

# Práctica 1

Para la resolución de estos problemas necesitará el manual de referencia del STM32F103xx (RM0008) y la hoja de datos del microcontrolador.

## A. Análisis de la documentación

### 1. Direcciones de memoria

- Encuentre la dirección de memoria del periférico que controla el led de la placa de desarrollo ¿Cómo puede encender el LED?
- ¿Cómo puede hacer lo mismo usando la característica de bit-band?

### 2. Fuentes de clock

- ¿Cual es la tasa de error del clock HSI? Si se intentan contar 300 segundos con un programa basado en el clock HSI a 8 MHz, ¿entre qué valores es esperable que cuente efectivamente?. Si en cambio se utiliza un cristal de la misma frecuencia con un error de 10 ppm ¿Qué tiempos resultarían?

### 3. Interrupciones con prioridades

A partir del siguiente esquema de ejecución de interrupciones deduzca la configuración de los bits PRIGROUP del registro AIRCR y del registro BASEPRI. Los segmentos verdes significan que la interrupción se apropió de la ejecución y los grises que se pidió la interrupción pero no se apropió de la ejecución. La primera columna de la tabla corresponde al número de interrupción.

0110 0000									
0101 0000									
0100 0000									
0111 0000									
T0	T1	T2	T3	T4	T5	T6	T7	T8	T9

#### 4. Bocadillo de assembler

Cree un proyecto y defina la siguiente función:

```
int fun1(int a, int b){
    int x = 13;
    int c = 1;
    int y = 8;
    int z = x+c+y+a+b;
    return z;
}
```

Llámelas en el main() antes del bucle infinito del while recibiendo el valor devuelto en una variable global. Compile el proyecto y abra el archivo .list generado en la carpeta /Debug o use la vista de Disassembly en Window->Show View->Others durante una sesión de debug. Analice el código de assembler ayudándose con los comentarios generados por el compilador:

- Ubique la función
- Ubique el punto en el que se llama
- ¿Cómo se “pasan” los valores de los argumentos?
- ¿Qué acciones realiza antes de ejecutar la suma?
- ¿Cómo usa el stack?
- ¿Puede encontrar la sentencia que implementa el while(1)?

#### **B. Manipulación de periféricos a través de registros**

##### 5. GPIO I

Cree un proyecto nuevo y siga los pasos enumerados a continuación para manipular un pin GPIO con herramientas de bajo nivel. Antes de continuar con el paso siguiente, verifique usando el debugger que se lograron realizar las acciones pedidas.

- Habilitar el clock del periférico que se desea utilizar.** Esto se realiza mediante el hardware RCC (Reset and clock control) .

- Del *Reference Manual* obtenga la dirección de memoria base de los registros que configuran el RCC y defina una constante

```
#define RCC_BASE ...
```

- Averigue el bus de periféricos (APB1, APB2) al que está conectado el periférico de interés y el offset del registro encargado de habilitar el clock para los periféricos de dicho bus. Defina un puntero constante a dicho registro:

```
#define OFFSET_APBXENR ...
```

```
uint32_t *const RCC_APBXENR = RCC_BASE + OFFSET_APBXENR;
```

- c. el **bit** del registro que habilita el clock para el periférico deseado y habilite dicho periférico poniendo el bit en 1:

```
#define BIT_PERIFERICO ...
```

```
*reg = *reg | (1 << BIT_PERIFERICO);
```

2. Una vez habilitado el clock del periférico, puede **configurar el periférico**. Para un GPIO por ejemplo, debe configurar el bit como entrada o salida. Obtenga del *reference manual*:
  - a. La dirección base de los registros del GPIOx
  - b. El offset del registro de configuración correspondiente al pin que quiere configurar (GPIOx\_CRL para pines 0 a 7 y GPIOx\_CRH del 8 al 15)
  - c. los bits que deben configurarse para el pin de interés
3. Una vez configurado puede **manipular el periférico** a través de sus registros. Para el GPIO puede usar los registros GPIOx\_IDR, GPIOx\_ODR y máscaras para leer/escribir un bit particular, que se corresponde directamente con el estado del pin.
4. Pruebe si efectivamente logró realizar estos pasos correctamente poniendo un pin en estado alto y bajo.

## 6. GPIO II

A partir de lo realizado en el ejercicio anterior, configure el pin conectado al LED de la placa blue pill como salida y escriba un código que lo haga destellar a 10 Hz. La espera de tiempo puede hacerse con un bucle vacío que itera una cantidad determinada de veces.

Modularice el código para que el programa principal se vea de la siguiente manera:

```
#define ITER_ESPERA 400000
void main()
{
    configurar_led();

    while(1) {
        prender_led();
        esperar(ITER_ESPERA);
        apagar_led();
        esperar(ITER_ESPERA);
    }
}
```

Utilice la ventana de depuración correspondiente para visualizar los registros de periféricos, ejecute paso a paso y observe la modificación de los registros utilizados.

