




10 DE ENERO DE 2016

SISTEMA DE E/S Y DISEÑO DE UNA PLATAFORMA INDEPENDIENTE

PRÁCTICA 2 Y 3 – PROYECTO HARDWARE

ALVARO JUAN CIRIACO (682531) – ALEJANDRO GUIU PÉREZ (680669)



ÍNDICE:

1. RESUMEN
2. OBJETIVOS
3. ESTRUCTURA DEL PROYECTO
4. GESTION ENTRADA/SALIDA
 - a. DIAGRAMA ESTADOS SUDOKU
 - b. DIAGRAMA GESTOR DE REBOTES
 - c. SALIDA GENERADA POR PANTALLA
5. LIBRERÍA PARA MEDIR TIEMPOS
 - a. TIMER0 – GESTOR REBOTES
 - b. TIMER2 – TIEMPO EJECUCIÓN
 - c. TIMER4 – TIEMPO TOTAL
6. GESTIÓN EXCEPCIONES
7. MEMORIA FLASH
8. CÓDIGO FUENTE
 - a. BUTTON.C
 - b. GESTION-EXCEPCIONES.C
 - c. GESTOR-PANTALLA.C
 - d. GESTOR-REBOTES.C
 - e. GESTOR-SUDOKU.C
 - f. LISTA-CIRCULAR-ESTATICA.C
 - g. MAIN.C
 - h. PILA-DEPURACION.C
 - i. TIMER2.C
 - j. TIMER4.C
9. PROBLEMAS ENCONTRADOS
10. CONCLUSIONES

1. RESUMEN:

En estas últimas prácticas se pide a los alumnos utilizar el código desarrollado en la práctica 1 en un entorno real. Para ello se tendrá que aprender a trabajar con la placa de desarrollo que hay en los laboratorios. Además, como objetivo principal se pide finalizar el proyecto en el que se lleva trabajando desde el principio de la asignatura, consiguiendo un sistema empotrado autónomo de los ordenadores del laboratorio con el que se pueda jugar al sudoku de forma directa. Para que esto se cumpla se van a utilizar tanto la pantalla LCD para visualizar el tablero de juego como cargar el código desarrollado anteriormente en la memoria Flash de la placa, de forma que al encender esta se pueda jugar sin necesidad de conectarse ni descargar el programa.

Para poder realizar la práctica de forma correcta y eficiente han sido necesarios: conocimiento iniciales de lenguaje ensamblador (obtenidos en asignaturas de años anteriores) y parte de los conocimientos obtenidos en AOC2. Además se ha utilizado el material disponible en Moodle para facilitar el trabajo, donde destacan el guión donde se explica el funcionamiento del LCD, el guión en el que se explica cómo programar la memoria flash, códigos fuentes para usar el LCD y para programar la flash, y manuales sobre el funcionamiento y programación de la placa.

Para el desarrollo de las prácticas se siguió la siguiente metodología de trabajo: antes de la sesión de laboratorios asignada al grupo, los alumnos se preparaban la parte que tocaba desarrollar en los laboratorios con la finalidad de avanzar trabajo sin la placa. Una vez se tenía la placa se probaba el código programado con anterioridad en casa y se solucionaban los errores que pudiese haber.

Teniendo en cuenta lo anterior, las prácticas se desarrollaron en el orden descrito en los guiones de cada práctica. Las primeras sesiones de laboratorio se utilizaron para programar y comprobar el funcionamiento del Timer2 y el tratamiento de excepciones. Posteriormente se creó la pila de depuración y la eliminación de los rebotes. Por último, se pasó a añadir las funcionalidades descritas en el guión de la práctica 2 (el código comenzará a ejecutarse cuando se pulse un botón,...).

Una vez comprobado el correcto funcionamiento y presentada la práctica 2 se procedió a desarrollar la práctica 3. Para esta se siguió la misma metodología de trabajo, aplicada al orden de tareas descrito en el guión de la práctica. Primero se desarrolló todo el trabajo que tenía relación con la pantalla LCD, tanto mostrar el sudoku, como la pantalla de instrucciones o la pantalla de finalización. Por último, se programó el código necesario para cargar el juego en la memoria flash de la placa, permitiendo así al usuario jugar al Sudoku sin necesidad de conectarse ni descargar el programa.

