Práctica 5:
Introducción al sistema de Entrada/Salida

5.1 Objetivos

El objetivo de esta práctica es tener un primer contacto con los elementos fundamentales de un sistema de entrada/salida. Para ello utilizaremos la placa S3CEV40 descrita en clase, y únicamente haremos operaciones de E/S mediante encuesta. Como ejemplo práctico usaremos los dispositivos más sencillos de los que disponemos: LEDs, pulsadores y el display de ocho segmentos. Los principales objetivos de la práctica son:

- Conocer el sistema de E/S de la placa S3CEV40 usada en el laboratorio
- Comprender el mecanismo de E/S localizado en memoria
- Aprender a programar dispositivos básicos mediante encuesta.

5.2 Acceso a los dispositivos de la placa

Ya se ha realizado una presentación detallada de los dispositivos presentes en la placa S3CEV40 y del funcionamiento de los tres que utilizaremos en esta práctica. A continuación se incluye, a modo de recuerdo, la información más relevante de cada dispositivo utilizado:

- LEDs.
  - Registro de control de 11 bits (PCONB) en dirección 0x01d20008
  - Registro de datos de 11 bits (PDATB) en dirección 0x01D2000C
  - Configuración: pines 9 y 10 de PCONB como salida escribiendo 0.
  - Encendido/apagado: escribir 0 para encender (1 para apagar) en el bit 9 (led1) y/o 10 (led2) de PDATB
- Pulsadores (botones)
  - Registro de control de 16 bits (PCONG) en dirección 0x01D20040
  - Registro de control de 8 bits PUPG en dirección 0x01D20048
  - Registro de datos de 8 bits (PDATG) en dirección 0x01D20044
  - Configuración. pines 6 y 7 como entrada (escribir 0's en PCONG) y poner a 0's PUPG
  - Detección de pulsación: leer de PDATG y comprobar si se está pulsando el botón. Si el bit 6 es 0, el botón 1 está pulsado (resp. bit 7 y botón 2)
- Display de 8 segmentos.
  - Registro de control: no existe (siempre configurados como salida)
  - Registro de datos: escribiremos en la dirección 0x2140000 (LED8ADDR).
  - Encendido/apagado de cada segmento: cada segmento tiene asociado un bit del registro de datos. Si un bit está a 0, el segmento correspondiente está encendido. Consulta las transparencias para ver la correspondencia *segmento-bit*.

Acerca del display de 8 segmentos

El registro de datos del display 8-segmentos es de 8 bits (como el de los pulsadores) pero su implementación hace que haya que tener especial cuidado en las escrituras a este dispositivo. Hasta esta práctica, todas las escrituras realizadas han sido de tamaño *palabra (str).* Sin embargo, por las peculiaridades del controlador de este dispositivo, una escritura de tamaño palabra no tendrá el comportamiento esperado (en concreto, dejará en el registro de datos los 8 bits de mayor peso de nuestra palabra, no los de menor peso como cabría esperar). Para evitar este comportamiento basta con indicar que queremos realizar una escritura de **tamaño byte**, usando el modificador de tamaño *b:* ***strb.*** El código proporcionado junto con el guión ya realiza la escritura correctamente. Simplemente hay que asegurarse de imitar ese comportamiento en caso de desarrollos propios.

5.3 Operaciones a nivel de bit: uso de máscaras

A la hora de escribir/leer los registros de control/datos de un dispositivo, a menudo querremos consultar el valor de un bit concreto. Sin embargo, como ya hemos visto en numerosas ocasiones, los registros del ARM son de 32 bits. ¿Cómo podemos consultar el valor de un bit cualquiera de entre esos 32? ¿Cómo podemos ponerlo a 0/1 sin modificar el resto? Para conseguirlo, usaremos las instrucciones AND y OR a nivel de bit que proporciona el repertorio ARM (*and* y *orr*) y un segundo operando constante (inmediato) al que denominaremos *máscara.*

Por ejemplo, si queremos consultar el valor del bit 6 (empezando a numerar en 0) del registro *r1*, podríamos hacer lo siguiente:

```
and r2,r1,#0x00000040
```

Con esa operación, forzamos a ‘0’ todos los bits de *r2* salvo el bit 6. Si el contenido de *r2* es 0, el valor del bit 6 de *r1* era 0. Si es diferente de 0, había un 1.