Utilice el analizador lógico para ajustar la frecuencia del destello a 10 Hz (ciclo de trabajo del 50%).

## 7. GPIO III

A la consigna del ejercicio anterior agréguele un pulsador que se conectará a un GPIO configurado como entrada con pull-down interno.

Modifique el código de tal modo que al estar el switch presionado el LED destelle a 1 Hz y al estar sin presionar lo haga a 10 Hz.

Organice el código de manera modular, en dos funciones:

```
void configurar_switch()//configura la entrada

uint8_t leer_switch() //retorna el nivel de pin (0 o 1)
```

## 8. TIMER

Partiendo del código del ejercicio 5, reemplace la función `void esperar(uint32_t)` por una nueva función llamada `void esperar_ms(uint32_t)` que bloquee la ejecución durante una cantidad de milisegundos pasada como parámetro.

Para que el tiempo de espera sea predecible se usará el periférico TIM3, configurado mediante una nueva función `void configurar_timer()` para una frecuencia de conteo de 1 MHz (deberá escalar el reloj principal utilizando el prescaler).

Verifique el cumplimiento de la consigna mediante medición de los tiempos.

Utilice una variable global para especificar los milisegundos pasados como parámetro y modifique mediante el debugger.

## 9. Una forma más elegante...como la HAL

Una forma más estandarizada de operar en la región de memoria de los periféricos es definir, para cada periférico, un tipo de dato con una estructura cuyos miembros sean variables de 32 bits, ordenadas según el orden de los registros en memoria.

Por ejemplo, en la página 194 del *Reference Manual RM0008* vemos que cada GPIO cuenta con 7 registros consecutivos en memoria: `GPIOx_CRL`, `GPIOx_CRH`, `GPIOx_IDR`, `GPIOx_ODR`, `GPIOx_BSRR`, `GPIOx_BRR` y `GPIOx_LCKR`. La forma de modelar ese espacio de memoria con una estructura sería entonces:

```
typedef struct{
    uint32_t CRL; //offset 0x00
    uint32_t CRH; //offset 0x04
    uint32_t IDR; //offset 0x08
    uint32_t ODR; //offset 0x0C
    uint32_t BSRR; //offset 0x10
    uint32_t BRR; //offset 0x14
    uint32_t LCKR; //offset 0x18
}GPIO_t;
```

De este modo, solo es necesario definir un único puntero

```
const GPIO_t* GPIOC = 0x40011000;
```

o una única constante simbólica para referenciar a la memoria física del grupo de registros del periférico:

```
#define GPIOC ((GPIO_t*)0x40011000) //GPIOC se comporta como un
                                     //puntero a estructura
```

Por ejemplo, con las siguientes líneas de código configuran como salida el pin 13 del GPIO y ponen en alto y luego en bajo el pin:

```
GPIOC->CRH &= ~(1<<20 | 1<<21 | 1<<22 | 1<<23); //4 bits de conf. en 0
GPIOC->CRH |= 1<<21; // establece PC13 como salida push-pull / 2MHz max
GPIOC->ODR |= 1<<13; // pone en alto PC13
GPIOC->ODR &= ~(1<<13); // pone en bajo PC13
```

Modifique el ejercicio 8 de esta guía para manipular los periféricos RCC, GPIOC y TIM3 con este nuevo esquema.

## **C. Interrupciones**

### **10. Interrupción externa**

Cree un nuevo proyecto y, usando el asistente gráfico y la HAL, configure con el asistente un pulsador conectado a un GPIO (entrada con pull-up). Asigne al pulsador una interrupción externa por flanco descendente. Implemente la siguiente función callback con un código que aumente el valor de un contador global cada vez que se ejecute. Analice lo que ocurre usando la característica de “live watch” en una sesión de depuración.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
```

### **11. Interrupción de Timers**

Cree un nuevo proyecto y configure con el asistente el TIM3 y el GPIO del led. Configure el TIM3 para que desborde cada segundo y habilite la interrupción.

Implemente la siguiente función callback con un código que invierta el estado del LED.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
```

## 12. Prioridades de interrupciones

- a. Conecte un pulsador a un GPIO configurado como entrada que dispare una interrupción. En la interrupción, ponga en alto un segundo GPIO durante 10 ms (usando la función HAL\_Delay). Es decir que cada vez que se presiona el pulsador, verá un pin en alto durante 10 ms. Compruebe lo realizado con el analizador lógico.
- b. Configure una interrupción de un timer con una prioridad menor a la del pulsador para dispararse cada 200 us. Cada vez que se dispara, invierta el estado de un tercer GPIO. Observe las salidas con el analizador lógico y verifique que cumple con sus expectativas en relación al disparo del timer. Obtenga una captura de pantalla que demuestre el funcionamiento.
- c. Invierta las prioridades y observe nuevamente las formas de onda. Obtenga una captura de pantalla que refleje la nueva situación.
- d. Si se quieren mantener las prioridades originales, pero quiere evitarse que el timer se vea relegado, ¿cómo podría mejorarse el código?