Con todo el trabajo ya finalizado se procedió a optimizar el código, realizando mejoras sobretudo en los apartados correspondientes a la práctica 2 y al desarrollo del código de la pantalla LCD realizado durante la práctica 3.

Tras realizar todo lo anterior se obtiene un producto que permite jugar al sudoku de manera independiente sin necesidad de estar conectado a un ordenador, con unos tiempos de ejecución mínimos (ya que se utiliza el código ARM-Thumb de la práctica 1 para recalcular los candidatos) y con una gestión de errores total, permitiendo al usuario jugar al sudoku a través de la pantalla LCD de la placa.

2. OBJETIVOS:

Entre los objetivos principales de la práctica se encuentran los siguientes:

- Interactuar con una placa real y ser capaces de ejecutar en ella el código desarrollado en la práctica anterior.
- Ser capaces de depurar un código con varias fuentes de interrupción activas.
- Aprender a desarrollar en C las rutinas de tratamiento de interrupción.
- Aprender a utilizar los temporizadores internos de la placa y el teclado.
- Profundizar en la interacción entre C/Ensamblador.
- Utilizar la pantalla LCD para visualizar el tablero.
- Cargar el código desarrollado durante las distintas prácticas en la memoria Flash de la placa mediante el estándar JTAG, de forma que al encenderla se pueda jugar sin necesidad de conectarse ni descargar el programa.

3. ESTRUCTURA DEL PROYECTO:

El proyecto realizado durante las prácticas consta de los siguientes ficheros:

- 8led.c: fichero C que contiene las funciones de control del display 8-segmentos.
- Bmp.c: fichero C que contiene las funciones de control y visualización del LCD.
- Button.c: fichero C que contiene las funciones de manejo y control de los pulsadores.
- GestionExcepciones.c: fichero C que contiene el código encargado de la gestión de las distintas excepciones que pueden ocurrir en la placa.
- GestorPantalla.c: fichero C encargado de mostrar por la pantalla LCD la pantalla de juego actual, ya sea la pantalla de instrucciones, la pantalla de juego o la pantalla de finalización.
- GestorRebotes.c: fichero C que contiene las funciones de manejo y control del timer 0 encargado de gestionar los rebotes de los botones.
- GestorSudoku.c: fichero C que contiene las funciones necesarias para el control del correcto funcionamiento del sudoku.
- Lcd.c: fichero C que contiene las funciones de visualización y control del LCD.
- Led.c: fichero C que contiene las funciones de control de los LED de la placa.
- ListaCircularEstatica.c: fichero C que implementa una lista circular estática utilizada como pila para la gestión de interrupciones de la placa.
- PilaDepuracion.c: fichero C que implementa la operación encargada de añadir un nuevo registro de excepción a la pila.
- Timer2.c: fichero C que contiene las funciones de control del timer 2 encargado de gestionar el tiempo de ejecución de las distintas operaciones que se realizan durante el sudoku.
- Timer4.c: fichero C que contiene las funciones de control del timer 4, encargado de gestionar el tiempo total de juego.
- Sudoku_2015.c: fichero C que contiene las operaciones de gestión del sudoku.
- Sudoku_2015_ARM-THUMB.asm: fichero ensamblador que contiene el código encargado de recalcular los candidatos. Este código se ejecuta al principio del sudoku y cada vez que el usuario inserta/borra un nuevo valor en el sudoku.
- Main.c: fichero principal del sudoku. Se encarga de inicializar todo lo necesario para el correcto funcionamiento del sudoku (placa, interrupciones, pulsadores, 8led, tratamiento de excepciones,...).

