PRÁCTICA 2: SISTEMA DE E/S Y DISPOSITIVOS BÁSICOS

En la práctica 1 hemos desarrollado código para un procesador ARM. En esta segunda práctica queremos utilizar ese código en un entorno real. Para ello, tendremos que aprender a trabajar con la placa de desarrollo que hay en el laboratorio (Embest S3CEV40). En esta placa el procesador está acompañado de muchos otros dispositivos, principalmente de entrada/salida.

Vamos a aprender a interaccionar con ellos trabajando en C, pero siendo capaces de bajar a ensamblador cuando sea necesario.

OBJETIVOS

- Interactuar con una placa real y ser capaces de ejecutar en ella el código desarrollado en la práctica anterior.
- Profundizar en la interacción C / Ensamblador.
- Ser capaces de depurar el código ensamblador que genera un compilador a partir de un lenguaje en alto nivel.
- Ser capaces de depurar un código con varias fuentes de interrupción activas.
- Gestionar la entrada/salida con dispositivos básicos, asignando valores a los registros internos de la placa desde un programa en C (utilizando las librerías de la placa).
- Aprender a desarrollar en C las rutinas de tratamiento de interrupción.
- Aprender a utilizar los temporizadores internos de la placa y el teclado.

ENTORNO DE TRABAJO

La forma de trabajar es casi idéntica a la utilizada en la práctica anterior, pero en esta práctica en lugar de simular la ejecución de nuestro código lo **ejecutaremos** de verdad **en la placa.** Para ello, se incluye un soporte especial que permite conectarse a la placa de desarrollo real a través del puerto **JTAG**. .

En el apartado **Depuración sobre la placa ARM** del documento *GuiaEntorno.pdf* se detallan los pasos a seguir para conectarnos a las placas y depurar el código en ellas.

Las placas sólo pueden usarse en el **laboratorio 0.5b** durante el horario reglado de la asignatura. **IMPORTANTE:** esta práctica **no se puede hacer a distancia.**

MATERIAL DISPONIBLE

En Moodle de la asignatura en podéis encontrar el siguiente material de apoyo:

- Un proyecto que utiliza los pulsadores, leds, 8led y el *timer 0* de la placa. Antes de escribir una línea de código debéis entender a la perfección este proyecto.
- Cuadernos de prácticas escritos por compañeros de la Universidad Complutense:
  - ***EntradaSalida.pdf***: Describe los principales elementos de entrada salida que tiene la placa, incluyendo los que vamos a utilizar en esta práctica. **Hay que estudiarlo en detalle.**
  - ***P2-ec.pdf***: Describe la gestión de excepciones y el mapa de memoria que vamos a usar. **Hay que estudiarlo en detalle.**

- Documentación original de la placa (proporcionada por la empresa desarrolladora): muy útil para ver más detalles sobre la entrada / salida.

ESTRUCTURA DE LA PRÁCTICA

Paso 1: estudiar la documentación.

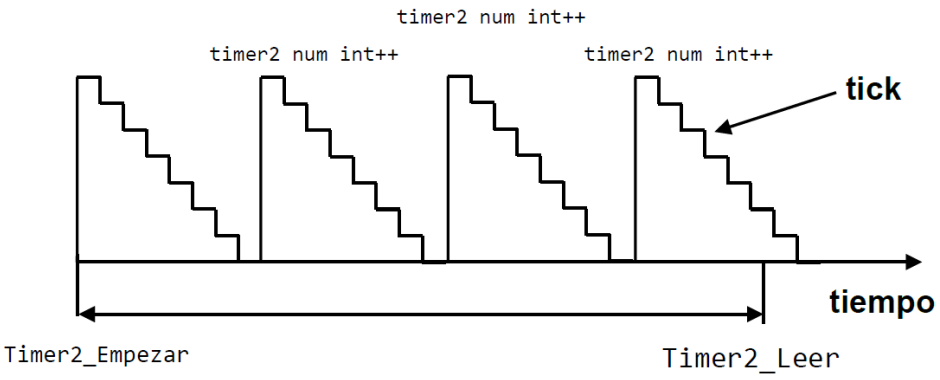
Paso 2: estudiar el proyecto que os proporcionamos, en el que se utilizan algunos de los dispositivos que se describen en este guión.

