PRÁCTICA 2: SISTEMA DE E/S Y DISPOSITIVOS BÁSICOS

En esta práctica, vamos a aprender a trabajar con periféricos sencillos que se encuentran en todos los sistemas actuales: `timers` y `GPIO` (General-Purpose I/O). Los timers, o temporizadores, permiten generar retardos de forma precisa, realizar tareas periódicas, o medir tiempos. La GPIO nos permitirá emular botones, leds, o un teclado. Para gestionar estos elementos trabajaremos con un controlador de interrupciones (`VIC`). También configuraremos los pines de entrada del sistema para que reciba interrupciones externas que simularán botones. Además, vamos a aprender a dormir y a despertar al procesador. De esta forma cuando el procesador no está realizando cálculos (por ejemplo, porque está esperando a que el usuario realice su movimiento) podemos dormirlo, sin perder el estado, y reducir su consumo de energía.

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

OBJETIVOS

- 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 General Purpose Input/Output (GPIO)
- Depurar eventos concurrentes asíncronos
- Entender los modos de ejecución del procesador y el tratamiento de excepciones
- Ser capaces de depurar un código con varias fuentes de interrupción activas

ENTORNO DE TRABAJO

Continuaremos con el mismo entorno de trabajo, ARM Keil 5, que en la práctica anterior y emplearemos funcionalidades que hasta ahora no manejábamos, sobre todo para trabajar con interrupciones.

Es conveniente leer el enunciado completo de esta práctica antes de la primera sesión para hacer mejor uso de ella.

MATERIAL DISPONIBLE

Este guion sólo incluye algunas indicaciones, sin explicar en detalle el funcionamiento de cada periférico. Antes de utilizar cada uno de los periféricos debéis estudiar la documentación de la placa, forma parte de los objetivos de la asignatura que aprendáis a localizar la información en las hojas técnicas. Allí encontraréis la descripción detallada del interfaz de cada periférico que vamos a utilizar. No hace falta que entendáis las cosas que no utilizamos, pero sí que debéis saber qué hace cada registro de entrada/salida que utilicéis.

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

- Un proyecto que utiliza pulsadores virtuales y el timer 0 de la placa. Antes de escribir una línea de código debéis entender a la perfección este proyecto.
- Documentación original de la placa (proporcionada por la empresa desarrolladora): muy útil para ver más detalles sobre la entrada / salida, los periféricos del System-on-Chip empleado (SoC) y su mapa de memoria.

Además, disponéis de vuestros proyectos generados en la práctica 1.

Nota: Keil te permite alterar el valor de los registros de entrada/salida escribiendo en ellos directamente a través de interfaces gráficos (tal y cómo hemos hecho en la P1 modificando la memoria para indicar la fila y la columna). Esto puede usarse para hacer pruebas y depurar. Pero eso sólo se puede hacer en el simulador, no en un procesador de verdad, por tanto no sirve para desarrollar código. El código que entreguéis debe asignar los valores adecuados a los registros de entrada/salida. Es decir, debe funcionar sin necesidad de que asignemos ningún valor en los interfaces gráficos. La única excepción serán las entradas externas que simularán botones o teclados y se introducirán modificando los pines de entrada del sistema.

ESTRUCTURA DE LA PRÁCTICA

Paso 0: Estudiar la documentación

Paso 1: Utilización de los temporizadores

Nuestro SoC (System on Chip) LPC2105 incluye dos temporizadores de propósito general (timers 0 y 1, capítulo 14), junto con otros de propósito específico (`WatchDog` para resetear al procesador en caso de que malfuncionamiento, y `RTC` (Real Time Clock), para tener la hora y fecha).

Debéis aprender a utilizar estos temporizadores y realizar una pequeña biblioteca que los controle implementando las siguientes funciones o servicios al resto del software:

