

# Práctica 2

## Estructura de Computadores

Entrada/salida mediante espera de respuesta

# Índice general

2.1. Objetivos de la práctica . . . . .	2
2.2. Lecturas previas obligatorias . . . . .	2
2.3. Mapa de memoria de un programa de prácticas . . . . .	3
2.4. E/S programada con espera de respuesta . . . . .	4
Bibliografía . . . . .	7

## 2.1. Objetivos de la práctica

En esta práctica comenzaremos nuestro estudio del sistema de entrada/salida del procesador ARM7TDMI, gestionando algunos dispositivos básicos mediante espera de respuesta. Los principales objetivos de la práctica son:

- Entender, conocer y saber manejar el sistema de E/S mediante espera de respuesta.
- Conocer y saber manejar dispositivos básicos como leds, pulsadores, displays.

En las tres primeras prácticas abordaremos el estudio del sistema de E/S de forma progresiva. En esta práctica aprenderemos a realizar una E/S programada por espera de respuesta utilizando los dispositivos más sencillos. En las siguientes 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 **1,2,3, 5 y 6** 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 [um-].

## 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, que se extiende hasta la región de código
- Tabla de ISRs o RTIs

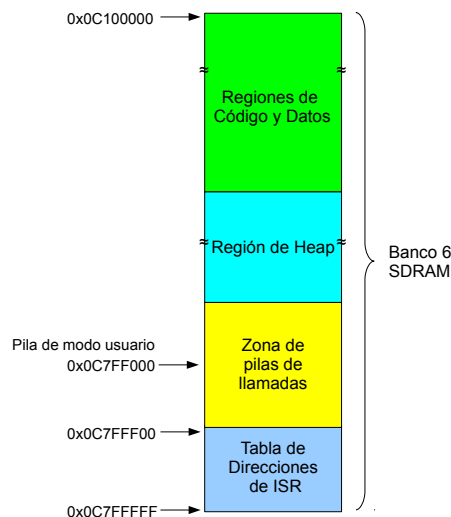


Figura 2.1: Mapa de memoria utilizado para los programas de las prácticas.

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 programada con espera de respuesta

En esta práctica vamos a manejar dispositivos de E/S sencillos, realizando espera activa para atenderlos. 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 haga que un segmento del display de 8 segmentos gire (como bordeando el símbolo 0), en una dirección o la otra.

Para comenzar haremos un proyecto Eclipse para compilación cruzada tal y como se indica en el documento [TPGa]. En el proyecto añadiremos los ficheros suministrados con la práctica, que 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** y **button.c**: interfaz e implementación de funciones para el manejo de los pulsadores. La función **read\_button** **deberá ser completada por el alumno**, utilizando el interfaz del puerto G definido en **gpio.h**.
- **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**.
- **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.
- **init.S** fichero de inicialización. Contiene el símbolo *start*, que es 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. Este fichero está completo y no necesita ser modificado.
- **ld\_script.ld**: script de enlazado que deberemos utilizar.
- **main.c**: fichero con el código del programa principal. **Este fichero deberá ser modificado 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 básicamente tiene dos funciones principales:

- **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 sean pines de entrada. Lo haremos utilizando el interfaz para el puerto G definido en `gpio.h`.
  - Display 8 segmentos: no necesita configuración inicial, más que si queremos, encender el led en la posición inicial deseada.
  - Rutina `Delay`: debe ser invocada con el valor 0 para su calibración.
- **loop**: representa el cuerpo de un bucle infinito. Es la **función principal de nuestro programa, que tendrá que codificar el alumno**. Al final de la rutina se hará una espera activa de 200ms (con esto eliminaremos los rebotes de los pulsadores y tendremos una base para determinar la velocidad de giro del led). Antes de esta espera la función deberá:
  1. Comprobar si se ha pulsado alguno de los pulsadores invocando la función `read_button`. Esta función devolverá un entero en el que los dos bits menos significativos indican si se ha pulsado o no alguno de los leds. El módulo `button` exporta además las macros `BUT1` y `BUT2` que pueden usarse para comprobar si se ha pulsado el botón 1 o el botón 2 (haciendo una and con el valor devuelto por `read_button`).
  2. Si se ha pulsado el pulsador 1, debe apagar ambos leds y cambiará la dirección de giro del segmento en el display de 8 segmentos.
  3. Si se ha pulsado el pulsador 2, se incrementará un contador de pulsaciones de ese botón. Si el contador es par debe permutar el estado del led 1 y si es impar, el del led 2. Además, permutará el estado de movimiento (en marcha o parado) del segmento en el display.
  4. Si el segmento no está detenido, decrementará un contador de las iteraciones que lleva en marcha y si se hace 0 deberá desplazar el segmento una posición y reiniciar dicho contador.

La función `main` del programa será por tanto:

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

    while (1) {
        loop();
    }
    return 0;
}
```

Para facilitar la codificación de nuestro programa, en 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).
- **iter**: contador de iteraciones del bucle principal desde el último movimiento del led. Se va decrementando en cada iteración del bucle si está en movimiento el led. Cuando llega a 0, el led se desplazará y este contador debe ser entonces inicializado al valor del campo **speed**.
- **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 modulo **D8Led**).

# 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](http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id=229&partnum=S3C44B0). Hay una copia en el campus virtual.