Debéis entender cómo se controlan estos dispositivos, y cómo se comunican con el procesador a través de las interrupciones.

Debéis ejecutar paso a paso el ensamblador que se genera a partir del código C de las rutinas de tratamiento de interrupción

Paso 3: Medidas de tiempo. Debéis desarrollar una sencilla biblioteca que permita utilizar el **timer 2** para medir tiempos. La biblioteca constará de las siguientes funciones:

- **`Timer2_Inicializar()`**: configura el **timer 2** para que trabaje a la **máxima precisión** posible**. El reloj de la placa** está configurado a **64MHz.** Para aumentar el rango del contador, el temporizador generará una **interrupción** cada vez que haga una **cuenta completa** (queremos que la cuenta sea lo mayor posible para no sobrecargar en exceso al sistema con interrupciones). La biblioteca dispondrá de la variable interna **`timer2_num_int`**, compartida entre la interrupción y el resto de funciones, que llevará la cuenta de los periodos completos del temporizador. Al acabar la cuenta completa el temporizador se reinicia al valor inicial y seguirá contando.
- **`Timer2_Empezar()`**: reinicia la cuenta de tiempo (contador y la variable), y comienza a medir.
- **`Timer2_Leer()`**: lee la cuenta actual del temporizador y el número de interrupciones generadas, y devuelve el tiempo transcurrido en microsegundos.



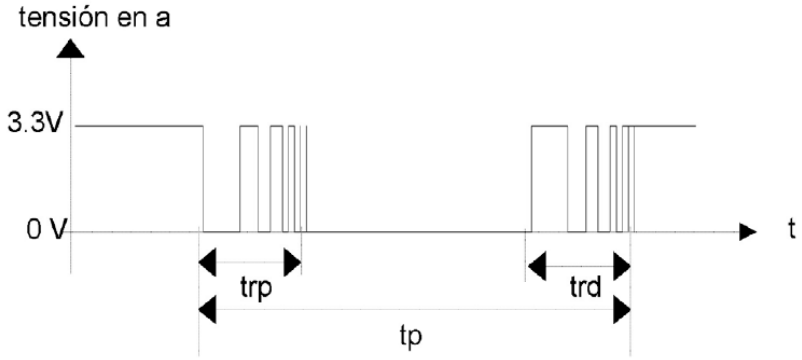
IMPORTANTE: comprobar que vuestra librería de medición está bien calibrada utilizando la función **`Delay()`** que podéis encontrar en **`44blib.c`**. Medid distintos retardos (1 ms, 10ms y 1 s) y aseguraos de que los resultados son parecidos. No tienen que ser idénticos, dado que **`Delay()`** es una aproximación, y por tanto es menos preciso que el método que vais a desarrollar.

Paso 4: Tratamiento de excepciones. Durante la ejecución del código se pueden dar casos en los que el procesador se encuentra que no sabe qué hacer. Por ejemplo al intentar ejecutar una instrucción inválida o un acceso a memoria no alineado, entonces se produce una excepción. Deberéis realizar una función de tratamiento de excepciones. Programaréis el vector de excepciones para que independientemente de la excepción que salte, se invoque a esa función. En la función se debe identificar el tipo y la instrucción causante. La excepción se tratará adecuadamente (por ejemplo sacando un código parpadeante de error por el 8leds y parando la ejecución).

