PRÁCTICA 2: SISTEMA DE E/S Y DISPOSITIVOS BÁSICOS

En esta práctica vamos a comenzar a desarrollar el soporte necesario para jugar al Reversi directamente desde la placa. Para ello crearemos la entrada salida básica mediante el uso de los dos botones y el display de 8 segmentos de la placa.

Seguiremos interaccionando con ellos en C y aprenderemos a depurar diversas fuentes concurrentes de interrupción.

OBJETIVOS

- 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 bibliotecas de la placa)
- Desarrollar en C las rutinas de tratamiento de interrupción
- Aprender a utilizar periféricos, como los temporizadores internos de la placa y el teclado
- Depurar eventos concurrentes asíncronos
- Entender los modos de ejecución del procesador y el tratamiento de excepciones
- Gestionar el tiempo de trabajo del proyecto correctamente en función de la disponibilidad de acceso al sistema específico

ENTORNO DE TRABAJO

El entorno del trabajo es prácticamente el mismo que el de la práctica anterior. Las placas sólo pueden usarse en el **laboratorio 0.5b** durante el horario reglado de la asignatura. **IMPORTANTE:** partes de esta esta práctica requieren hardware específico y **no se puede hacer a distancia. El resto deberían realizarse como trabajo previo a las sesiones de laboratorio para no perder el tiempo de placa haciendo cosas que no la necesitan.**

Cuando se trabaja con un sistema real es muy normal no tenerlo siempre disponible. Por ello es imprescindible planificar adecuadamente el trabajo y en la medida de lo posible, abstraerse del dispositivo para poder avanzar en el proyecto.

Es conveniente **leer el enunciado completo de esta práctica antes de la primera sesión** y determinar las partes que se deben adelantar, las que necesitan obligatoriamente acceso al hardware y las que se deben abstraer para independizarnos de él.

MATERIAL DISPONIBLE

En Moodle de la asignatura 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:

  - o ***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.**
  - o ***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.
- Vuestros proyectos generados en la práctica 1.

ESTRUCTURA DE LA PRÁCTICA

Paso 1: Estudiar la documentación.

Paso 2: Asimilar el proyecto ejemplo que os proporcionamos, en el que se utilizan algunos de los dispositivos que se describen en este guion.

Tanto los botones, como el teclado, y los *timers*, 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:

- o Debéis entender cómo se controlan estos dispositivos, y cómo se comunican con el procesador a través de las interrupciones.
- o Debéis ser capaces de entender el código ensamblador que genera el compilador y explicar qué hace en cada paso. Ejecutar paso a paso el ensamblador que se genera a partir del código C de las rutinas de tratamiento de interrupción observando y haciendo notar las diferencias en su bloque de activación con respecto a otras subrutinas.
- o Debéis mejorar la gestión de la entrada salida en el proyecto que os damos:
  - o Convertir variables compartidas con la rutina de servicio de interrupción a “static”.
  - o Asignar “volatile” donde sea necesario.
  - o Impedir que al modificar un bit en un registro compartido se modifiquen también otros bits que pueden pertenecer, por ejemplo, a otros dispositivos.

Paso 3: Abstracción del Hardware. Es conveniente que sea posible trabajar gran parte de este proyecto sin necesidad de tener físicamente la placa. Para conseguirlo hay que **abstraerse del hardware** lo máximo posible. Para ello es imprescindible hacer una programación modular, donde se puedan probar cosas sin disponer del hardware por ejemplo encapsulando bien todas las funciones de entrada/salida de forma que se ejecuten sobre la placa si disponéis de ella, y en el emulador en caso contrario. Por ejemplo, podéis emular **el comportamiento de los periféricos con variables** en memoria que representen su estado, y que puedan ser modificadas desde el entorno y/o el programa al depurar. Se puede utilizar compilación condicional (**`#ifdef/#else/#endif`**) en el proyecto para poder pasar fácilmente de una versión a la otra. Para más información sobre la compilación condicional por favor consultar el manual de gcc:

`https://gcc.gnu.org/onlinedocs/cpp/Conditional-Syntax.html#Conditional-Syntax`

Debéis prestar especial atención a la inicialización de los periféricos y a su acceso desde el programa principal. La E/S debe estar convenientemente estructurada, las rutinas de servicio de interrupción no deben compartir variables con el programa principal.

