

Ficheros que contiene la práctica 3

E/S por interrupciones

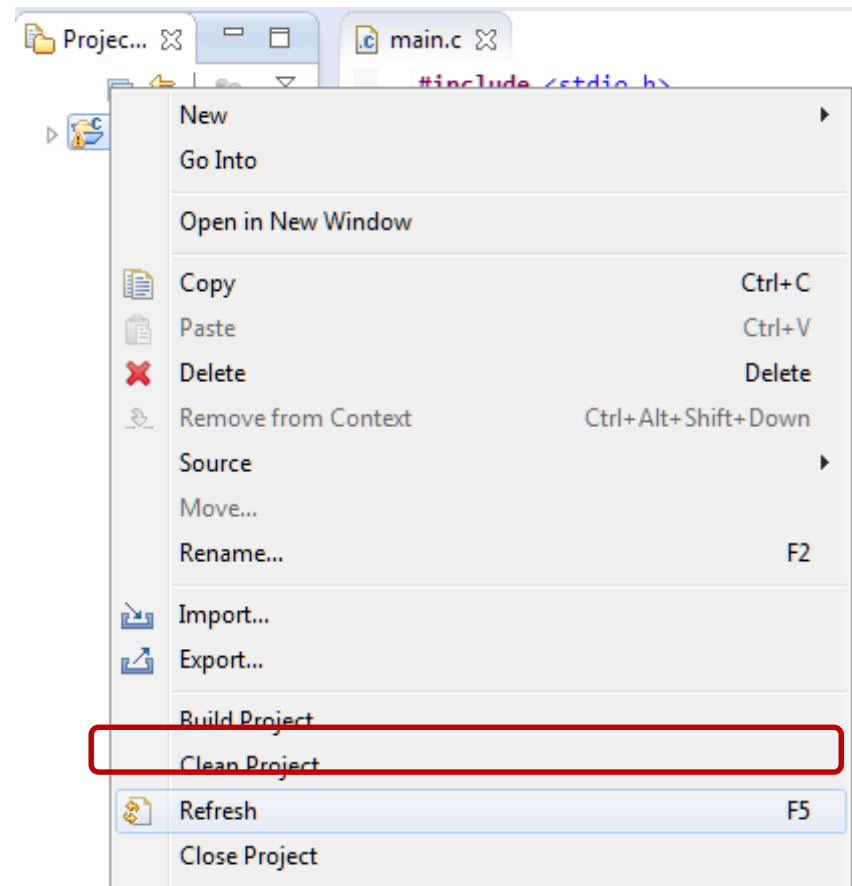
Ficheros que contiene la práctica 3



- Tenéis que **añadir** los siguiente **ficheros** que habéis hecho en la **práctica 2**
 - **gpio.h y gpio.c**: interfaz e implementación de funciones para el **manejo de los puertos B y G del controlador GPIO**
 - **leds.c y leds.h**: ficheros para implementar las funciones para **el manejo de los leds**
 - **D8Led.h y D8Led.c**: ficheros para implementar las funciones para el **manejo del display 8 segmentos**
 - **button.c y button.h** **NO se necesitan** porque **ahora los botones van a generar interrupciones**

Para añadir los ficheros de la práctica 2

- 1- **Añadir** estos ficheros al workspace **desde el explorador**
- 2- Abrir el proyecto con el Eclipse →ratón encima del proyecto → botón derecho → **refresh**



Ficheros que contiene la práctica 3



- **intcontroller.h e intcontroller.c**: ficheros para implementar las funciones que manejan el controlador de interrupciones

Variables definidas **intcontroller.h**

```
enum int_mode {  
    IRQ = 0,  
    FIQ = 1  
};
```

```
enum int_vec {  
    VEC = 0,  
    NOVEC = 1  
};
```