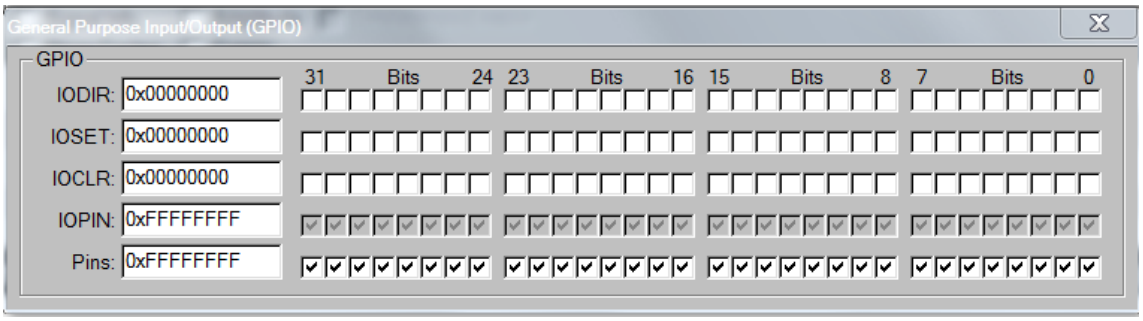
- temporizador_iniciar(): función que programa un contador para que pueda ser utilizado.
- temporizador_empezar(): función que inicia la ejecución de un contador de forma indefinida.
- temporizador_leer(): función que lee el tiempo que lleva contando el contador desde que se ejecutó temporizador_empezar y lo devuelve en microsegundos.
- temporizador_parar(): detiene el contador y devuelve el tiempo transcurrido desde temporizador_empezar.

Programa el *timer1* para que mida tiempos con la máxima precisión pero interrumpiendo (y por tanto sobrecargando al programa principal) lo menos posible.

Como vais a utilizar interrupciones a parte de estudiar los timers deberéis estudiar el VIC.

Para comprobar que vuestra librería funciona correctamente podéis utilizar la herramienta de `Keil Logic Analizer`. En moodle tenéis un video con un ejemplo de funcionamiento.

Paso 2: Programación de GPIO



Ventana de Keil para visualizar la GPIO, e introducir entradas en los pines del procesador. Se puede encontrar en *Peripherals>>GPIO*.

El `GPIO` es un periférico que se puede utilizar para conectar al chip elementos de entrada salida como leds, o entradas digitales. Hay disponible un puerto de 32 bits que se pueden utilizar como entrada o salida.
Debéis realizar una pequeña biblioteca que interactúe con la GPIO con las siguientes funciones

- GPIO_iniciar(): Permite emplear el GPIO y debe ser invocada antes de poder llamar al resto de funciones de la biblioteca.
- GPIO_leer(bit_inicial, num_bits): bit_inicial indica el primer bit a leer. num_bits indica cuántos bits queremos leer. La función devuelve un entero con el valor de los bits indicados. Ejemplo:
  - valor de los pines: `0x0F0FAFF0`
  - bit_inicial: 12 num_bits: 4
  - valor que retorna la función: `10` (lee los 4 bits 12-15)
- GPIO_escribir(bit_inicial, num_bits, valor): similar al anterior, pero en lugar de leer escribe en los bits indicados el valor (si valor no puede representarse en los bits indicados se escribirá los num_bits menos significativos a partir del inicial)
- GPIO_marcar_entrada(bit_inicial, num_bits): los bits indicados se utilizarán como pines de entrada.
- GPIO_marcar_salida(bit_inicial, num_bits): los bits indicados se utilizarán como pines de salida.

Paso 3: Diseño de una cola de eventos

En un sistema con distintos eventos asíncronos a la hora de depurar nos interesa no solo saber cómo hemos llegado a un punto, sino la secuencia previa de eventos; ya que el orden entre ellos o la producción de unos u otros puede condicionar la correcta ejecución del programa.

Para depurar la ejecución y la secuencia de eventos producida vais a crear una cola de eventos en la que se registren todos los eventos asíncronos del sistema.

La comunicación y sincronización entre los distintos módulos del diseño se realizará a través de una cola circular que almacenará la información de los últimos 32 eventos generados en el sistema. Un evento puede ser cualquier cosa que nos interese. Por ejemplo, una interrupción. Los eventos se guardarán con la función *cola_guardar_eventos(uint8_t ID_evento, uint32_t auxData)*. Esta función guardará en la cola dos palabras. La primera será el campo ID_evento, que permita identificar el evento (p.e. qué interrupción ha saltado), concatenado con los 24 bits menos significativos del campo *auxData* (que se puede usar para datos auxiliares sobre el evento ocurrido, por ejemplo el botón pulsado). La segunda 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 encolar eventos desde distintos módulos es conveniente definir adecuadamente los ID_evento de los eventos e incluirlos en cada módulo que los use con el fichero *eventos.h*. Deberéis definir un identificador único para cada posible tipo de evento que se pueda producir. Será necesario definir un nuevo módulo cola con ficheros *cola.h* y *cola.c*.

Como la iteración con el usuario se realiza por entrada/salida (E/S), el programa principal estará pendiente de ella para realizar cálculos. Podemos hacer uso de la cola de eventos para convertir el programa en orientado a eventos añadiendo un planificador. Básicamente el programa estará pendiente de la cola de eventos. Cuando aparezca un evento nuevo no tratado, se procesara. De esa forma la comunicación entre la E/S y el programa principal siempre se realizara mediante la cola de eventos.