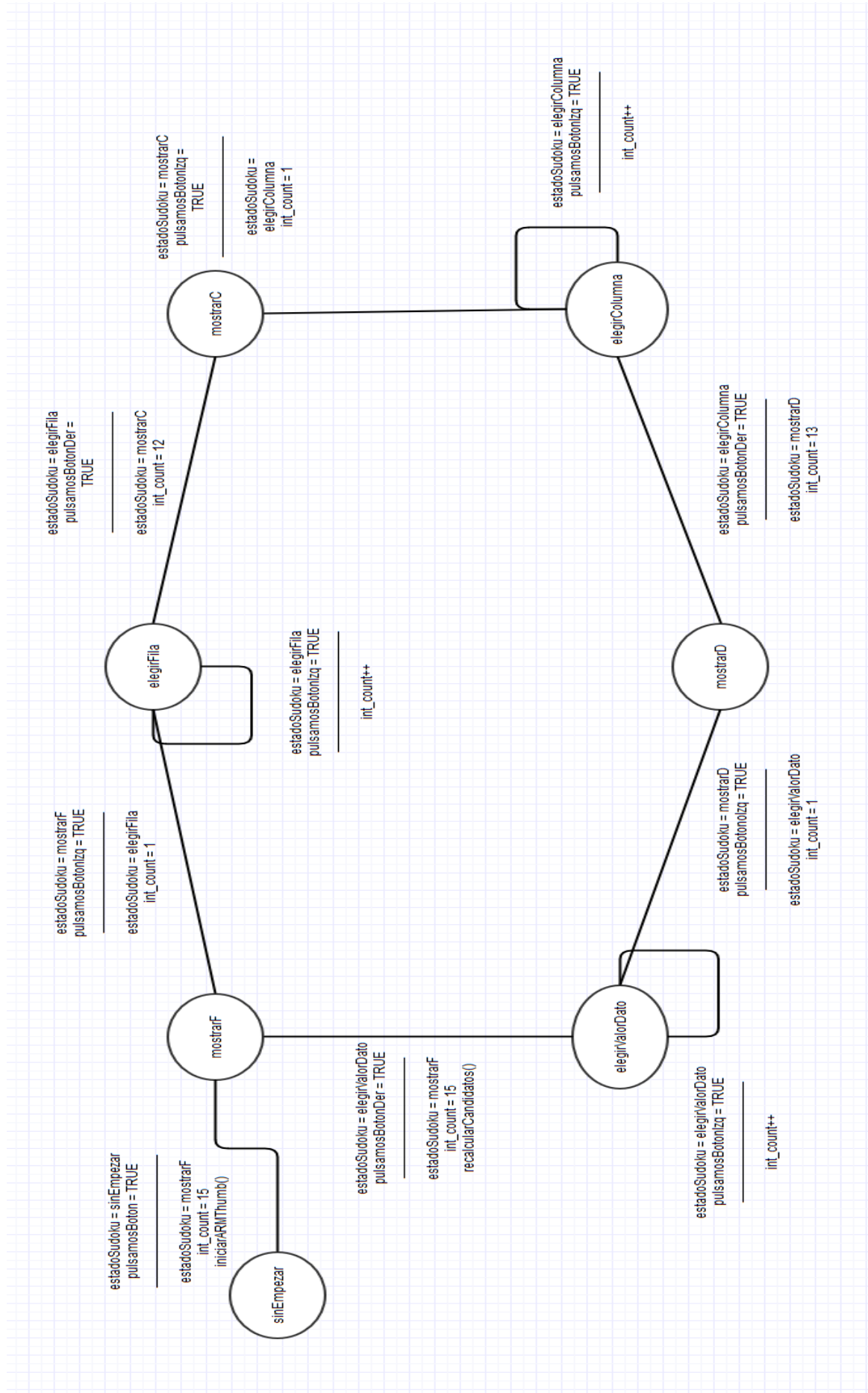
4. GESTIÓN ENTRADA/SALIDA:

4.1 DIAGRAMA DE ESTADOS SUDOKU:

A continuación se presenta el diagrama de estados que representa el funcionamiento interno del sudoku. Este consta de 7 estados que son los siguientes:

- **sinEmpezar**: estado inicial en el que se encuentra el sudoku antes de empezar el juego. Si pulsamos cualquier botón avanza al siguiente estado y da comienzo el juego. Esto hace que se dibuje en pantalla el sudoku con los distintos candidatos, además se inicializan todos los timers.
- **mostrarF**: estado que da comienzo a la inserción de un nuevo dato en el sudoku. Simboliza que el usuario tiene que insertar la FILA donde quiere añadir el nuevo número.
- **elegirFila**: mientras el usuario aprete el botón izquierdo irá aumentando el número que simboliza la fila en la que se va a insertar el nuevo dato. Una vez elegido, si el usuario presiona el botón derecho se avanza al siguiente estado.
- **mostrar**: simboliza que el usuario tiene que insertar la COLUMNA donde quiere añadir el nuevo dato.
- **elegirColumna**: mientras el usuario presione el botón izquierdo irá aumentando el número que simboliza la columna en la que se va a insertar el nuevo dato. Una vez elegido, si el usuario presiona el botón derecho avanzará al siguiente estado.
- **mostrarD**: simboliza que el usuario tiene que insertar el número que desee en la fila y columna anteriormente seleccionadas.
- **elegirValorDato**: mientras el usuario presione el botón izquierdo irá aumentando el número que simboliza el valor del dato que va a insertar. Una vez elegido, si el usuario presiona el botón derecho se insertará el número seleccionado en la fila y columna elegidas y se recalcularán de nuevo todos los candidatos. La pantalla se refrescará con los nuevos valores y el usuario podrá seguir insertando nuevos datos.

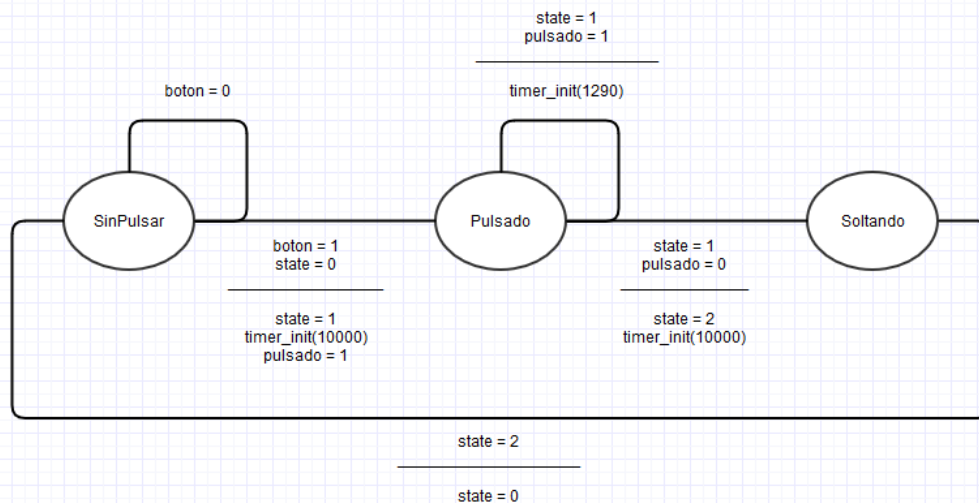
Nota: la variable `int_count` representa el número ASCII que aparece en el 8Led.



4.2. DIAGRAMA DE ESTADOS GESTOR REBOTES:

A continuación se presenta el diagrama de estados que representa la gestión de los rebotes. Este consta de 3 estados:

- **sinPulsar:** la placa se encuentra en este estado cuando no se está pulsando ningún botón. Se mantiene en él mientras el usuario no pulse ningún botón y avanza al siguiente estado cuando el usuario pulsa cualquiera de los botones de la placa. Esto hace que se inicialice el timer0 con un tiempo de 250ms (calculado para que funcione en todas las placas sin problemas). Esto hace que se desactive la interrupción de los botones con la finalidad de evitar los rebotes.
- **Pulsado:** la placa se encuentra en este estado cuando se está pulsando un botón. Mientras se mantenga dicho botón, se inicializa el timer0 con un tiempo de 10ms, esto se utiliza para comprobar si el botón sigue pulsado. Cuando el usuario suelta el botón se avanza al siguiente estado, inicializando el timer0 con un tiempo de 250ms, al igual que en el apartado anterior, con la finalidad de evitar los rebotes al soltar.
- **Soltando:** la placa se encuentra en este estado cuando el usuario está soltando un botón. Se reinician los botones y se vuelve a programar la interrupción de estos. Se avanza al estado inicial (sinPulsar).