Paso 4: Tratamiento de excepciones. Durante la ejecución del código se pueden dar casos en los que el procesador detecta un error. Por ejemplo, al intentar ejecutar una instrucción inválida o un acceso a memoria no alineado. En esos casos se produce una excepción. Deberéis realizar una función de tratamiento que capture las excepciones ***Data Abort***, ***Undefined*** *Instruction* y ***Softw**are **I**nterrupt (**SWI**)*. 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, actualizando una variable global de error y si se está ejecutando sobre la placa mostrando el código de error parpadeando por el `8leds` y parando la ejecución).

Paso 5: Desarrollo de una pila de depuración. Vamos a tener diversas funciones concurrentes en el sistema. Depurar en estos casos el código implica poder detectar por ejemplo condiciones de carrera producidas por eventos asíncronos. Para simplificar la labor anotaremos cada vez que se produzca un evento relevante. Debéis gestionar una **pila de depuración** en la que introduciréis datos de eventos que os resulten útiles, como por ejemplo las interrupciones. En cualquier momento se puede parar la ejecución y observando la pila poder determinar el orden de activación.

Debéis ubicar la pila al final del espacio de memoria, antes de las pilas de los distintos modos de usuario. Definid un tamaño suficiente para que las pilas no se solapen y gestionadla 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(uint8_t ID_evento, uint32_t auxData) que debéis implementar. Esta función apila 2 enteros. El primero incluirá en su byte más significativo el campo ***ID_evento***, que permita identificar qué interrupción ha saltado, concatenado con los 24 bits menos significativos del campo ***auxData*** (datos auxiliares del evento ocurrido, por ejemplo el botón pulsado). El segundo dato a apilar nos indicará el momento exacto en que se ha invocado a la función. Para ello hará uso de la biblioteca de medidas de tiempo ya desarrollada.

Como se piensa apilar eventos desde distintos módulos es conveniente definir adecuadamente los ***ID_evento*** de los eventos e incluirlos en cada módulo que los use.

Paso 6: Integración de juego. A lo largo de esta práctica y la siguiente vamos a ir incorporando al juego diversos periféricos con los que interactuar. Para estructurar la forma de jugar deberéis crear una función ***`reversi_main()`***. Esta función será llamada desde la actual función ***`Main()`*** una vez inicializada la placa y los periféricos. En dicha función tendréis un bucle infinito (el juego nunca acaba) siempre en ejecución.

El bucle está pendiente de una cola de eventos asíncronos generados por los las rutinas de servicio de las interrupciones de los periféricos (salvo del timer2 que sigue como en la práctica 1). Vamos a utilizar la pila de depuración. Cuando se ejecute el bucle principal se mirará si hay eventos sin tratar desde la ejecución anterior. Si no los hay se llamara a la función `dormir_procesador()` que se encargará de dormir al procesador dejándolo en modo de bajo consumo de energía si no hay nada que hacer (crear para depurar una función vacía).

Si existen eventos pendientes deberemos procesarlos y atender a la lógica del juego (tiempo de juego, latido, detección de pulsaciones de botones, si hay que colocar ficha, etc).

Los eventos pasados deben quedar en la pila de depuración para poder depurar viendo lo ocurrido en el pasado), el tamaño debe ser lo suficientemente grande como para almacenar al menos todos los eventos sin procesar en un ciclo.

```
void reversi_main(){ //pseudocódigo
	reversi_inicializar(…);
	while (1) {
		while (hay_eventos_encolados()) {
			//procesar eventos
			……
		}
		dormir_procesador();
	}
}
```

Paso 7: Latido. Para saber si el programa sigue vivo vamos a implementar un latido con los leds de la placa con la ayuda del **`timer0`**. Programa el `timer0` para que genere 60 eventos por segundo (es necesario reducir el número de interrupciones del timer0 a una por evento). El **led de la izquierda** deberá parpadear (on/off) a **4 Hz**.

```
			//procesar eventos
			if (ev_tick0) Latido_ev_new_tick();
```

La función `Latido_ev_new_tick()` se llamara desde el gestor de eventos y deberá contar el tiempo y decidir si debe encender o apagar el led. Al presentar este apartado debéis ser capaces de detallar la configuración de los registros de todos los temporizadores y tener indicados la resolución y rango de cada contador.

Paso 8: Eliminación de los rebotes en los pulsadores. Para poder jugar, vamos a utilizar los pulsadores de la placa, dado que son dispositivos mecánicos reales al tocarlos producen una señal oscilante (como ilustra la Fig. 1) que en argot se denominan rebotes. 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 (deberéis mantener un código de test para mostrar esta parte al profesorado en cualquier momento o poder probar los tempos en cada placa nueva).

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**.

