Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

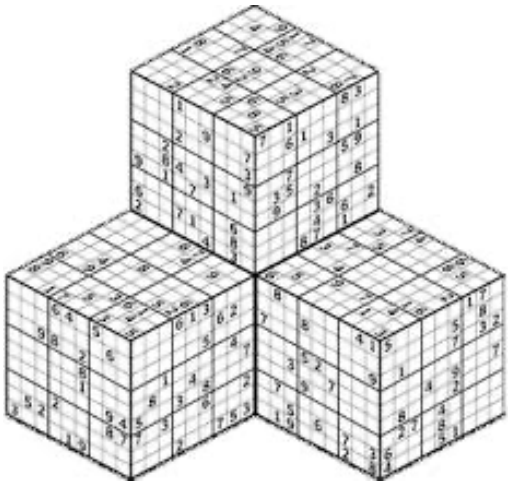
# PROYECTO HARDWARE

3º DE GRADO EN INGENIERÍA INFORMÁTICA, CURSO 2021/2022

UNIVERSIDAD DE ZARAGOZA

# PRÁCTICA 2

## SISTEMA DE E/S Y DISPOSITIVOS BÁSICOS

# PRÁCTICA 2: SISTEMA DE E/S Y DISPOSITIVOS BÁSICOS

En la Práctica 1 hemos desarrollado código para un procesador ARM y se ha ejecutado en un emulador. 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 y depurar** 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 **periféricos**, como los temporizadores internos de la placa y los botones (General Purpose Input/Output (GPIO)).
- **Gestionar el tiempo de trabajo** del proyecto correctamente, en función de la disponibilidad del alumno, del material necesario y del resto de asignaturas.

## ENTORNO DE TRABAJO

El entorno del trabajo es prácticamente el mismo que el de la práctica anterior. La diferencia principal es que en esta práctica ejecutaremos nuestro código en la placa ***Embest S3CEV40.*** Para ello, se incluye un soporte especial que facilita conectarse a la placa de desarrollo a través del puerto **JTAG,** permitiendo la ejecución paso a paso y acceso en tiempo real al estado tanto del procesador (incluyendo registro de estado y registros), como de la memoria.

En el apartado **1.1.2 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. También se recomienda ver el Vídeo 3.

**IMPORTANTE:** dado que algunas partes de esta práctica requieren hardware específico, es necesario el uso del maletín que se os proporciona.

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 pueden abstraer para independizarnos de él.

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

- Un proyecto que utiliza los pulsadores, `leds`, `8led` y el *`timer0`* 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:
  - ***P2-EC.pdf***: Describe la gestión de excepciones y el mapa de memoria que vamos a usar. **Hay que estudiarlo en detalle.**
  - ***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.**
- 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 ejemplo** que os proporcionamos, en el que se utilizan algunos de los dispositivos que se describen en este guión.

Tanto los botones, como 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:

- Debéis entender cómo se controlan estos dispositivos, y cómo se comunican con el procesador a través de las interrupciones.
- 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.
- Debéis mejorar la gestión de la entrada salida en el proyecto que os damos:
    - Convertir variables compartidas con la rutina de servicio de interrupción a "static".
    - Asignar "volatile" donde sea necesario.
    - Impedir que, al modificar un bit en un registro compartido, se modifiquen también otros bits que pueden pertenecer, por ejemplo, a otros dispositivos.

**La descripción del marco de pila** utilizado deberá aparecer convenientemente explicada en la memoria.

**Paso 3: Medidas de tiempo.** Debéis desarrollar una sencilla biblioteca que permita utilizar el **timer2** para medir tiempos. La biblioteca constará de las siguientes funciones, esquematizadas en la Fig. 1:

- **`void timer1_init()`**: configura el **timer 1** 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 **`timer1_numero_int`**, compartida entre la interrupción y el resto de funciones de ese módulo, 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.
- **`void timer1_start()`**: reinicia la cuenta de tiempo (contador y la variable), y comienza a medir.
- **`unsigned int timer1_count()`**: lee la cuenta actual del temporizador y el número de interrupciones generadas, y devuelve el tiempo transcurrido en microsegundos.
- **`unsigned int timer1_stop()`**: función que lee el tiempo que lleva contando el contador desde la última vez que se ejecutó timer1_start y lo devuelve en microsegundos.