Deberéis diseñar una función que compruebe si la cola tiene nuevos eventos, y otra que lea el evento más antiguo sin procesar. Además, al ser una cola circular, podría pasar que se borrasen eventos que no han sido procesados (*overflow*). Si eso ocurriese se debe encender el pin 30 de la GPIO y parar la ejecución (bucle infinito).

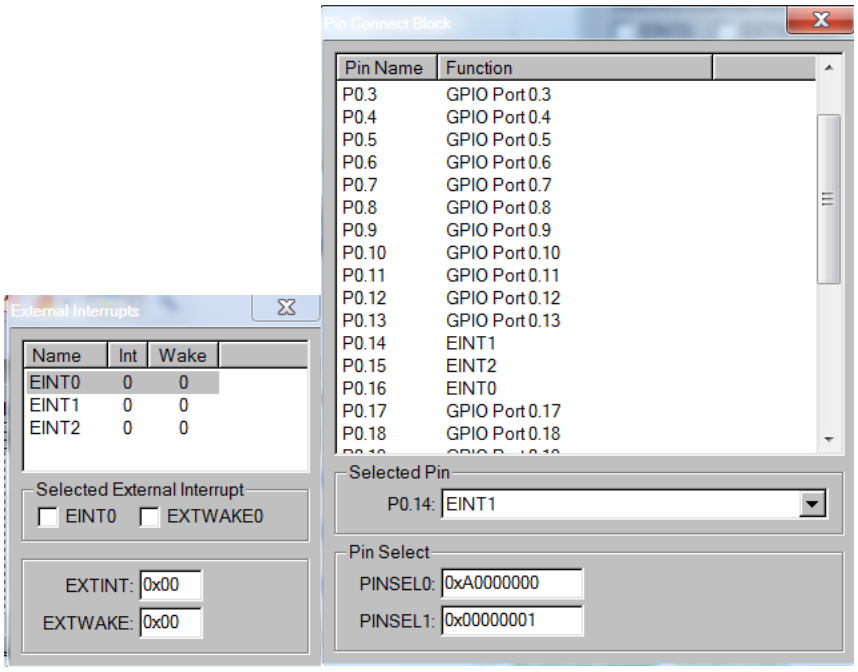
A la hora de planificar tareas a realizar es interesante poder programar una alarma. Utilizando el `timer0` añade las siguientes funciones a la librería del temporizador.

- temporizador_alarma(retardo): función que genera una notificación (evento en cola de eventos) dentro de un retardo indicado en milisegundos.
- temporizador_alarma_periodica(periodo): programa el temporizador para que realice una notificación periódica. El periodo se indica en ms.

Ambas alarmas pueden estar programadas simultáneamente.

Paso 4: Interrupciones externas

Nuestro SoC dispone de tres líneas de interrupción externas que permiten interaccionar al procesador con dispositivos de entrada salida que están fuera del chip. Vamos a utilizar dos de ellas `EINT0` y `EINT1` para simular botones.



Ventanas de Keil para visualizar las interrupciones externas (*Peripherals>>System Control Block>> External Interrupts*) y las conexión de los pines *(Peripherals>> Pin Connect Block)*. En este ejemplo se han configurado los pines 14, 15 y 16 para las interrupciones externas.

Para usarlas hay que conectarlas a los pines del sistema. Como hay muchas más posibles conexiones que pines, estos incluyen multiplexores que se configuran con los registros `PINSEL0` y `PINSEL1` (ver manual, capítulos 6 y 7).

Según el manual las interrupciones se deberían poder configurar para trabajar por flanco o nivel, y activas a alta o a baja (capítulo 3.6 del manual) utilizando los registros `EXTPOLAR` y `EXTMODE`. Pero ni en el simulador ni en las librerías aparecen esos registros. Por tanto, vamos a tener que trabajar con los valores por defecto: Las interrupciones se activan por nivel y son activas a baja: es decir que si hay un cero en el pin `P0.14` se activará la solicitud de interrupción de `EINT1`.