4.3. SALIDA GENERADA POR LA PANTALLA:

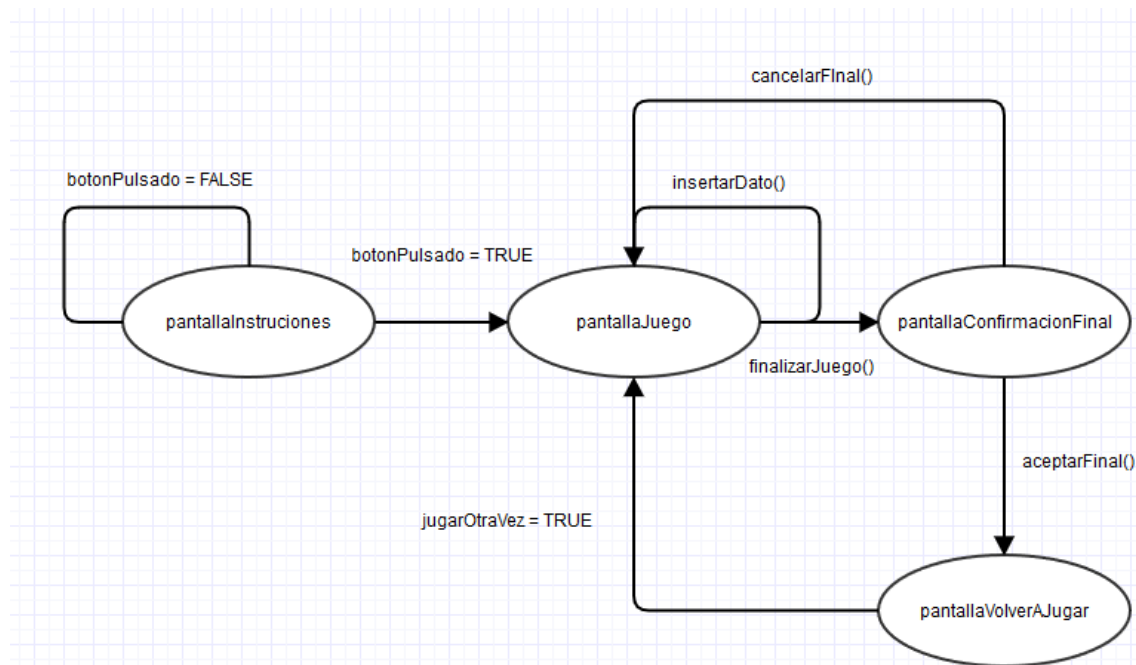
Para generar la salida correspondiente por la pantalla de forma correcta se ha utilizado la siguiente idea. Se ha creado un fichero denominado `gestorPantalla.c`, que es el encargado de gestionar lo que se muestra por pantalla para cada caso concreto del sudoku. En él se encuentran todas las operaciones de dibujar y de actualizar la pantalla.

En el método `main` del fichero `Main.c` el programa se encuentra en un bucle infinito hasta que el usuario desea finalizarlo. En este bucle el programa está preguntando constantemente si hay que actualizar el tiempo dibujado en la pantalla o hay que

refrescar por completo esta (ya sea porque el usuario ha añadido un nuevo número y hay que recalcular los candidatos, o porque el usuario desea finalizar el sudoku y hay que mostrarle el resultado final por pantalla).

Al inicio del juego se le muestra al usuario por la pantalla las instrucciones del sudoku, cuando este pulsa un botón se da comienzo al juego y aparece por pantalla el sudoku con los números pista y los candidatos calculados, junto al tiempo total de juego y el tiempo de ejecución de las operaciones (los cuales van aumentando conforme el usuario juega). Si el usuario inserta un nuevo número, la pantalla se refresca por completo actualizando así los candidatos del sudoku y señalando si existe algún error. Si el usuario desea finalizar el juego, la pantalla se refresca y muestra el resultado final del sudoku junto al tiempo total de juego y el tiempo total de ejecución de las operaciones realizadas (recalcular).

De esta forma, se mantiene la pantalla siempre actualizada con el tiempo (tanto total como de ejecución) correcto y con el sudoku actualizado.