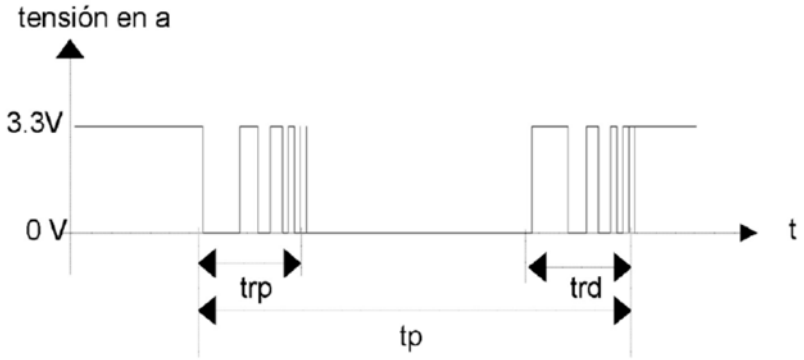

Fig. 1 Respuesta eléctrica de un pulsador

Para ello deberéis crear el dispositivo de bajo nivel **button.** El dispositivo dependerá del hardware. La RSI deberá tratar la pulsación del botón y el apilado del evento. Además de la rutina de servicio de interrupción deberéis implementar la siguiente interface con sus cabeceras:

enum estado_button {button_none, button_iz, button_dr}: conjunto de estados que pueden tener los botones.

button_iniciar(): Inicializa el dispositivo dejándolo listo para ser usado.

button_resetear(): re-activa interrupciones y el botón queda listo para ser pulsado de nuevo.

estado_button button_estado(): lee el estado actual del botón

button_ev_pulsacion (estado_button boton): Rutina para procesar el evento de nueva pulsación del botón llamada desde el gestor de eventos, se pasa como parámetro el botón pulsado.

button_ev_tick0 (): Rutina para procesar el evento de tick de timer0 para gestionar los retardos. En vez de usar el timer0 para el control de retardos podéis usar otro temporizador, crear las funciones necesarias para inicializarlo programando retardos, tratar la RSI apilando evento, y avisar del evento a button.

```
//procesar eventos
if (ev_tick0) button_ev_tick0(); //o equivalente
if (ev_pulsacion) button_ev_pulsacion(boton_pulsado);
```

Implementar una máquina de estados de un nuevo dispositivo software denominado **botones_antirebotes** cuya función o funciones serán invocadas desde el tratamiento de eventos de la función ***reversi_main()*** utilizando los eventos servidos por **button** y **timer0.** Se seguirá el siguiente esquema:

1) Al detectar la pulsación se identifica y se encola en la pila el evento, y se deshabilitan las interrupciones de los botones para ignorar los rebotes de entrada. El evento se tratará en ***reversi_main()***.

2) Tras un retardo inicial (*trp*) en número de **ticks** del *timer0* dependiente de cada placa, se monitoriza el botón cada 30ms aproximadamente para detectar cuándo el usuario levanta el dedo.
3) Cuando se suelta el botón, de nuevo se introduce un retardo para filtrar los posibles rebotes de salida (*trd*) (en número de **ticks** del *timer0*).
4) Tras el retardo final se habilita la IRQ de los botones quedando a la espera de una nueva pulsación.

IMPORTANTE: las rutinas de interrupción deberán ser **ligeras**. En ningún caso se permitirá hacer una espera activa dentro de una IRQ o de ***reversi_main()***. Identificar los retardos *trp* y *trd* de la cada placa con la ayuda de la pila de depuración. Los retardos pueden variar según el estado de cada placa. Los dispositivos **button** y **timer0** se consideran de bajo nivel y acceden directamente a los periféricos. El dispositivo **botones_antirebotes** no es de bajo nivel, debe ser lo más modular posible y no debe acceder a los registros de entrada salida.

Paso 9: **Vamos a jugar**. Para jugar al juego debemos saber en qué fila y columna quiere colocar ficha el usuario. En la práctica 1 lo hicimos mediante las variables fila, columna y ready. En esta práctica queremos realizarlo mediante los botones (dispositivo **botones_antirebotes**) y el dispositivo 8leds para la visualización de los valores.

Debemos añadir al bucle principal la gestión de **jugada_por_botones** con una nueva máquina de estados que permita al usuario introducir los movimientos (fila, columna y ready).