Si queremos escribir un ‘0’ únicamente en el bit 3 del registro *r3*, podríamos hacer lo siguiente:

```
and r3,r3,#0xFFFFFFF7
```

De ese modo, dejamos intactos los bits restantes de *r3*, pero forzamos un ‘0’ en el bit 3 (puesto que 0x7, expresado en base 2 es ‘0111’: se realiza la *and* lógica del bit 3 con ‘0’).

En ocasiones, no es posible codificar el valor de la *máscara* como segundo operando de una instrucción ARM. En esos casos, podremos cargar el valor de la máscara en un registro para posteriormente hacer la *and/or* lógica entre registros. Para poder almacenar en un registro el valor de una máscara arbitraria, podemos usar la macro: `LDR <ri>,=CTE`. Para el ejemplo anterior nos quedaría:

```
ldr r4,=0xFFFFFFF7
and r3,r3,r4
```

Se deja al alumno como ejercicio pensar cómo podríamos escribir un ‘1’ en un bit cualquiera de un registro.

5.4 Estructura básica de un programa dedicado a Entrada/Salida

Determinadas aplicaciones están enfocadas únicamente a realizar tareas de E/S: leer valores de sensores (temperatura, luz, posición…), teclado u otros, y actuar sobre elementos externos al procesador (motores, displays, leds…) en función de la información obtenida en la lectura. En esos casos, la estructura del código de dichas aplicaciones, extremadamente simplificada y asumiendo una E/S basada en encuesta y espera activa, podría esquematizarse como sigue:

```
función principal {

	// Configurar dispositivos como entrada y/o salida
	inicialización_de_dispositivos;

	while (true) {

		// Rutina(s) para leer los valores de cada dispositivo
		// A su vez, contendrán bucles de espera para el dispositivo
		lectura_dispositivos_de_entrada;

		calcular_salidas_en_función_de_etradas;

		escritura_dispositivos_de_salida;
	}

}
```

Como se verá en el código proporcionado para el desarrollo de la práctica, nuestros ejemplos siguen fielmente ese modelo.

5.5 Configuración del proyecto

A diferencia de las prácticas anteriores, en esta ocasión deberemos volcar el código en la placa. Es decir, no vamos a realizar una simulación del comportamiento sino que el código escrito se ejecutará realmente en el procesador ARM7TDMI e interaccionará con los dispositivos de E/S presentes en la placa S3CEV40.

Para ello, deberemos cambiar ligeramente el proceso de configuración del proyecto, para indicar que debe usar el conector UnetICE para volcar el código en la memoria de la placa y permitir la depuración sobre la propia placa. El único cambio respecto a nuestros anteriores proyectos se ilustra en la Figura 1: en la pestaña ***Remote*** seleccionamos *Remote device* **UNetICE**; en *Speed* seleccionamos ***Medium Speed*** y en *Communication type* **USB**.

El resto de elementos de la configuración (*Processor*, *Assembler*…) son similares a las prácticas anteriores.