Todas las líneas que
gestiona el controlador
de interrupciones

```
enum int_line {  
    INT_ADC      = 0,  
    INT_RTC      = 1,  
    INT_UTXD1    = 2,  
    INT_UTXD0    = 3,  
    INT_UTXD0    = 3,  
    INT_SIO      = 4,  
    INT_IIC      = 5,  
    INT_URXD1    = 6,  
    INT_URXD0    = 7,  
    INT_TIMER5   = 8,  
    INT_TIMER4   = 9,  
    INT_TIMER3   = 10,  
    INT_TIMER2   = 11,  
    INT_TIMER1   = 12,  
    INT_TIMER0   = 13,  
    INT_UERR01   = 14,  
    INT_WDT      = 15,  
    INT_BDMA1    = 16,  
    INT_BDMA0    = 17,  
    INT_ZDMA1    = 18,  
    INT_ZDMA0    = 19,  
    INT_TICK     = 20,  
    INT_EINT4567 = 21,  
    INT_EINT3    = 22,  
    INT_EINT2    = 23,  
    INT_EINT1    = 24,  
    INT_EINT0    = 25,  
    INT_GLOBAL   = 26  
};
```

Ficheros que contiene la práctica 3



■ **intcontroller.h e intcontroller.c**

- El alumno deberá **completar las siguientes funciones** :
 - **void ic_init** (void)
 - Con esta función **se inicializa el controlador de interrupciones**. Poner:
 - **int ic_conf_irq** (enum enable **st**, enum int_vec **vec**)
 - Con esta función **se configuran las interrupciones IRQ**
 - **st** indica si se quieren habilitar o no estas interrupciones
 - **vec** indica si se quieren Vectorizadas o NO vectorizadas
 - **int ic_conf_fiq** (enum enable **st**)
 - Con esta función **se configuran las interrupciones FIQ**
 - **st** indica si se quieren habilitar o no estas interrupciones

Importante

tenéis que saber **con cuál de los registros trabaja cada función**

Ficheros que contiene la práctica 3



■ **intcontroller.h e intcontroller.c**

– El alumno deberá **completar las siguientes funciones** :

- **int ic_conf_line** (enum int_line **line**, enum int_mode **mode**)
 - Con esta función **se configura si la línea activa el modo IRQ o FIQ**
 - **line** indica una de las 26 líneas del controlador de interrupciones
 - **mode** indica el modo que se quiere utilizar
- **int ic_enable** (enum int_line **line**)
 - Con esta función **se habilita la interrupción** por la línea dada en line
- **int ic_disable** (enum int_line **line**)
 - Con esta función **se deshabilita la interrupción** por la línea dada en line
- **int ic_cleanflag** (enum int_line **line**)
 - Con esta función **se borra la interrupción pendiente** de la línea dada en line

NOTA: se puede usar la función **INT_BIT(line)**: Devuelve un nº que tiene un “1” en el bit que indica line. Ej. : INT_BIT(21) devuelve 000001000000000000000000000000= 0x0200000

Importante

tenéis que saber con cuál de los registros trabaja cada función

Ficheros que contiene la práctica 3



- **timer.h y timer.c:** ficheros para implementar las funciones del módulo que maneja los temporizadores

Variables definidas timer.h

```
enum tmr_timer {  
    TIMER0 = 0,  
    TIMER1 = 1,  
    TIMER2 = 2,  
    TIMER3 = 3,  
    TIMER4 = 4,  
    TIMER5 = 5  
};
```

```
enum tmr_div {  
    D1_2 = 0,  
    D1_4 = 1,  
    D1_8 = 2,  
    D1_16 = 3,  
    D1_32 = 4,  
    EXTCLK = 5,  
    TCLK = 6  
};
```

```
enum tmr_mode {  
    ONE_SHOT = 0,  
    RELOAD = 1  
};
```

Ficheros que contiene la práctica 3



■ timer.h y timer.c

– El alumno deberá **completar las siguientes funciones** :

- **int tmr_set_mode** (enum tmr_timer **t**, enum tmr_mode **mode**)
 - Con esta función **se configura el modo del temporizador**
 - **mode** indica el modo: ONE-SHOT o auto RELOAD
- **int tmr_update** (enum tmr_timer **t**)
 - Con esta función **activa la carga de los registros cuenta y comparación**
- **int tmr_start** (enum tmr_timer **t**)
 - Con esta función **se activa el temporizador**
- **int tmr_stop** (enum tmr_timer **t**)
 - Con esta función **se para el temporizador**