Paso 5: Desarrollo de una pila de depuración. Debéis gestionar una pila en la que introduciréis datos que os resulten útiles para depurar eventos asíncronos como interrupciones. Debéis ubicar la pila al final del espacio de memoria, antes de las pilas de los distintos modos de usuario. Dejad una distancia suficiente para que las pilas no se solapen y gestionarla como **una lista circular controlando los límites**, de forma que guarde los últimos "n" elementos. Para introducir datos se invocará a la función ***`push_debug(int ID_evento, int auxData)`*** que debéis diseñar. Esta función escribirá en la pila tres enteros, por un lado los parámetros ID_***`evento,`*** que permita identificar qué interrupción ha saltado, y el parámetro ***`auxData,`*** con datos auxiliares aclaratorios, y por otro el momento exacto en que se ha invocado a la función. Para ello hará uso de la biblioteca de medidas de tiempo.

Paso 6: Eliminación de los rebotes en los pulsadores. Para poder jugar al sudoku, vamos a utilizar los pulsadores de la placa, dado que son dispositivos mecánicos reales al tocarlos producen una señal oscilante (como ilustra la figura). Usando la pila de depuración podemos observar cuántas veces se ha entrado en cada rutina de interrupción, y saber si hay o no rebotes, y cuando se producen. Tras hacer este análisis deberemos tomar las medidas necesarias para eliminarlos: hay que eliminar **tanto los que se producen al pulsar, como los que se producen al levantar**.

Para ello deberéis implementar una máquina de estados siguiendo el siguiente esquema:



1) Al detectar la pulsación se identifica el botón pulsado, se deshabilitan las interrupciones de los botones para ignorar los rebotes y se programa un retardo inicial con un temporizador (`trp`).
2) Tras el retardo, cada 10 milisegundos se monitoriza el botón con un temporizador para detectar cuándo levanta el dedo el usuario.
3) Cuando se levante el dedo, de nuevo se introduce un retardo para filtrar los posibles rebotes de salida (`trd`).

4) Tras el retardo final se habilita la IRQ de los botones.

IMPORTANTE: las rutinas de interrupción deberán ser **ligeras**. En ningún caso se permitirá hacer una espera activa dentro de una IRQ. Identificar los retardos `trp` y `trd` de la placa con la ayuda de la pila de depuración. Los retardos pueden variar según el estado de cada placa.

Paso 7: Debéis desarrollar y ejecutar en la placa el juego sudoku que habéis desarrollado en la práctica 1, siguiendo el mismo esquema que en el proyecto que os hemos proporcionado e incluyendo vuestros fuentes de la práctica anterior.

Importante: comprobar en todas las funciones que trabajan con el tipo de dato `char` si se está haciendo bien la gestión de los signos. En caso contrario solucionadlo garantizando la extensión de signo cuando sea necesaria tanto en C como en las versiones Thumb y ARM.

Una vez comprobado que vuestro código funciona bien en la placa y que vuestra librería mide bien los tiempos, **repetiréis la comparación entre versiones realizada en la práctica 1, pero esta vez midiendo de forma precisa los tiempos con vuestra librería**.

Paso 8: Al código de la práctica 1 hay que añadirle las siguientes funcionalidades de entrada/salida para poder jugar con los pulsadores:

- El código comenzará a ejecutarse cuando se pulse un botón. Se calcularan por primera vez los candidatos de todas las celdas del tablero.
- A continuación, se seleccionará en que casilla se quiere introducir un número. La fila y la columna se introducirán con el botón izquierdo (para elegir el número) y el derecho (para confirmar):
  - Al comenzar aparecerá una **F** en el 8led (7-segmentos).
  - Cuando el usuario pulse el botón izquierdo se visualizará un **`1` en el 8led**.
  - Si el usuario mantiene el botón pulsado más de medio segundo, el número se irá incrementando cada 300 milisegundos del **1** al **9**. Si llega al **9** se volverá al **1**.
  - Cuando levante el dedo el número que hay en el 8led se mantendrá fijo. Si vuelve a pulsar el botón izquierdo de nuevo se irá incrementado como antes.
  - Si el usuario pulsa el botón derecho confirmará el número actual. Entonces aparecerá una **C** en el 8led y se repetirá el proceso para elegir la columna.
- Cuando se confirmen la fila y la columna deberemos introducir el **Valor** comenzando con el **1**. Si se introduce un **0** implicara el borrado del valor de esa celda.
- Una vez introducido el nuevo valor se procederá a actualizar el Valor de la celda. Como queremos poder borrar el valor de las celdas, y que cuando se repite un número no siempre podremos saber cuál de las celdas con ese valor es la errónea, se procederá a recalcular los candidatos de todo el tablero. Se deben modificar las funciones **`sudoku_recalcular`** y **`sudoku_candidatos`** para comprobar si hay errores.