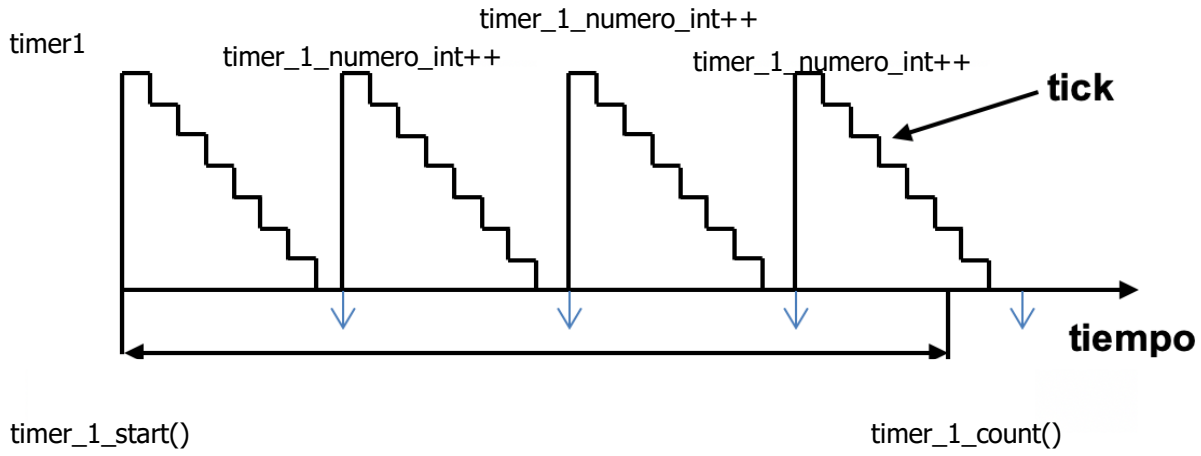



Fig. 1 Esquema del funcionamiento del timer 2

**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, 10 ms y 1 s) y aseguraos de que los resultados son parecidos. Tened en cuenta que 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: 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. Para ello, debéis gestionar una **pila de depuración** en la que introduciréis los datos de eventos que os resulten útiles, como por ejemplo, las interrupciones. De esta forma, 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 ***`cola_guardar_eventos(uint8_t ID_evento, uint32_t auxData)`*** que debéis implementar. Esta función apilará tres valores. El primero 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 en el Paso 3. El segundo permitirá identificar qué interrupción ha saltado o qué evento se ha producido. Finalmente, el tercer dato a apilar incluirá datos auxiliares del evento, por ejemplo, el botón pulsado, el número de ticks, etc.).

Como se piensa encolar eventos desde distintos módulos es conveniente definir adecuadamente el tipo de datos y 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*.

**Paso 5: 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 Fig. 2), que en argot se denominan rebotes. Usando la pila de depuración podremos observar cuántas veces se ha entrado en cada rutina de interrupción, saber si hay o no rebotes, y cuándo se producen (deberéis mantener un código de test para mostrar esta parte al profesor en cualquier momento, o poder probar los tiempos 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**.

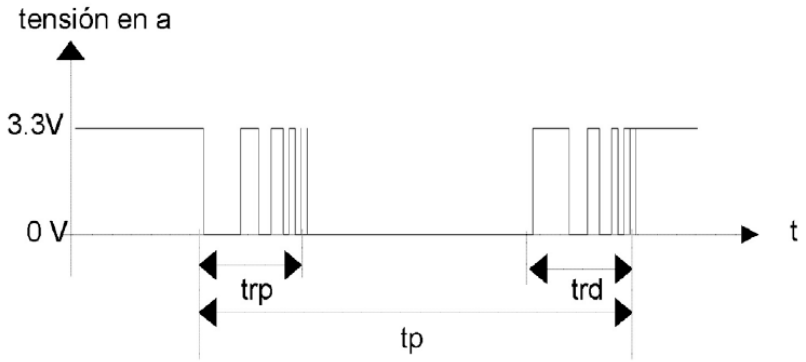



**Fig. 2 Respuesta eléctrica de un pulsador**

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 (*trp*) con un temporizador.
2) Tras el retardo, cada 50 milisegundos se monitoriza el botón con un temporizador para detectar cuándo el usuario levanta el dedo.
3) Cuando se suelta el botón, de nuevo se introduce un retardo (*trd*) para filtrar los posibles rebotes de salida.
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. Identificad los retardos *trp* y *trd* de la

placa con la ayuda de la pila de depuración. Tened en cuenta que los retardos pueden variar en cada placa, según su estado.

**Paso 6: Latido.** Para saber si el programa sigue vivo, vamos a implementar un latido con los leds de la placa con la ayuda del **`timer2`**. Para ello, debéis programar el `timer2` para que genere 60 eventos por segundo (es necesario reducir el número de interrupciones del timer2 a una por evento). El **led de la derecha** deberá parpadear (on/off) a **4 Hz**.

