Práctica 2

Estructura de Computadores

Modos de ejecución, gestión de excepciones y entrada/salida mediante interrupciones

Índice general

2.1. Objetivos de la práctica

En esta práctica completaremos nuestro estudio del procesador ARM7TDMI analizando sus modos de ejecución, sus excepciones y su sistema de entrada/salida, que gestionaremos mediante interrupciones. Los principales objetivos de la práctica son:

- Conocer los modos de ejecución del procesador.
- Entender, conocer y saber manejar el sistema de E/S por interrupciones.
- Conocer y saber manejar dispositivos básicos como leds, pulsadores, displays, teclado y temporizadores.

En esta práctica abordaremos el estudio del sistema de E/S de forma progresiva. Primero analizaremos el sistema de interrupciones nativo del ARM7TDMI y sus limitaciones, incluyendo para ello un dispositivo muy importante como son los temporizadores. Finalmente, estudiaremos el sistema de E/S con interrupciones vectorizadas que nos proporciona el controlador de interrupciones del chip S3C44B0X.

2.2. Lecturas previas obligatorias

Es imprescindible leer los capítulos 1 a 8 del documento titulado Sistema de memoria y de entrada/salida en la placa S3CEV40 ([TPGb]) publicado a través del Campus Virtual. Sin leer y comprender completamente ese documento, es absolutamente imposible hacer esta práctica. Puedes consultar toda la documentación en [arm] y [?].

2.3. Mapa de memoria de un programa de prácticas

Los detalles del sistema de memoria del S3CEV40 se detallan en el capítulo 2 del documento [TPGb] (lectura obligatoria).

De aquí en adelante vamos a utilizar un mapa de memoria muy similar para todos los programas, que está ilustrado en la figura 2.1. Como vemos tendremos varias regiones diferentes en el mapa de memoria de un programa, que son:

- Región de código
- Región de datos
- Región de heap (nosotros no usaremos, pero se colocaría ahí)
- Región de pilas
- Tabla de ISRs o RTIs

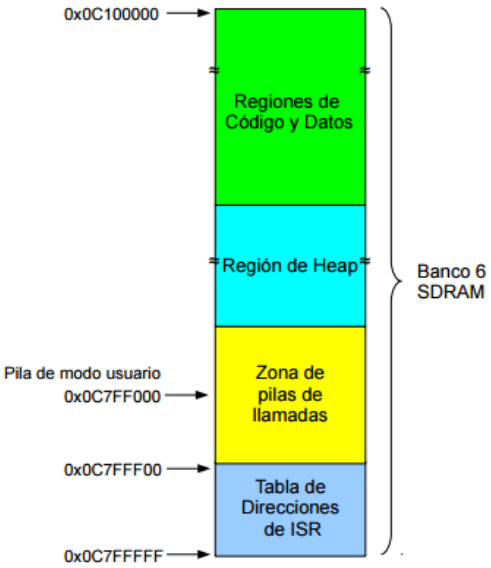

Figura 2.1: Mapa de memoria utilizado para los programas de las prácticas.

Como vemos, son direcciones que corresponden todas al banco 6 del controlador de memoria, en el que está ubicado el único chip SDRAM de la placa S3CEV40.

La tabla de ISRS tendrá una entrada por vector de interrupción, que se utilizará para almacenar la dirección de la rutina que debe tratar dicha interrupción (o excepción interna).

La región de pilas reserva espacio para ubicar las pilas de los distintos modos de ejecución del ARM7TDMI (generalmente inicializadas por el programa de test de la placa residente en la flash).

La región de heap, de existir, es gestionada por la biblioteca del sistema (en este caso newlib) para asignar dinámicamente trozos de esta región por medio de llamadas a *malloc*. Nosotros no utilizaremos memoria dinámica en este laboratorio.

El espacio restante será utilizado para ubicar las distintas secciones de nuestro programa (**text**, **data**, **rodata** y **bss**) comenzando en la dirección más baja del banco 6.

2.4. E/S por interrupciones no vectorizadas

En esta primera parte vamos a manejar dispositivos de E/S sencillos, mediante interrupciones no vectorizadas por la línea IRQ del procesador. Concretamente vamos a trabajar con leds y pulsadores, gestionados a través de pines multifunción de los puertos B y G del GPIO (consultar el capítulo 5 de [TPGb]); y el display de 8 segmentos, ubicado en el banco 1 (consultar el capítulo 6 de [TPGb]).