sudoku_candidatos debe comprobar, al evaluar los candidatos de una celda, si finalmente su valor, (sea Pista o valor introducido por el usuario), está o no en la lista de candidatos. Si no está deberemos marcar el bit correspondiente de esa celda como errónea. La función devolverá como resultado: 0 celda vacía, 1 celda llena, -1 celda llena pero error detectado.
sudoku_recalcular tiene que re-evaluar todas las celdas, incluidas las Pistas. Debe llevar la cuenta de celdas vacías y si hay o no hay errores. Devolverá el número de celdas vacías como hasta ahora, pero si hay errores el valor será negativo.

Tanto los botones, como el teclado, y el timer 0, se gestionarán utilizando interrupciones IRQ. Las rutinas de tratamiento de interrupción se desarrollarán en C siguiendo la misma estructura que en el proyecto que os damos. El compilador se ocupará de algunos de los detalles a bajo nivel, pero:

- Debéis ser capaces de entender el código ensamblador que genera el compilador y explicar qué hace en cada paso.

Al presentar este apartado, **debéis ser capaces de explicar cómo funciona su código y qué hace cada una de las instrucciones**.

APARTADO OPCIONAL 1

Permitir interrupciones anidadas

APARTADO OPCIONAL 2

Estudiar la estructura del *linker script* y el código de la función init. Detectar un problema que puede aparecer con alguna de las direcciones de los segmentos al declarar datos de tamaños inferiores al tamaño de palabra y plantear una solución.

EVALUACIÓN DE LA PRÁCTICA

La primera parte de la práctica (paso 6) se deberá mostrar al profesor de vuestro grupo al inicio de vuestra tercera sesión de esta práctica(es decir, en vuestra sesión de la semana del 9 al 14 de noviembre).

La segunda parte (paso 7) deberá entregarse aproximadamente el 23 de Noviembre.

Las fechas definitivas y turnos de corrección se publicarán en Moodle de la asignatura en.

ANEXO 1: REALIZACIÓN DE LA MEMORIA

La memoria de la práctica tiene diseño libre, se recomienda seguir las pautas de la guía proporcionada por la asignatura. Es obligatorio que incluya:

1. Resumen ejecutivo (una cara máximo).
2. Descripción de la librería desarrollada para medir tiempos y resultados obtenidos al medir las funciones desarrolladas en la práctica 1. Comparar con las estimaciones realizadas en la práctica 1, utilizando el número de instrucciones ejecutadas y el tiempo de simulación.
3. Breve descripción de la gestión de la entrada/salida en vuestro proyecto. Mostrar los diagramas de las máquinas de estados debidamente comentados.

4. Descripción de los problemas encontrados en la realización de la práctica y sus soluciones.
5. Código fuente comentado (sólo el que habéis desarrollado vosotros). Como siempre, cada función debe incluir una cabecera en la que se explique qué hace, qué parámetros recibe...

Se valorará que el texto sea **claro y conciso**. Cuanto más fácil sea entender el funcionamiento del código, mejor.

ANEXO 2: ENTREGA DE LA MEMORIA

La entrega de la memoria será a través de la página web de la asignatura (moodle en http://add.unizar.es). Debéis enviar un fichero comprimido en formato ZIP con los siguientes documentos:

1. Memoria en formato PDF
2. Código fuente desarrollado (en formato texto)

Se mandará un único fichero por pareja con el siguiente nombre::
p1_NIP-Apellidos_Alumno1_NIP-Apellidos_Alumno2.zip

Por ejemplo: p1_345456-Gracia_Esteban_45632-Arribas_Murillo.zip