Como las interrupciones son muy rápidas y un botón puede estar bastante tiempo presionado, si no hacemos algo una misma pulsación podría generar multitud de interrupciones. Para evitarlo debéis implementar un esquema que garantice que sólo hay una interrupción por pulsación:

- Cuando se detecta la interrupción se procesa la pulsación. Procesar la pulsación consistirá en asignar a una variable (flag) el valor 1. Como usaremos dos botones tendremos dos flags: *nueva_pulsacion_eint0* y *nueva_pulsacion_eint1*. Diseñaremos dos funciones para leer estas variables (*nueva_pulsacion_0()* y *nueva_pulsacion_1()*) que devolverán 1 si ha habido una nueva pulsación, y otras dos para resetearlas a cero (*clear_nueva_pulsacion_0()* y *clear_nueva_pulsacion_1()*).
- Durante la gestión de la interrupción se deshabilitará la interrupción externa correspondiente en el VIC, para que no vuelva a interrumpir hasta que no termine la gestión de la pulsación.
- Se utilizará un temporizador para monitorizar la pulsación cada 50ms. Si la tecla sigue pulsada no se hará nada. En caso contrario se volverá a

habilitar la interrupción de ese botón. Importante: hay que limpiar su solicitud antes de volver a habilitarla tanto en el VIC, como en el registro EXTINT, o conforme se habilite se producirá una interrupción no deseada.

Paso 5: Eliminación de bucles de espera innecesarios

Normalmente el programa principal estará pendiente de si tiene algo que hacer en un bucle *while* en lo que se conoce como espera activa. Si la mayor parte del tiempo el planificador no tiene nada que hacer útil estará desperdiciando energía simplemente mirando si hay algo que hacer.

Cuando el procesador no tenga cálculos que hacer lo que haremos será dormirlo y por tanto poniéndolo en modos de menor consumo. Para ello utilizaremos el registro `PCON` (capítulo 3.10 del manual).

Haréis dos funciones. Una función que ponga al procesador en modo `idle` y otra que lo duerma (`power-down`). En `idle` el procesador se para, pero los periféricos del chip, como el temporizador, siguen activos y lo pueden despertar al realizar una interrupción. En el estado *power-down* los periféricos también entran en bajo consumo y dejan de funcionar pero se sigue manteniendo el estado. El procesador despertará si recibe una interrupción externa si las configuráis con el registro `EXTWAKE` (ver la configuración de las interrupciones externas en el manual).

Paso 6: integración de la I/O en el juego

Vamos a realizar toda la entrada utilizando la GPIO y las EINT0 y 1. La salida seguirá siendo mostrando el tablero en memoria.

Queremos incluir las siguientes funcionalidades:

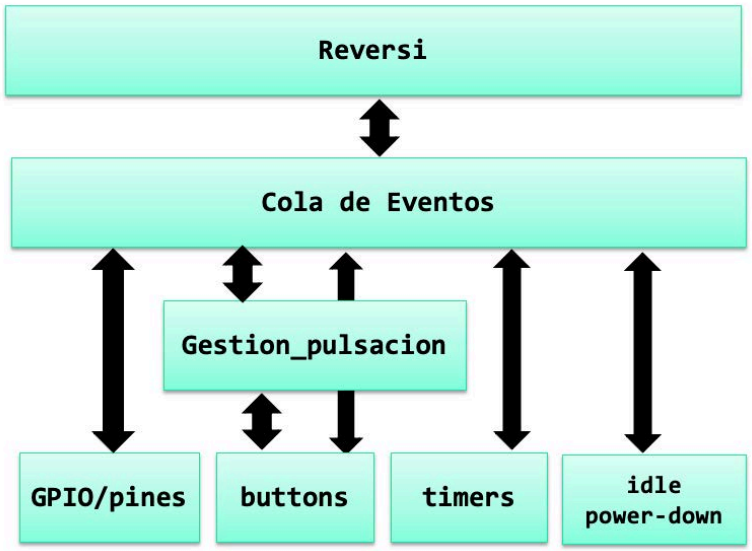
- Se programará una alarma periódica cada 10ms.
- Se medirá el tiempo de procesamiento de la IA en microsegundos. Es decir, desde que el jugador ha introducido su movimiento, hasta que la IA del juego ha calculado el suyo.
- La fila y la columna se introducirán utilizando la GPIO:
  - Fila: pines 2-0
  - Columna: pines 10-8
  - Fila y columna se numeran de 0 a 7.
- El botón `EINT0` (conectado al pin 16) indicará que el usuario ha introducido un nuevo movimiento.
- El botón `EINT1` (conectado al pin 14) se usa para pasar: si en lugar de introducir movimiento con EINT0, se pulsa el botón asociado a EINT1, quiere decir que el usuario pasa (cosa que sólo debería hacerse si no hay movimiento posible, aunque no lo comprobaremos).
- Cuando se introduzca un movimiento, o el jugador pase, se colocará la ficha en el tablero pero no se realizará el movimiento de forma inmediata. En su lugar la ficha parpadeará (se escribirá la ficha, luego un cero, luego la ficha…) cuatro veces por segundo durante tres segundos. Si en este

tiempo el jugador pulsa el botón EINT1 se cancelará el movimiento, si pulsa EINT0 se confirmará, y se procesará el movimiento. Transcurridos 3 segundos, si no se ha pulsado nada, el movimiento se dará por confirmado y se procesará.