Haremos un sencillo programa que permitirá encender y apagar dos leds a través de dos pulsadores, y que haga que un segmento del display de 8 segmentos gire (como bordeando el símbolo 0), en una dirección o la otra. El movimiento del led alrededor del display de 8 segmentos estará gobernado por una interrupción periódica (de periodo 2s) generada por el TIMER0.

- **Encendido/apagado de los leds:** Cada uno de los botones tendrá asociado un led que cambiará su estado (apagado/encendido) a cada pulsación del botón.
- **Movimiento del Display:** Uno de los botones (botón1) se pulsa para parar o activar el movimiento del segmento y el otro (botón2) hace que cambie la dirección de giro.

Los ficheros con los que trabajaremos en esta primera parte de la práctica son:

- **44b.h:** fichero de cabecera con definiciones de macros para facilitar el acceso a los controladores de los dispositivos de nuestro sistema.
- **gpio.h** y **gpio.c**: interfaz e implementación de funciones para el manejo de los puertos multifunción. El alumno deberá implementar todas las funciones del fichero **gpio.c**, siguiendo las indicaciones de los comentarios y consultando la documentación [TPGb, um-].
- **button.h**: interfaz para el manejo de los pulsadores.
- **D8Led.h** y **D8Led.c**: interfaz e implementación de funciones para el manejo del display de 8 segmentos. El alumno deberá completar la implementación de las funciones **D8Led_segment** y **D8Led_digit**.
- **leds.h** y **leds.c**: interfaz e implementación de funciones para el manejo de los leds. El alumno deberá completar la implementación de las funciones **leds_init** y **leds_display**, usando el interfaz del puerto B definido en **gpio.h**.
- **intcontroller.h** e **intcontroller.c**: interfaz e implementación de funciones del módulo que maneja el controlador de interrupciones. El alumno deberá completar la implementación de todas las funciones de este módulo, siguiendo las indicaciones de los comentarios y la documentación de [TPGb, um-].
- **timer.h** y **timer.c**: interfaz e implementación de funciones del módulo que maneja los temporizadores PWM. El alumno deberá completar la implementación de todas las funciones de este módulo, siguiendo las indicaciones de los comentarios y la documentación de [TPGb, um-].
- **utils.h** y **utils.c**: interfaz e implementación de funciones auxiliares. En este caso sólo tenemos implementada una función para realizar una espera activa **Delay**. Esta función debe ser invocada en la fase de configuración (*setup*) con el argumento 0 para que se auto ajuste. Tras esta calibración el argumento a la

función estará en unidades de 0.1 ms. Así, por ejemplo, **Delay** (1000) esperaría 100ms. Este fichero está completo y no necesita ser modificado.

- **init.asm**: fichero de inicialización. Contiene el símbolo *start*, es por donde comenzará la ejecución de nuestro programa. Su misión es realizar una configuración básica mínima y luego invocar la rutina main de nuestro programa. Entre otra cosas: configura las pilas de los distintos modos de ejecución, habilita las interrupciones e inicializa la región de datos sin valor inicial a 0. **Tendremos que completar la rutina irq_isr en ensamblador.**
    - Esta rutina será declarada como rutina de tratamiento de interrupciones IRQ. Esta rutina hará una encuesta accediendo al registro **ISPR** del controlador de interrupciones:
        - ✓ Si es una interrupción por la línea del timer invocará la función **timer_ISR** (localizada en main.c) para tratarla y luego borrará el flag utilizando el interfaz definido en **intcontroller.h**.
        - ✓ Si es una interrupción por la línea **EINT4567** invocará la función **button_ISR** (localizada en main.c) para tratarla y después borrará el flag de interrupción utilizando el interfaz definido en **intcontroller.h**.
- **ld_script.ld**: script de enlazado que deberemos utilizar.
- **main.c**: fichero con el código del programa principal. Este fichero deberá ser codificado en su mayoría por el alumno, siguiendo las instrucciones que se darán a continuación.

