Práctica 4

Proceso de compilación, modularización de librerías de STM y propias

Objetivos de la práctica: Comprender el proceso de compilación; comprender el uso de la memoria del microcontrolador por parte de C; comprender las ventajas de modularizar el código en librerías reutilizables; lograr agrupar las funciones en diferentes niveles de abstracción o capas; identificar las funciones expuestas (API) de las privadas (estáticas); modelar la información utilizada en estructuras de tipo "handle"; comprender modelos de librerías de STM.

1. Compilación de proyecto con varios módulos fuente

Cree un programa que haga titilar un LED a una frecuencia conocida, utilizando un timer para contabilizar el tiempo. Para hacerlo, el timer se deberá configurar para producir un evento de update cada 1 ms y una función permitirá esperar x ms contabilizando x ocurrencias del evento. Realice la implementación siguiendo los siguientes pasos:

a) Sin utilizar el IDE, modularize el código en cuatro archivos diferentes:
   - main.c: sólo contendrá la función main() con el algoritmo principal y la variable global con los ms a esperar

```
volatile uint32_t ms = 100;
int main()
{
configura_led();
configura_timer();
while(1){
   prende_led();
   //espera(ITER_ESPERA);
   espera_ms(ms);
   apaga_led();
   //espera(ITER_ESPERA);
   espera_ms(ms);
}
}
```

   - led.c: funciones de inicialización, encendido y apagado del led
   - tmr.c: funciones de inicialización y espera de ms.

```
#define RCC_APB1ENR *((volatile uint32_t *)(RCC_BASE + 0x1C))
#define RCC_APB1ENR_TIM3EN (1<<1)
#define TIM3_BASE 0x40000400 /** Reference manual - p. 50 **/
#define TIM3_CR1 *((volatile uint32_t *)(TIM3_BASE + 0x00))
#define TIM3_PSC *((volatile uint32_t *)(TIM3_BASE + 0x28))
#define TIM3_ARR *((volatile uint32_t *)(TIM3_BASE + 0x2C))
#define TIM3_EGR *((volatile uint32_t *)(TIM3_BASE + 0x14))

void configura_timer(){
// Habilita el clock del TIM3
RCC_APB1ENR |= RCC_APB1ENR_TIM3EN;
```

```
//configura TMR3
// Resetea CR1 por las dudas
TIM3_CR1 = 0x0000;
// fCK_PSC / (PSC[15:0] + 1)
// 8 Mhz / 7 + 1 = 1 Mhz
TIM3_PSC = 7;
// Seteo ARR al máximo valor, total el contador se resetea al llegar a 1000
TIM3_ARR = 0xFFFF;
// Habilito el TIM3
TIM3_CR1 |= (1 << 0);
// Genero evento de upgrade que lleva el periférico a un estado conocido (EVITA BUG AL RESTEAR)
TIM3_EGR |= (1 << 0);
}


#define TIM3_CNT *((volatile uint32_t *)(TIM3_BASE + 0x24))
void espera_ms(uint32_t ms)
{
// Habilito el TIM3
TIM3_CR1 |= (1 << 0);
for(; ms>0; ms--){
   TIM3_CNT = 0;
   while(TIM3_CNT < 1000);
}
// Resetea CR1 por las dudas
TIM3_CR1 = 0x0000;
}
```

- startup.c: tabla de vectores y código de inicialización.

Puede utilizar los códigos vistos en teoría para implementar esta funcionalidad.

b) Desde la línea de comandos, compile separadamente cada uno de los archivos fuente y luego enlace todos los archivos objeto para obtener el ejecutable. Utilice un linker script apropiado para el enlazado.
c) A partir de las herramientas de diagnóstico del toolchain, realice una tabla indicando dirección en memoria y tamaño en bytes de cada función. Haga lo mismo para la variable global que indica los milisegundos que se le pasan como parámetro a la función de espera.
d) Utilice el programa STM32 Cube Programmer para grabar el ejecutable en el microcontrolador y verifique el funcionamiento.

2. Agregando interrupciones

a) A partir de la Tabla 63 del manual de referencia del STM32F1x (RM0008) y de la Figura 12 del STMF1x… Cortex®-M3 programming manual (PM0056) identifique la entrada en la tabla de vectores que le corresponde a la interrupción global del TIM3.
Modifique el archivo startup.c del ejercicio 1 de manera de completar la tabla de vectores hasta (por lo menos) la interrupción global del TIM3 y asígnele la dirección en memoria de una función llamada `TIM3_IRQHandler`, que la definirá en otro módulo.