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 7**: **Vamos a jugar al Sudoku**. Para ello, debéis **adaptar 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 vuestro código fuente de la práctica anterior. Al código de la Práctica 1 debemos añadirle la máquina de estados que permita al usuario jugar. En concreto:

- El código comenzará a ejecutarse cuando se pulse un botón. Se calcularán por primera vez los candidatos de todas las celdas del tablero.
- A continuación, se seleccionará en qué casilla se quiere introducir un número. La fila y la columna se introducirán con el botón derecho (para elegir el número) y el izquierdo (para confirmar):
  - Al comenzar aparecerá una **F** (de fila) en el `8led`.
  - Cuando el usuario pulse el botón derecho se visualizará un **`1` en el `8led`**.
  - Cuando levante el dedo, el número que hay en el `8led` se mantendrá fijo.
  - Si de nuevo vuelve a pulsar el botón derecho, se irá incrementado. Si llega al **`9`,** se volverá al **`1`**.
  - Si el usuario pulsa el botón izquierdo, confirmará el número actual. Entonces aparecerá una **C** (de columna) en el `8led` y se repetirá el proceso para elegir la columna.
- Cuando se confirmen la fila y la columna, si esa celda es una **`pista`**, al no poder actualizarla, volveremos a pedir una nueva Fila. Si todo va bien, deberemos introducir el **Valor** comenzando con el **`1`**. Si se introduce un **`0`,** se producirá el borrado del valor de esa celda.
- Una vez introducido el nuevo valor, se procederá a comprobar si está dentro de la lista de candidatos para esa celda. En caso de que no lo esté, deberemos indicárselo al usuario, activando el bit correspondiente en la celda, y poniendo una **E** en el `8led`.
- Al introducir un nuevo valor, éste deberá ser propagado a sus vecinos (fila, columna, región), borrándolo de la lista de candidatos con la función más optimizada de **`candidatos_propagar`**. Sin embargo, hay dos casos especiales en los que invocar candidatos_propagar no es suficiente: (i) tanto si se ha borrado el valor de una celda (introduciendo un cero), como (ii) si se ha

modificado un valor previo, las listas de candidatos de todos sus vecinos están en un estado inconsistente y la única forma de recuperarlo es recalculando de nuevo todas las listas. Por ello, en vez de propagar, invocaremos con el nuevo valor a la función más optimizada de **`candidatos_actualizar`**.

Una vez comprobado que vuestro código funciona bien en la placa y que vuestra biblioteca mide bien los tiempos (Paso 3), debéis medir los tiempos de ejecución reales sobre el procesador de la placa.

Para ello, **repetiréis la comparación entre las versiones realizadas en la Práctica 1,** pero esta vez midiendo de forma precisa los tiempos con vuestra biblioteca. Es imprescindible que **comentéis los resultados obtenidos en la memoria de la práctica**. Para asegurar la calidad de los resultados, hay que cerciorarse que son estables realizando múltiples mediciones.

El compilador se ocupará de algunos de los detalles a bajo nivel, pero al presentar esta práctica, **debéis ser capaces de explicar cómo funciona el código y qué hace cada una de las instrucciones**.

## APARTADO OPCIONAL

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 1/2 de 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 500 milisegundos, mientras se mantenga la pulsación.

## EVALUACIÓN DE LA PRÁCTICA

La primera parte de la práctica (hasta el Paso 6) se deberá mostrar al profesor aproximadamente la penúltima semana de noviembre.

La segunda parte (paso 7) deberá entregarse el 14 de diciembre.

**En cualquier caso, las fechas definitivas de entrega se publicarán en Moodle de la asignatura**.

## ANEXO I. 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 o unas 4.000 palabras, sin contar la tabla de contenidos, los índices, ni apéndices necesarios. En cualquier caso, es obligatorio que incluya los siguientes puntos:

1. Resumen ejecutivo (una cara, máximo). El resumen ejecutivo puede ser considerado como un documento independiente del resto de la memoria que describe brevemente qué habéis hecho, por qué lo habéis hecho, qué resultados obtenéis y cuáles son vuestras conclusiones.
2. Descripción de la librería desarrollada para medir tiempos. Es imprescindible incluir la metodología de medida y los resultados obtenidos al medir las funciones desarrolladas en la Práctica 1. Comparad 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 ejecución.
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. 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...
6. Conclusiones.

Se valorará que el texto sea **claro y conciso**. Cuánto más fácil sea entender el funcionamiento del código y vuestro trabajo, mejor. **Revisad los documentos de recomendaciones para la redacción de una memoria técnica** disponibles en la web de la asignatura.

## ANEXO II. ENTREGA DE LA MEMORIA

La entrega de la memoria será a través de la página web de la asignatura (*moodle* en https://moodle.unizar.es/add/course/view.php?id=46338). Debéis enviar un fichero comprimido **en formato ZIP** con los siguientes documentos:

1. Memoria en formato PDF
2. Código fuente de los diferentes apartados

Se debe mandar un único fichero por pareja. El fichero se nombrará de la siguiente manera:

p1_NIP-Apellidos_Estudiante1_NIP-Apellidos_Estudiante2.zip

Por ejemplo: p1_345456-Gracia_Esteban_45632-Arribas_Murillo.zip