Para la codificación del programa principal seguiremos la estructura de un sketch de arduino, que tiene las siguientes funciones:

- **setup**: cuya misión es configurar el sistema para su correcto funcionamiento. Su misión principal será configurar los controladores HW de los dispositivos que vamos a manejar, que en nuestro caso será:
    - Leds: debemos configurar el puerto B para que los pines 9 y 10 sean pines de salida. Esto lo haremos en la función **leds_init** por lo que setup sólo tendrá que invocar dicha función.
    - Pulsadores: debemos configurar el puerto G para que los pines 6 y 7 activen las señales EINT6 y EINT7 respectivamente, utilizando el interfaz del puerto G definido en **gpio.h**.
    - Display 8 segmentos: no necesita configuración inicial. Si queremos, podemos encender el led en la posición inicial deseada.
    - TIMER0: debemos configurarlo para que genere interrupciones periódicas, con un periodo de 2 segundos, utilizando el interfaz definido en **timer.h**.
    - Rutina **Delay**: debe ser invocada con el valor 0 para su calibración.
    - Debemos configurar el controlador de interrupciones para que active la línea IRQ en modo no vectorizado y deje la línea FIQ deshabilitada,

configure las líneas **TIMER0** y **EINT4567** por la línea IRQ del procesador y las deje habilitadas.

- Función **timer_ISR**: a pesar del nombre, esta función no es propiamente una rutina de tratamiento de interrupción, ya que es invocada desde **irq_ISR**. Debe declararse como una rutina corriente. Es la encargada de mover el led en el display de 8 segmentos una posición. La dirección del movimiento será la que esté almacenada en la variable **RL**.
- Función **button_ISR**: a pesar del nombre, esta función no es propiamente una rutina de tratamiento de interrupción, ya que es invocada desde **irq_ISR**. Debe declararse como una rutina corriente. Es la encargada de saber qué botón se ha pulsado consultando el registro **EXTINTPND** y realizar las tareas asociadas a la pulsación de cada uno de los botones (encender o apagar un led, cambiar la dirección de giro del *led rotante* y parar o arrancar el timer).

La función main del programa será por tanto:

```
int main(void)
{
	setup();
	while (1) {

	}
	return 0;
}
```

Para facilitar la codificación de nuestro programa, el fichero main.c tiene declarada una variable global que es una estructura que representa el estado del *led rotante* sobre el display de 8 segmentos, cuyos campos son:

- **moving**: usada como variable booleana, a 0 si el led está quieto y a 1 si está rotando.
- **speed**: indica la velocidad actual del led en número de vueltas del bucle principal (aproximamos a que una iteración del bucle principal son 200ms, asumiendo que el tiempo de espera domina sobre el tiempo de cómputo). En nuestro código **no se modificará**.
- **direction**: variable que indica la dirección de giro del led rotante. Un valor 0 indica sentido antihorario y un valor 1 indica sentido horario.
- **position**: indica el segmento que debe estar encendido actualmente (codificado como una posición en el **array Segments** del módulo **D8Led**).

Como ayuda final, veremos qué valores debemos dar a los registros de configuración del *TIMER0* para que produzca interrupciones periódicas de periodo M segundos (en nuestro caso M=2). De acuerdo con la documentación del chip, la frecuencia con la que trabajan los timers depende de tres factores: la frecuencia del sistema (MCLK), el factor de pre-escalado (P) y el factor de división (D). Conocidos estos valores la frecuencia de funcionamiento sería:

$$F = \frac{MCLK}{(P+1) \cdot D}$$

(2.1)

Queremos que se produzcan interrupciones cada M segundos contando N ciclos de frecuencia F, con 1 <= N <= 65535 (ya que el contador es de 16 bits), es decir:

$$1/F \cdot N = M \qquad (2.2)$$

$$\frac{(P+1) \cdot D}{MCLK} \cdot N = M$$

(2.3)

$$P = \frac{M \cdot MCLK}{N \cdot D} - 1$$

(2.4)

Nos interesa que M · MCLK sea divisible entre N · D, con P <= 255, ya que tenemos sólo 8 bits para el factor de pre-escalado. Por lo tanto, tomaríamos N como el mayor divisor de M · MCLK/D, representable con 16 bits.

En la placa MCLK = $64 \cdot 10^6 = 2^{12} \cdot 5^6$. Tomando D = 8, tendríamos que N debe ser el mayor divisor de $M \cdot 2^9 \cdot 5^6$. Para M = 2 quedaría que N debe ser el mayor divisor de $2^{10} \cdot 5^6$, es decir: $N = 5^6 \cdot 2^2 = 62500$. Y entonces nos quedaría:

$$P = \frac{2 \cdot 64000000}{62500 \cdot 8} - 1 = 255$$

Por lo tanto, para un periodo de 2 segundos exactos podemos tomar el factor de división 8, con pre-escalado 255 e inicializar la cuenta con 62500. El valor del registro de comparación puede ser cualquier valor mayor que 0 y menor que el número de cuenta, en este caso, 62500.

