Diseño Basado en Microprocesadores
Práctica 9
Interrupciones en el LPC4088

Índice

---

1. Objetivo

El objetivo de esta práctica es poner en funcionamiento el sistema de interrupciones del microcontrolador LPC4088.

2. Entradas de interrupciones externas

El microcontrolador LPC4088 posee cinco entradas de interrupción externa llamadas EINT0, EINT1, EINT2, EINT3 y NMI.

- La entrada de interrupción EINT0 está disponible en los pines

  P0[29]/USB_D+1/**EINT0**

  P2[10]/**EINT0**/NMI

En el pin P2[10] también está disponible la entrada de interrupción no enmascarable NMI.

- La entrada de interrupción EINT1 está disponible en los pines

  P0[30]/USB_D-1/**EINT1**

  P2[11]/**EINT1**/SD_DAT[1]/I2S_TX_SCK/LCD_CLKIN

- La entrada de interrupción EINT2 está disponible en el pin

  P2[12]/**EINT2**/SD_DAT[2]/I2S_TX_WS/LCD_VD[4]/LCD_VD[3]/LCD_VD[8]

  LCD_VD[18]

- La entrada de interrupción EINT3 está disponible en el pin

  P2[13]/**EINT3**/SD_DAT[3]/I2S_TX_SDA/LCD_VD[5]/LCD_VD[9]/

  LCD_VD[19]

2.1. Interrupciones a través de los pines de los puertos 0 y 2

Además de las entradas de interrupción externa EINT0, EINT1, EINT2, EINT3 y NMI, todos los pines de los puertos 0 y 2 pueden usarse también como entradas de interrupción (Nótese los pines P0[29], P0[30], P2[10], P2[11], P2[12] y P2[13] pueden hacer también las funciones de entradas de interrupción específicas EINT0, EINT1, EINT2, EINT3 y NMI).

Todas las interrupciones que se producen en alguno de los pines de los puertos 0 y 2 desembocan en la ejecución de la misma función de servicio de interrupción, la cual se llama `GPIO_IRQHandler`. Por tanto, en caso de que se hayan habilitado como entradas de interrupción más de un pin de los puertos 0 y/o 2, dentro de la función `GPIO_IRQHandler` será necesario comprobar cuál de ellos ha sido el causante de la interrupción.

En la figura 1 se muestra el camino se siguen las señales de interrupción a través de los pines de los puertos 0 y 2 y los registros implicados.

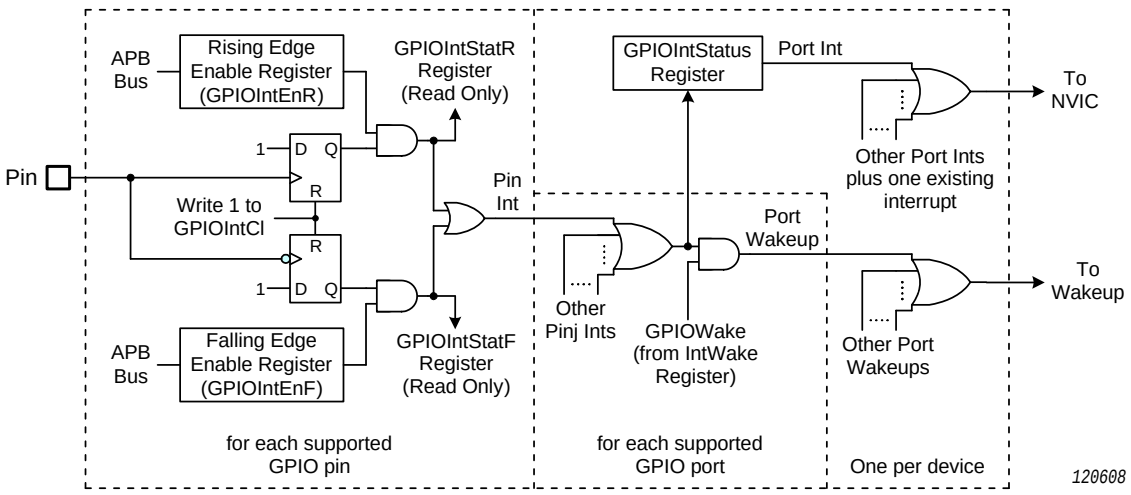

Figura 1: Diagrama de bloques de las interrupciones de los puertos 0 y 2.

Los registros usados para configurar y manejar las interrupciones de los puertos 0 y 2 están situados en el bloque `LPC_GPIOINT`. A continuación se describen cada uno de los registros.