5. LIBRERÍA PARA MEDIR TIEMPOS:

Para gestionar los tiempos se han utilizado 3 timers. El timer 0 encargado de la gestión de los rebotes, el timer 2 encargado de calcular el tiempo total de ejecución de las operaciones (recalcular, etc...) y el timer 4 encargado de contar el tiempo total de juego en un partida.

5.1. TIMER0 – GESTOR_REBOTES:

```
void gestionarRebotesPulsacion(int boton){
    estado = Pulsado; //va a Estadol
    botonPulsado = boton;
    timer0_init(tiempoAlPulsar_Soltar);
}

// Inicializa los valores del timer.
void timer0_init(int tiempo)
{
    /* Configuración controlador de interrupciones */
    rINTMSK &= ~(BIT_TIMER0); // Emascara el bit del timer 0.

    /* Establece la rutina de servicio para TIMER0 */
    pISR_TIMER0 = (unsigned) timer0_ISR;

    /* Configura el Timer0 */
    rTCFG0 |= 0x000000FF; // ajusta el preescalado a 255
    rTCFG1 &= 0xFFFFFFF0; // selecciona la entrada del mux que proporciona el reloj. La 00 corresponde a un divisor de 1/2.
    rTCNTB0 = tiempo; // valor inicial de cuenta (la cuenta es descendente)
    rTCMPB0 = 0; // valor de comparación
    /* establecer update=manual (bit 1) + inverter=on */
    rTCON = (rTCON & 0xFFFFFFF0) | 0x2;
    /* iniciar timer (bit 0) con auto-reload (bit 3)*/
    rTCON = (rTCON & 0xFFFFFFF0) | 0x9;
}
```

Timer encargado de gestionar el correcto funcionamiento de los rebotes de los distintos botones de la placa. Cuando se pulsa un botón se realiza la operación “gestionarRebotesPulsacion(int botón)” a la que se le pasa como parámetro un entero que representa el botón que ha pulsado el usuario. Esta operación se encarga de inicializar el timer con el tiempo necesario para evitar los rebotes (250 ms). Cuando se produzca la interrupción del timer0 programada anteriormente, si el usuario ya ha soltado el botón se volverá a activar la gestión de interrupción de los botones.

5.2. TIMER2 – TIEMPO EJECUCIÓN:

```
/*--- código de las funciones ---*/
void timer_ISR2(void)
{
    timer2_num_int++; //Aumentamos el número de interrupciones del timer2
    /* borrar bit en I_ISPC para desactivar la solicitud de interrupción*/
    rI_ISPC |= BIT_TIMER2;
}

void Timer2_Inicializar()
{
    /* Configuración controlador de interrupciones */
    // Enmascara el bit del timer 2.
    rINTMSK &= ~(BIT_TIMER2);

    /* Establece la rutina de servicio para TIMER2 */
    pISR_TIMER2 = (unsigned) timer_ISR2;

    /* Configura el Timer2 */
    rTCFG0 &= 0xFFFF00FF; // Preescalado a 0
    rTCFG1 &= 0xFFFF00FF; // Entrada del mux. 00 divisor 1/2
    rTCNTB2 = 65535; // Valor inicial de la cuenta del timer (valor máximo como se pide enunciado)
    rTCMPB2 = 0; // Valor con el que se comparará la cuenta del timer2
    /* Activamos los bits para iniciar la cuenta (12 y 13) */
    rTCON = (rTCON & 0xFFFF0FFF) | 0x2000; //Activamos manual update bit.
    rTCON = (rTCON & 0xFFFF0FFF) | 0x9000; //Activamos start/stop y desactivamos manual update bit a la vez.
}
```

Timer encargado de calcular el tiempo que cuesta realizar una operación en el sudoku (recalcular los candidatos cuando un usuario inserta un nuevo número en el sudoku).