- Para un óptimo funcionamiento del juego, cuando se coloca una ficha, el juego debería comprobar y verificar al seleccionar la celda donde se coloca la ficha que se puede poner en esa celda. Es un requisito opcional que añade funcionalidad.
- Si los dos jugadores pasan, la partida terminará. El juego se quedará sin hacer nada hasta que se pulse un botón (cualquiera de los dos). Al detectar la pulsación se comenzará una partida nueva.
- Sobre la gestión de `idle` y `power-down`:
    - Apagaremos el procesador siempre que estemos esperando que el jugador introduzca su movimiento ya que esto puede durar muchos segundos. Debéis configurar `EXTINT0` y `EXTINT1` para que despierten al procesador al pulsar usando el registro `EXTWAKE0` y `EXTWAKE1`. De esta forma cuando pulséis se despertará al procesador.
        - Para saber si el procesador está despierto o no durante el juego tendréis un led parpadeando 4 veces por segundo asociado al pin 31. Cuando el procesador esté apagado el timer también lo estará y el led no parpadeará.
    - Pondremos al procesador en idle, cuando estemos procesando las pulsaciones, es decir, esperando a que el jugador levante el dedo del botón, o cuando estemos esperando la confirmación de un movimiento. En idle los temporizadores están activos, así que periódicamente despertarán al procesador para ver si la pulsación ha terminado. EXTINT0 y EXTINT1 también están activas así que se procesará cualquier pulsación de confirmación o cancelación.

DESARROLLO DE CÓDIGO:

El objetivo es tener un código modular, en el que no se mezcle la lógica del juego y la lógica que controla los periféricos, ni la de un periférico con la de otro a no ser que sea imprescindible.
En la siguiente figura os presentamos un esquema de los módulos principales a desarrollar:



El código que desarrolléis debe cumplir las siguientes reglas:

- Cada módulo tendrá su código separando las definiciones, fichero.h, de su implementación, fichero.c. La configuración de ese módulo sólo se realizará con las funciones que incluyáis en la biblioteca. Por ejemplo, si el controlador de un periférico gestiona una variable interna, ésta no será visible de forma directa desde el exterior. La variable se encapsulara y se incluirán funciones visibles en el código para leerla o modificarla desde fuera si procede. Además, dichas variables no dependerán ni incluirán información del juego, sino información dependiente del periférico. Por ejemplo, puede haber una variable que indique si el botón ha sido pulsado, y otra cuántas interrupciones ha realizado un timer.
- La E/S es retardada, las interrupciones deben ser tan ligeras cómo se pueda. No se hacen cálculos, sólo se procesa el periférico y se informa que ha recibido una interrupción de un determinado tipo.
- La comunicación de los eventos asíncronos se realizará mediante la cola de eventos previamente creada y serán tratados por el planificador.
- Como se ve en la figura la gestión de la pulsación de un botón estará en un módulo aparte. En él se gestionará una sencilla máquina de estados para cada botón. Basta con dos estados (pulsado, no pulsado)
- Habrá que cambiar la lógica del reversi para adaptarla al esquema propuesto. Los cambios deben estar bien estructurados **siguiendo un esquema de máquina de estados**.

El código debe ser legible. En lugar de usar constantes directamente se recomienda definir etiquetas auto descriptivas. Por ejemplo, en el Reversi usamos FICHA_BLANCA en lugar de poner 01. De la misma forma no debéis usar estados tipo *S0*, *S1* o *S2*, sino por ejemplo; e_inicial, e_*pulsado*, e_*no_pulsado*, e_*esperando_pulsacion*, e_esperando_fin_pulsacion….

EVALUACIÓN DE LA PRÁCTICA

La primera parte de la práctica (paso 5) se deberá mostrar al profesor de vuestro grupo al inicio de vuestra tercera sesión de esta práctica (es decir, en vuestra sesión 6 del calendario de la EINA).

La práctica completa deberá entregarse aproximadamente el 26 de noviembre. Las fechas definitivas y turnos de corrección se publicarán en Moodle de la asignatura.