NOTA: En todas estas funciones **t** indica el temporizador que se quiere inicializar

Importante

tenéis que saber **con cuál de los registros del timer trabaja cada función**

Ficheros que contiene la práctica 3



■ timer.h y timer.c

– El alumno deberá **completar las siguientes funciones** :

- **int tmr_set_prescaler** (int **p**, int **value**)
 - Con esta función se **asigna un valor de pre-escalado para el temporizador**
 - **p** indica el temporizador
 - **value** indica el valor de pre-escalado que se quiere aplicar (nº entre 0 y 255)
- **int tmr_set_divider** (int **d**, enum tmr_div **div**)
 - Con esta función se **asigna un valor al divisor de frecuencia del temporizador**
 - **d** indica el temporizador
 - **div** indica el valor del divisor de frecuencia que se quiere aplicar
- **int tmr_set_count** (enum tmr_timer **t**, int **count**, int **cmp**)
 - Con esta función se inicializan el valor de la cuenta a **count** y el valor de comparación a **cmp**
 - **t** indica el temporizador que se quiere inicializar

Importante

tenéis que saber **con cuál de los registros del timer trabaja cada función**

Ficheros que contiene la práctica 3



- **init.asm**: fichero de inicialización, está en ensamblador
 1. **Pasa a modo supervisor**
 2. Desde el modo supervisor **inicializa la pila (SP) de todos los modos** de ejecución privilegiados
 3. En el controlador de interrupciones **deshabilita todas las interrupciones**
 4. **Habilita** las líneas **IRQ y FIQ** del procesador
 5. **Inicializa la tabla de las direcciones** de la subrutina de tratamiento de interrupción **RTI** que vamos a llamar **irq_ISR**
 6. **Elimina** posibles **interrupciones pendientes**
 7. **Llama la rutina main** de nuestro programa
- **En este fichero el alumno tiene que programar en ensamblador la RTI irq_ISR**
 - Tiene que reconocer qué periférico ha producido la interrupción

Programa principal de la práctica 3



- **main.c**: fichero con el código del programa principal de la práctica 3
 - Este fichero contiene las siguientes funciones que **deberán ser codificadas** por el **alumno**
 - **int setup** (void)
 - Configurar los controladores HW de los dispositivos que vamos a manejar
 - **void timer_ISR** (void)
 - Función para tratar la interrupción del temporizador
 - **void button_ISR** (void)
 - Función para tratar la interrupción de los pulsadores
- La función **void irq_ISR** (void)
 - Esta escrita en ensamblador y por lo tanto en el fichero init.asm

**No son propiamente RTI,
ya que son llamadas
desde irq_ISR**
Deben declararse como
una función corriente

Programa principal de la práctica 3



■ `int setup (void):`

- **Inicializar los leds** y configurarlos para que sean pines de salida
 - Utilizar las funciones definidas en `leds.c`
- **Inicializar el Display** 8 segmentos
- **Configurar los pulsadores:** Utilizar las funciones definidas en `gpio.c` para:
 - Activar interrupciones (EINT6 y EINT7)
 - Que la interrupción se detecte en flanco de bajada
 - Activar la resistencia pull-up
- **Configurar el TIMER0** para que genere interrupciones periódicas (2s). Utilizar las funciones definidas en `timer.c`
 - Factor de división: 1/8
 - Pre-escalado: 255
 - Inicializar la cuenta con 62500
 - Valor de comparación puede ser cualquier valor mayor que 0 y menor que cuenta
- **Configurar el controlador de interrupciones.** Utilizar las funciones definidas en `incontroller.c`
 - Activar la línea IRQ en modo no vectorizado
 - Deshabilitar la línea FIQ
 - Configurar las líneas TIMER0 y EINT4567 por la línea IRQ del procesador y las deje habilitadas