Nota: Conserve sin modificaciones las dos primeras posiciones de la tabla para el MSP y el PC. Para el resto de las excepciones de la tabla asigne una rutina de interrupción por defecto, que “cuelgue” el programa en un bucle infinito. A las posiciones de la tabla que figuren como “reservadas” en la documentación asignarles el valor cero.

b) En el archivo tmr.c del Ejercicio 1, modifique la definición de la función `configura_timer()` para que el contador del TIM3 se reinicie automáticamente cada un segundo.

Nota: Modifique por un lado el prescaler para que la frecuencia de entrada al contador sea 10 kHz y por el otro el auto-reload register para que el contador se reinicie cada 10000 cuentas (1 segundo). Cada vez que se reinicie el contador del TIM3 se producirá un "Upgrade Event" (UEV). La función `espera_ms()` dejará de funcionar correctamente pero no la utilizaremos.

c) Agregue al archivo tmr.c las funciones:
   - `habilita_irq_timer()` que habilite la interrupción cada vez que se produce un UEV (registros del TIM3) y habilite la interrupción global del TIM3 (registros del NVIC)
   - `limpia_flag_irq_timer()` que limpie el flag (ponga en cero el bit) correspondiente la interrupción debido al UEV (registros del TIM3)

d) Reemplace main.c por el siguiente código, genere el ejecutable y descargue a flash:

```
#include <stdint.h>
void configura_led();
void prende_led();
void apaga_led();
void configura_timer();
void habilita_irq_timer();
void limpia_flag_irq_timer();
void TIM3_IRQHandler();

uint8_t led = 0;

void main(){
    configura_led();
    configura_timer();
    habilita_irq_timer();
    for(;;);
}
void TIM3_IRQHandler(){
    limpia_flag_irq_timer();
    if(led==0){ prende_led();   led=1;}
    else{       apaga_led();    led=0;}
}
```

3. Entendiendo qué hace el IDE

Genere un nuevo proyecto en STM32CubeIDE (File->new->STM32Project - C/Executable/Empty project)

Antes de compilar por primera vez, Identifique entre los elementos que provee el IDE con el proyecto:

- Archivo/s fuente
- Definición de la tabla de vectores y ubicación en memoria
- Reset_handler y acciones que realiza
- Linker script y secciones de salida

Luego compile y observe la salida en la consola integrada en el IDE. Identifique el tipo de compilación (comando único, o compilación por módulos y enlazado).

Enumere tanto las opciones y flags de compilación como las de enlazado. Investigue las que no fueron explicadas en los ejemplos de la clase. Una buena referencia es la documentación de gcc.

Haciendo click con el botón derecho sobre el proyecto, ingrese a la opción Properties y navegue las distintas configuraciones, especialmente dentro de la jerarquía de configuraciones de C/C++ Build.

Modifique las opciones que desee y observe cómo cambian los comandos de compilación y la memoria ram/flash que consume la aplicación.

5. Driver para LEDs

Cree una librería de dos niveles (fundada sobre la HAL de STM32) que permita controlar múltiples LEDs. Cada LED se identificará mediante un handler que debe contener los siguientes campos:

- una estructura anidada con la información del GPIO físico: Puerto y pin
- un flag que identifique si el led es de lógica invertida o directa
- un flag que indique el estado actual (encendido/apagado/no-inicializado)

El driver debe exponer la siguiente API:

- una función de inicialización que configure el GPIO y el tipo de lógica. La función debe apagar el led luego de configurarlo.
- una función de encendido
- una de apagado
- una de toggle
- una función de des-inicialización, que configure el GPIO como entrada flotante y actualice el estado.

Todas las funciones deberán recibir por referencia el handler del LED.

6. LED parpadeante

Cree una librería de un único nivel (fundada sobre la HAL de STM32 y la librería del Ej. 5) que haga uso de un timer a elección y el LED de la placa blue-pill.

El LED puede estar en tres estados: prendido/apagado permanentemente y parpadeante.

Las funciones de la API deben ser:

- void LED_parp_init() /*configura led de la placa (PC13) y timer a elección */
- void LED_parp_on()
- void LED_parp_off()
- void LED_parp_parp(uint16_t periodo_ms)