Para ello se inicializa el timer con el preescalado a 0, y el valor inicial de la cuenta al máximo (tal y como se pedía en el enunciado de la práctica). Cuando el timer2 interrumpa se aumenta el valor de la variable “timer2_num_int” que marca el número de veces que ha interrumpido.

```
// Función encargada de reiniciar la cuenta de interrupciones del timer2
void Timer2_Empezar()
{
    timer2_num_int = 0;
}

// Devuelve el tiempo actual del timer 2.
uint32_t Timer2_Leer()
{
    uint32_t interrupcionesAntes, interrupcionesDesp, interrupcionesCuenta;
    interrupcionesAntes = timer2_num_int;
    uint32_t cuentaActual = rTCNT02;
    interrupcionesDesp = timer2_num_int;

    if (interrupcionesAntes == interrupcionesDesp) {
        interrupcionesCuenta = interrupcionesDesp;
    } else {
        interrupcionesCuenta = cuentaActual >= 65535/2 ?
            interrupcionesAntes : interrupcionesDesp;
    }
    return (interrupcionesCuenta*(65535/32)+((65535-cuentaActual)/32));
}
```

Cuando se desea obtener el tiempo de ejecución de la operación se realiza la operación Timer2_Leer(). Esta operación ha sido mejorada para evitar las condiciones de carrera, tal y como se ve en la imagen anterior. Primero se guarda el número de interrupciones del timer y posteriormente la cuenta actual del mismo. Cuando se han leído los dos datos anteriores se vuelve a leer el número de interrupciones por si estas han aumentado justo cuando se estaban leyendo las variables. Después se comprueba si el número de interrupciones leídas coinciden en los dos casos, si es así tomamos cualquiera de los dos y realizamos la operación necesaria para sacar el tiempo de ejecución (en microsegundos), si estas no coinciden comprobamos el valor de cuentaActual, si es mayor que la mitad del valor inicial de la cuenta nos quedamos con las primeras interrupciones tomadas, en caso contrario nos quedamos con las segundas. Después se realiza la operación necesaria para sacar el tiempo de ejecución (en microsegundos).

5.3. TIMER4 – TIEMPO TOTAL:

```
void timer4_init(void){ // iniciar timer4 para interrumpir cada segundo

    /* Configuración controlador de interrupciones */
    rINTMSK &= ~(BIT_TIMER4);

    /* Establece la rutina de servicio para TIMER4 */
    pISR_TIMER4=(unsigned)timer4_ISR;

    /* Configura el Timer4 */
    rTCFG0 |= ~(0x00FFFF); // preescalado del timer4 = 255 (compartirá preescalado con timer5)
    rTCFG1 &= 0xFFF0FFFF; // divisor del timer4 = 1/4
    rTCFG1 |= 0x10000; // divisor del timer4 = 1/4
    rTCNTB4 = 64516; // valor inicial de cuenta (la cuenta es descendente)
    rTCMPB4 = 0; // valor de comparación
    rTCON |= 0x200000; // activar manual update de los registros TCNTB4 y TCMPB4

    rTCON |= 0x900000; // lanza el timer4 con auto reload
    rTCON &= ~(0x200000); // Desactiva manual update del timer4
}
```

Timer encargado de llevar la cuenta (en segundos) del tiempo total de juego de la sesión actual. Este timer interrumpe cada segundo, siendo así la manera más fácil de llevar el cálculo del tiempo. Para que interrumpa cada segundo se buscó información en los distintos manuales de la placa. Se llegó a la conclusión de que con un preescalado de 255 y un divisor de $\frac{1}{4}$ el timer interrumpía cada 1,2 segundos. A partir de estos datos se obtuvo que si el timer se inicia con un valor inicial de 64516 este interrumpiría cada segundo.

```
void timer4_ISR(void)
{
    tiempoTotal++;
    actualizarTiempo = 1;
    rI_ISPC |= BIT_TIMER4; // limpiar bit de pendiente del timer4
}

void timer4_stop(void){
    //Deshabilita interrupciones de timer4
    rINTMSK |= BIT_TIMER4;
    //Para el reloj timer4
    rTCON &= ~(0x100000);
}

void timer4_reset(void) {
    tiempoTotal = 0;
}
```

Cada vez que el timer4 interrumpe se aumenta la variable tiempoTotal (lleva la cuenta del número de segundos que lleva el timer) y pone la variable “actualizarTiempo” a 1. Esto indica al gestorPantalla que tiene que actualizar la pantalla para que aparezca el nuevo tiempoTotal.