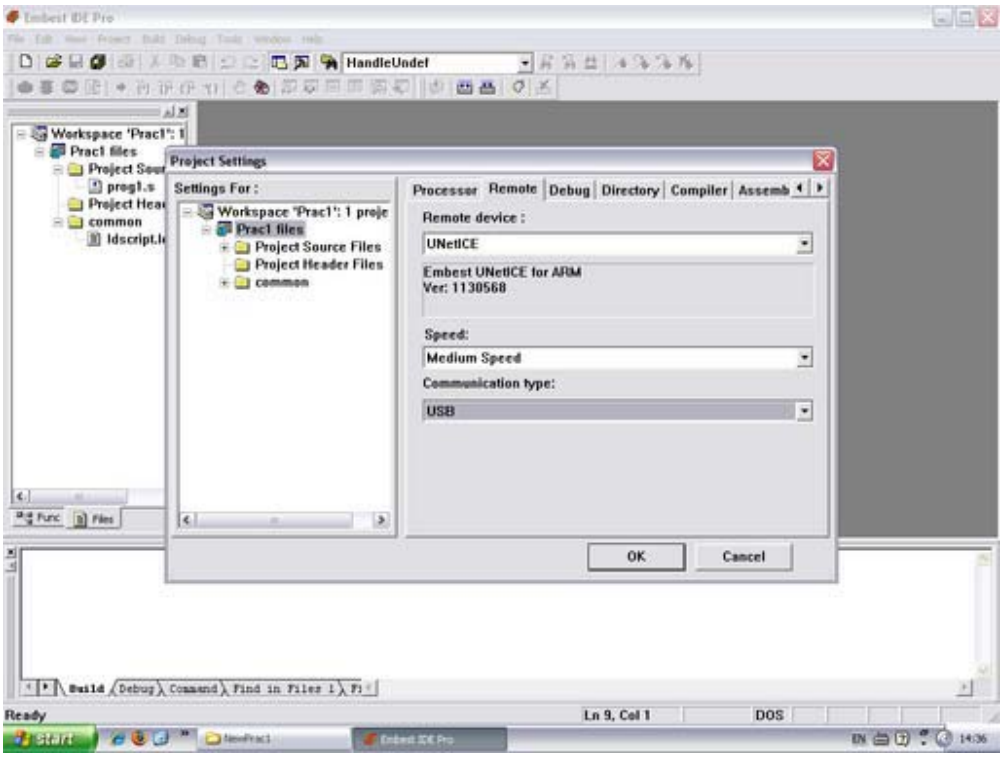

Figura 1. La pestaña Remote debe configurarse para acceder al dispositivo UnetICE

Una vez configurado el proyecto, procedemos del mismo modo que en prácticas anteriores: añadimos el/los ficheros con nuestro código ensamblador, construimos el proyecto (*Build*), nos conectamos a la placa (*Remote Connect*) y descargamos el código (*Download*). La única diferencia en el proceso es el tiempo de espera: la conexión y descarga ahora sí tiene lugar y, por tanto, tarda más. Sí es posible que la **primera vez que se realice la orden *Remote Connect*** el entorno se quede aparentemente *colgado* sin dejarnos proceder con la descarga del código (*Download).* En ese caso, **basta con pulsar el botón de *Stop*** del entorno de desarrollo *Embest IDE* para que la placa responda y se pueda descargar el código.

Conexión de la placa

Por supuesto, antes de poder realizar la conexión y envío del código a la placa, ésta debe conectarse al PC del laboratorio. La placa se conecta al PC a través de **dos conexiones USB** que deben conectarse a puertos USB del PC (preferiblemente a los de la zona posterior).

Al conectar la placa, debe encenderse la luz “PWR” del dispositivo *Embest UNetICE*, que indica que el dispositivo está siendo alimentado a través del puerto USB. Asimismo, la pantalla LCD de la placa (así como algunos LEDs) deben encenderse al conectarla al PC. En caso contrario, consulta con el profesor/técnico de laboratorio para solucionar el problema.

5.6 Desarrollo de la práctica

Junto con el guión, se han proporcionado tres ficheros ensamblador con tres programas que, de forma independiente, hacen uso de los dispositivos explicados (LEDs, display de segmentos y pulsadores). Como **trabajo previo (obligatorio)** deberéis realizar las siguientes tareas:

1) Junto con este guión, se proporcionan cuatro ficheros que incluyen ejemplos de uso de los diferentes dispositivos (en concreto, *uso_8seg.s* proporciona un sencillo ejemplo de uso del display de 8-segmentos; *uso_led.s* y *uso_boton.s* contienen respectivamente ejemplos para el uso de LEDs y de los pulsadores SB1 y SB2).

   El alumno **deberá comprender y ejecutar dichos códigos en la placa.** Asimismo, determinad qué valor hay que escribir en LED8ADDR para que en el display aparezcan los caracteres ,’1,’2’,3’….’9’,’0’.‘A’, ‘B’,’C’,’D’,’E’ y ‘F’.

2) El cuarto fichero (*boton_leds.s*) contiene un programa algo más elaborado que sigue la estructura general presentada en la sección 5.4. En concreto, el programa principal consta de un bucle infinito que realiza la detección de pulsación de los botones y que modifica el estado del LED 1 en función de las pulsaciones a SB1.

   En este apartado se pide **completar dicho programa para que el botón SB2 modifique el estado del LED2**. Más concretamente, deberá escribir una rutina ***switch_led2*** que cambie el estado del LED2 a cada invocación, y deberá **completar el cuerpo de la función principal** (*start*) para que considere el caso en que la rutina *detecta* devuelva un 2 o un 3.