2.2. STATUS

Tiene implementados únicamente los bits 0 y 2. Si el bit 0 de este registro está a uno indica que hay una petición de interrupción pendiente solicitada por alguno de los pines del puerto 0. Análogamente, si el bit 2 de este registro está a uno indica que hay una petición de interrupción pendiente solicitada por alguno de los pines del puerto 2.

2.3. STATR0 y STATR2

Estos registros permiten conocer si se ha producido una petición de interrupción por la aplicación de un flanco de subida a alguno de los pines del correspondiente puerto. Si un determinado bit del registro STATR0 está a uno, indicará que se ha aplicado una señal de interrupción por flanco de subida al correspondiente bit del puerto 0. Los bits del registro STATR2 comunican la misma información en relación con los pines del puerto 2. Ambos registros son sólo de lectura.

2.4. STATF0 y STATF2

Estos registros permiten conocer si se ha producido una petición de interrupción por la aplicación de un flanco de bajada a alguno de los pines del correspondiente puerto. Si un determinado bit del registro STATF0 está a uno, indicará que se ha aplicado una señal de interrupción por flanco de bajada al correspondiente bit del puerto 0. Los bits del registro STATF2 comunican la misma información en relación con los pines del puerto 2. Ambos registros son sólo de lectura.

2.5. CLR0 y CLR2

Escribiendo en estos registros pueden borrarse las peticiones de interrupción de los puertos 0 y 2 independientemente del flanco que las haya provocado. Al escribir un dato en el registro CLR0 resultan borrados aquellos bits de los registros STATR0 y STATF0 que se corresponden con bits a uno en el dato escrito. El registro CLR2 opera de forma análoga en relación a los registros STATR2 y STATF2. Ambos registros son sólo de escritura.

2.6. ENR0 y ENR2

Estos registros se ocupan de habilitar interrupciones por flanco de subida en los pines de los respectivos puertos. Poniendo a uno un determinado bit del registro ENR0 se habilitará la generación de una interrupción cuando se aplique un flanco de subida al correspondiente pin del puerto 0. El registro ENR2 actúa de forma análoga en relación a los pines del puerto 2.

2.7. ENF0 y ENF2

Estos registros se ocupan de habilitar interrupciones por flanco de bajada en los pines de los respectivos puertos. Poniendo a uno un determinado bit del registro ENF0 se habilitará la generación de una interrupción cuando se aplique un flanco de bajada al correspondiente pin del puerto 0. El registro ENF2 actúa de forma análoga en relación a los pines del puerto 2.

3. Nombres de los registros de interrupción de GPIO en μVision

En el fichero `LPC407x_8x_177x_8x.h` los nombres de los registros que controlan las interrupciones de los puertos 0 y 2 difieren de los que aparecen en el manual del LPC4088. La correspondencia entre unos y otros se indica en el cuadro 1.

Cuadro 1: Correspondencia entre los nombres de registros de interrupciones de GPIO en el manual del LPC4088 y en `LPC407x_8x_177x_8x.h`

Nombre en el manual	Nombre en `LPC407x_8x_177x_8x.h`
STATUS	IntStatus
STATR0	IO0IntStatR
STATR2	IO2IntStatR
STATF0	IO0IntStatF
STATF2	IO2IntStatF
CLR0	IO0IntClr
CLR2	IO2IntClr
ENR0	IO0IntEnR
ENR2	IO2IntEnR
ENF0	IO0IntEnF
ENF2	IO2IntEnF

Si preferimos usar los nombres de registros que aparecen en el manual del microcontrolador, podemos incluir el siguiente fragmento (después de los include):

```
typedef struct
{
    __I  uint32_t STATUS;
    __I  uint32_t STATR0;
    __I  uint32_t STATF0;
    __O  uint32_t CLR0;
    __IO uint32_t ENR0;
    __IO uint32_t ENF0;
    uint32_t RESERVED0[3];
    __I  uint32_t STATR2;
    __I  uint32_t STATF2;
    __O  uint32_t CLR2;
    __IO uint32_t ENR2;
    __IO uint32_t ENF2;
} LPC_GPIOINT_TypeDef_corregido;
```

```
#undef  LPC_GPIOINT
#define LPC_GPIOINT ((LPC_GPIOINT_TypeDef_corregido *) LPC_GPIOINT_BASE)
```