2.5. E/S por interrupciones vectorizadas

En el apartado anterior, con interrupciones no vectorizadas, hemos tenido que añadir un proceso de encuesta, en este caso facilitado por el controlador de interrupciones (consultando sólo uno de sus registros podíamos saber qué dispositivo necesitaba ser atendido).

En este apartado final vamos a modificar el código anterior para que trabaje con interrupciones vectorizadas por la línea IRQ, y vamos a añadir un nuevo dispositivo, el teclado matricial, que también trataremos por interrupciones. Para comprender el funcionamiento del teclado debe leerse la sección 8 de [TPGb]. El funcionamiento será el mismo que en el apartado anterior con la diferencia de que cuando se pulse una tecla del teclado matricial, primero se verá dicha tecla en el display y después volverá a

aparecer el led rotante. La velocidad de movimiento del led rotante variará en función de la tecla pulsada como se explica al final de esta sección.

Para comenzar, tendremos que adaptar el código de la parte anterior para que las interrupciones por la línea IRQ se traten en modo vectorizado. Para ello tenemos que:

- Copiar los archivos gpio.c, leds.c, D8Led.c, intcontroller.c y timer.c codificados en el apartado anterior.
- Cambiar la configuración del controlador de interrupciones para que la línea IRQ esté en modo vectorizado.
- Convertir las rutinas **button_ISR** y **timer_ISR** en rutinas de tratamiento de interrupción reales y registrarlas como RTIs para el tratamiento de las señales **EINT4567** y **TIMER0** respectivamente. Para esta conversión es necesario hacer dos cosas:
    - Registrar las ISRs, añadir la dirección de la RTI a la tabla de direcciones de ISRs
    - Añadir al final de las funciones el borrado del flag de interrupción

Realizando los cambios anteriores tendremos que comprobar que funciona como en el apartado anterior, pero esta vez utilizando interrupciones vectorizadas, sin necesidad de hacer la encuesta en la RTI.

A continuación debemos añadir el soporte para utilizar el teclado de la siguiente forma:

- Crearemos una función **keyboard_ISR** para el tratamiento de la interrupción por pulsación de tecla en el teclado matricial. Deberá declararse como RTI y registrarse adecuadamente para el tratamiento de la línea **EINT1**.
- En la función **setup** añadiremos a la configuración del controlador de interrupciones la configuración de la línea **EINT1** por IRQ y la habilitación de esta línea.

Al alumno se le proporcionarán:

- Dos nuevos ficheros **keyboard.h** y **keyboard.c** que definen e implementan el interfaz del módulo de teclado. El alumno deberá completar la función de escaneo definida en el fichero **keyboard.c**.
- El esqueleto de la función **keyboard_ISR** es:
    - Esperar 20ms para eliminar los rebotes de presión
    - Escanear el teclado utilizando el interfaz definido en el fichero **keyboard.h**.
    - Cambiar la configuración del timer 0 para que la generación de interrupciones periódicas tenga un periodo distinto en función de la tecla pulsada:
        - ✓ Tecla 0: periodo de 2s, valor de cuenta 62500 y divisor 1/8
        - ✓ Tecla 1: periodo de 1s, valor de cuenta 31250 y divisor 1/8
        - ✓ Tecla 2: periodo de 0.5s, valor de cuenta 15625 y divisor 1/8

- ✓ Tecla 3: periodo de 0.25s, valor de cuenta 15625 y divisor ¼
- ✓ Resto de teclas: no cambiaremos la configuración del timer

A completar por el alumno.

Esperar a que se deje de presionar la tecla leyendo el bit 1 del registro de datos del puerto G. **A completar por el alumno.**

- Esperar 20ms para eliminar rebotes de depresión.
- Borrar el flag de interrupción por la línea EINT1. **A completar por el alumno.**

Bibliografía

[arm] Arm architecture reference manual. Accesible en http://www.arm.com/miscPDFs/14128.pdf. Hay una copia en el campus virtual.

[TPGa] Christian Tenllado, Luis Piñuel, and José Ignacio Gómez. Introducción al entorno de desarrollo eclipse-arm.

[TPGb] Christian Tenllado, Luis Piñuel, and José Ignacio Gómez. Sistema de memoria y de entrada/salida en la placa s3cev40.

[um-] S3c44b0x risc microprocessor product overview. Accesible en http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id =229&partnum=S3C44B0. Hay una copia en el campus virtual.