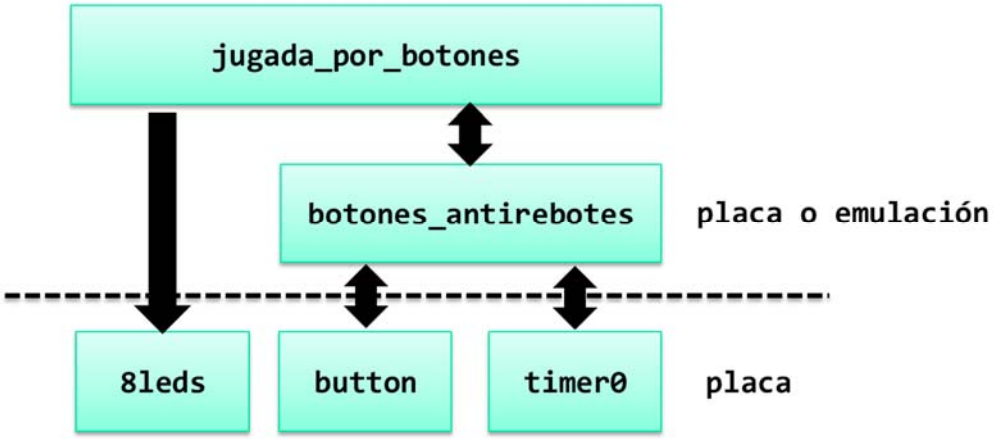

Fig. 3 Arquitectura sistema de botones

La máquina de estados debe seguir este esquema:

- Se seleccionará en que casilla se quiere jugar. 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** (de fila) en el 8leds.
  - Cuando el usuario pulse el botón izquierdo se visualizará un **1 en el 8leds** (primera celda).
  - Cuando levante el dedo, el número que hay en el 8led se mantendrá fijo.
  - Si de nuevo vuelve a pulsar el botón izquierdo, se irá incrementado como antes. Si llega al **9** se volverá al **1**.

- Si el usuario pulsa el botón derecho, confirmará el número actual. Entonces aparecerá una **C** (de columna) en el `8leds` y se repetirá el proceso para elegir la columna.
- Cuando se confirmen la fila y la columna el movimiento se procesará en la lógica del juego (`reversi_procesar`).

La máquina de estados no está aún completa. En la Práctica 3 añadiremos algunos otros elementos. Por tanto, tratad de hacer una máquina clara, ordenada y documentada con la que os resulte fácil trabajar.

APARTADO OPCIONAL 1

Introducir los datos pulsación a pulsación puede resultar aburrido. Una forma mucho más interactiva es utilizar auto-repetición. Esto es: si el usuario mantiene pulsado el botón de incrementar el número más de un cierto tiempo, el número se irá auto-incrementando.

Por ejemplo, una pulsación corta de menos de 1/3 segundo incrementará el número una unidad. Si pasado este tiempo se sigue manteniendo la pulsación, se incrementará nuevamente el número y se seguirá incrementando cada aprox. 180 milisegundos mientras se mantenga la pulsación.

APARTADO OPCIONAL 2

Permitir interrupciones anidadas

APARTADO OPCIONAL 3

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 8) 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 lunes 8 al martes 12 de noviembre).

La segunda parte (paso 9) deberá entregarse el 27 de noviembre.

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

ANEXO 1: REALIZACIÓN DE LA MEMORIA

En la memoria debéis seguir las pautas de la guía proporcionada por la asignatura. La extensión de la memoria debe ser de aproximadamente 10 hojas (en un estilo similar a este) o unas 4000 palabras, sin contar tabla de contenidos, índices, ni apéndices necesarios. Es obligatorio que incluya:

1. Resumen ejecutivo (una cara máximo).

2. Descripción de la biblioteca desarrollada para medir tiempos, es imprescindible incluir la metodología de medida (configuración, resolución y rango) y los resultados obtenidos al medir las funciones desarrolladas en la práctica 1. Comparar cuantitativamente y analizando el resultado de las distintas versiones de las funciones y con las estimaciones realizadas en la práctica 1, utilizando para ello el número de instrucciones ejecutadas y el tiempo de simulación/ejecución.
3. Breve descripción de la gestión de la entrada/salida en vuestro proyecto. Mostrar los diagramas de las máquinas de estados debidamente comentados y justificando las decisiones de diseño adoptadas, por ejemplo, la frecuencia de muestreo del botón pulsado elegida o seudocódigo o fragmentos de código representativos.
4. Descripción de los problemas encontrados en la realización de la práctica y sus soluciones.
5. Anexo: 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:

p2_NIP-Apellidos_Alumno1_NIP-Apellidos_Alumno2.zip

Por ejemplo: p2_345456-Gracia_Esteban_45632-Arribas_Murillo.zip