctos según los pull up o down configurados.	
4	Conecte un pulsador a cada uno de los pines, asegurándose que cada pin cambie de estado al presionar el pulsador correspondiente y se corresponda con la respuesta lógica al depurar.	
5	Mejore el código anterior utilizando macros para definir los estados: PRESIONADO y NO_PRESIONADO. Compruebe que sigue funcionando	
6	Implemente la siguiente máquina de estados y verifique su funcionamiento	


inicio
LED ON
presiona sw1
presiona sw2
LED OFF