4. Ejercicios

4.1. Ejercicio 1

En este ejercicio realizaremos el programa de cronómetro que desarrollamos en la práctica anterior pero esta vez haciendo que el evento de match con MR0 del timer 0 genere una interrupción. De esta forma, la CPU no tendrá que comprobar constantemente la finalización de cada periodo de un segundo. En lugar de ello, una vez configurado convenientemente el timer 0 para solicitar interrupciones a intervalos de un segundo y configurada y habilitada su interrupción en el NVIC, la CPU puede llevar a cabo cualquier tarea sin tener que estar pendiente del timer. A intervalos de un segundo, el timer 0 solicitará una interrupción que hará que la CPU abandone momentáneamente la tarea que está realizando y ejecute automáticamente la función manejadora de interrupción correspondiente. Dentro del manejador de interrupción, se actualizarán las variables `segundos`, `minutos` y `horas` y se imprimirá el cronómetro en pantalla. Después de cada ejecución del manejador de interrupción, la ejecución retornará automáticamente al punto del programa en el que la ejecución fue suspendida para que la CPU continúe con la tarea que estaba llevando a cabo.

Partiendo del proyecto de μVision disponible en el campus virtual de la asignatura completa el fichero `main.c` siguiendo los pasos que se indican.

En la función `main`:

1. Inicializa la pantalla LCD llamando a la función `glcd_inicializar`.
2. Inicializa el timer 0 llamando a la función `timer_inicializar` que desarrollaste en la práctica anterior.
3. Programa el timer 0 para generar una petición de interrupción cada segundo. Para ello puedes usar la función `timer_iniciar_ciclos_ms`. Recuerda que esta función programa el timer indicado para que periódicamente se produzca un *match* del contador del timer con el registro de match MR0. Cada vez que esto ocurre, el registro contador del timer se pone a cero (para que comience un nuevo ciclo) y se solicita una interrupción. Tal como se describió en la práctica anterior, esta función se encarga de que el estado inicial del flag de petición de interrupción de match 0 del registro IR del timer sea 0.
4. Programa el NVIC para configurar y habilitar la interrupción del timer 0:
   a) Primero, es conveniente asegurarse de que no hay ninguna petición de interrupción del timer 0 pendiente de procesar. En este programa sencillo no es probable que esto ocurra pero, en un programa más complejo donde el timer se use en diversas partes del mismo, podría ser que el timer haya generado anteriormente una interrupción que haya quedado sin procesar por no encontrarse habilitada en ese momento. En ese caso, el NVIC habría mantenido la interrupción del timer en estado de **pendiente** y, en cuanto habilitásemos la interrupción, se dispararía inmediatamente una llamada al manejador de interrupción. Este comportamiento no sería correcto porque en nuestra aplicación de cronómetro se consideraría

erróneamente que ha transcurrido ya un segundo de tiempo. Por tanto, para asegurar que el NVIC "olvida" cualquier petición de interrupción del timer 0 en estado pendiente, llama a la función `NVIC_ClearPendingIRQ`:

```
NVIC_ClearPendingIRQ(TIMER0_IRQn);
```

b) Asigna la prioridad 1 a la interrupción del timer 0 (recuerda que la prioridad más alta es la 0) llamando a la función `NVIC_SetPriority`:

```
NVIC_SetPriority(TIMER0_IRQn, 1);
```

c) Habilita la interrupción del timer 0 mediante la función `NVIC_EnableIRQ`:

```
NVIC_EnableIRQ(TIMER0_IRQn);
```

d) La línea anterior hará que la interrupción específica del timer 0 quede habilitada en el NVIC pero recuerda que la CPU posee un registro llamado PRIMASK en el que se encuentra el bit de enmascaramiento general de interrupciones. Tras el reset, el estado de dicho bit es uno y esto enmascara (deshabilita) todas las interrucpiones. A menos que pongameos a cero este bit, todas las interrupciones permanecerán dehabilitadas independientemente de que nos hayamos preocupado de habilitarlas en el NVIC. Para poner a cero el bit de enmascaramiento de interrupciones del regisro PRIMASK lo más sencillo es usar la función `__enable_irq`:

```
__enable_irq();
```

5. Imprime en pantalla la indicación inicial del cronómetro, es decir, `00:00:00`.
6. Ahora el programa entrará en el bucle principal. Queremos verificar que podemos llevar a cabo alguna tarea en el bucle principal del programa mientras el cronómetro se maneja enteramente dentro de la interrupción del timer 0. Como lo importante es constatar este hecho, haremos algo sencillo.

   Dentro del bucle principal, enciende el LED1 (según la denominación de los LEDs en la práctica 4) cuando se pulse el pulsador central del joystick. Mientras no se esté accionando el pulsador central del joystick, el LED1 deberá estar apagado.

Ahora escribe la función manejadora de la interrupción del timer 0. Recuerda que esta función es la que va a ejecutarse automáticamente cada vez que el timer 0 provoque una interrupción (es decir, cada segundo en este ejemplo). El prototipo de esta función es

```
void TIMER0_IRQHandler(void);
```

La razón de que el nombre del manejador de interrupción sea precisamente el indicado es que así se encuentra definido en el fichero `startup_LPC407x_8x_177x_8x.s`. Este es uno de los ficheros de inicialización del microcontrolador que añadimos al crear cada proyecto μVision para el LPC4088. Este fichero define la tabla de vectores de excepción/interrupción y fija un nombre concreto para cada una de las funciones manejadoras de interrupción.

En la función manejadora de interrupción:

1. Declara variables locales **estáticas** llamadas `horas`, `minutos` y `segundos` inicializadas con el valor 0. Recuerda que una variable local estática toma el valor con el que aparece inicializada (o 0 si no se inicializa expresamente) sólo la primera vez que

la ejecución entra en la función, mientras que en las siguientes ocasiones en que la función se ejecuta la variable "recuerda" el valor que tenía la última vez que se salió de la función.

2. Borra el flag de petición de interrupción de match 0 en el registro IR del timer 0.
3. Actualiza las variable `segundos`, `minutos` y `horas` igual que hiciste en la práctica anterior.
4. Imprime el cronómetro en pantalla.
5. Retorna de la función manejadora de interrupción. No es necesario poner expresamente una sentencia `return` sino simplemente dejar que la función alcance su llave de cierre.

Descarga el programa en el microcontrolador y comprueba que la pulsación central del joystick actúa sobre el LED1 mientras "simultáneamente" va cambiando la indicación de tiempo en el cronómetro.

4.2. Ejercicio 2

Añade al programa del ejercicio 1 la posibilidad de parar y reanudar el conteo de tiempo en respuesta a interrupciones generadas pulsando el joystick hacia abajo y hacia arriba. Recuerda que la dirección arriba del joystick está conectada al pin P2[25] y que la dirección abajo del joystick está conectada al pin P2[27]. Cuando se pulsan, los contactos del joystick ponen a nivel bajo el correspondiente pin.

En la función `main`, añade los siguientes pasos antes de entrar en el bucle principal:

1. Asegúrate de que los flags de petición de interrupción de los pines P2[25] y P2[27] parten en estado 0. Para ello, usa el registro `LPC_GPIOINT->CLR2` (o `LPC_GPIOINT->IO2IntClr` según su denominación en `LPC407x_8x_177x_8x.h`).
2. Programa el registro `LPC_GPIOINT->ENF2` (o `LPC_GPIOINT->IO2IntEnF`) para permitir que los pines P2[25] y P2[27] generen interrupciones por flanco de bajada.
3. Programa el NVIC para permitir interrupciones a través de los pines de los puertos 0 y 2. Asigna la prioridad 0 (la más alta) a las interrupciones procedentes de los puertos 0 y 2. Las funciones del NVIC a usar son las misma que se usaron para preparar la interrupción del timer 0 en el ejercicio 1 pero el símbolo para designar el número de interrupción de los puertos 0 y 2 es ahora `GPIO_IRQn` (no `TIMER0_IRQn` como antes).

Escribe la función manejadora de interrupción para las interrupciones de los puertos 0 y 2. El prototipo del manejador debe ser

```
void GPIO_IRQHandler(void);
```

que es el nombre con el que se define en `startup_LPC407x_8x_177x_8x.s`.

En el manejador de interrupción, comprueba cuál de las dos direcciones ha disparado la interrupción consultando el registro `LPC_GPIOINT->STATF2` (o `LPC_GPIOINT->IO2IntStatF`).

- Si la interrupción ha sido causada por la pulsación hacia abajo del joystick el timer 0 debe pararse poniendo a cero el bit `CEN` (bit 0) de su registro `TCR`

  El bit de petición de interrupción causante de la interrupción deberá ponerse a cero escribiendo en el registro `LPC_GPIOINT->CLR2` (o `LPC_GPIOINT->IO2IntClr`).

- Si la interrupción ha sido causada por la pulsación hacia arriba del joystick el timer 0 debe ponerse en marcha poniendo a uno el bit `CEN` (bit 0) del registro `TCR`.

  El bit de petición de interrupción causante de la interrupción deberá ponerse a cero escribiendo en el registro `LPC_GPIOINT->CLR2` (o `LPC_GPIOINT->IO2IntClr`).

Después de realizar las modificaciones, descarga el programa en el microcontrolador y comprueba que funciona correctamente. Debe ser posible parar y reanudar el cronómetro pulsando el joystick abajo y arriba y el LED1 debe encenderse al accionar su pulsador central.