6. GESTIÓN EXCEPCIONES:

Durante la ejecución del código se pueden dar situaciones en las que el procesador no sepa que hacer. Para ello se ha realizado una función de tratamiento de excepciones. Independientemente de la excepción que salte, se invoca siempre a la misma función, la cual identifica el tipo y la instrucción causante.

```
void gestionGlobal(void){
    uint32_t direccion;
    uint32_t tipo;
    asm ("mov %0, lr;" : "=r" (direccion));
    asm ("mrs %0, cpsr;" : "=r" (tipo)); //tipo = cpsr
    switch(tipo&0x1F){ //compara con el modo (con los ultimos 5 bits)
        case 0x1B: //undefined
            push_debug(1,direccion);
            avisoExcepcion(1);
            break;
        case 0x17: //abort --> en data lr = lr+8, en prefetch lr = lr+4
            push_debug(2,direccion);
            avisoExcepcion(2);
            break;
    }
}
```

Se obtiene el tipo de excepción que es comparando el valor de la variable tipo con los valores de las distintas excepciones. Si la excepción es de tipo UNDEFINED se llama a la función avisoExcepcion(1) que se encarga de bloquear todo el proceso que lleva a cabo la placa y presentar por el 8Led de forma indefinida un 1 parpadeando, indicando así que ha habido una excepción.

Si la excepción es de tipo DATA o PREFETCH se gestiona de la misma forma (tal y como se indica en el pdf de la práctica). Se llama a la función avisoExcepcion(2) que se encarga de bloquear todo el proceso que lleva a cabo la placa y presenta por el 8Led de forma indefinida un 2 parpadeando.

```
/* Avisa de que ha ocurrido una excepcion mostrando el número de la excepcion */
void avisoExcepcion(int tipo){
    int paridad = 0; //Cuando sea par mostrara el dato, impar se apagara
    while(1){
        if(paridad%2==0){ //Par = muestra codigo de error
            D8Led_symbol(tipo);
        }else{ //Impar = se apaga para parpadear
            D8Led_symbol(16);
        }
        paridad++;
        Delay(5000);
    }
}
```

Para comprobar su correcto funcionamiento se creó un fichero ARM en el cual se introducía un nuevo dato en una dirección de memoria desalineada. Al ejecutar este fichero se producía una excepción de tipo DATA ABORT, lo que hacía que por el 8led apareciese un 2 parpadeando y se diese por finalizado el programa.

```
.global start
.arm
.data
.ltorg /*Garantiza la alineación*/
.text

.global Data_abort_test

Data_abort_test:
    PUSH {lr} /* Guarda la dirección desde la que se ha saltado */
    ldr r0, =0x4001 /* Apuntamos a una @memoria impar */
    mov r1, #2 /* r1 = 2 */
    str r2, [r0] /* Guardamos en @r2 el valor de r1 */
    POP {lr} /* Dirección de retorno */

fin: B .
.end
```

6.1. PILA DE DEPURACIÓN:

Esta pila se encarga de gestionar la introducción de datos que resultan útiles para depurar eventos asíncronos como interrupciones, excepciones,... Esta pila está basada en una lista circular estática. Cada dato que se introduce en ella consta de un parámetro ID_evento que permite identificar que interrupción ha saltado, un parámetro auxData que contiene datos aclaratorios, y un último dato que indica el momento exacto en el que se ha invocado a la función para apilar la interrupción.

```
void push_debug(int ID_evento, int auxData);
// Funcion encargada de apilar un nuevo elemento en la pilaDepuracion
void push_debug(int ID_evento, int auxData){
    uint32_t tiempo = (leerTiempoTimer4()); // Tiempo en el que se ha producido la interrupción
    anadirElemento(ID_evento, auxData, tiempo);
}
```

Como se observa en la imagen, la operación recibe como parámetros un ID_evento y un auxData, esta lee el tiempo actual del timer y llama a la operación añadirElemento, que se encarga de añadir un nuevo elemento en la lista circular estática implementada anteriormente.

La lista está formada por elementos de tipo “tipoNodo” formados por dos enteros (ID_evento y auxData) y un dato de tipo uint32_t (tiempo). Esta tiene su inicio en la dirección de memoria 0xc7fedfc tal y como se pide en el pdf de la práctica y tiene un tamaño concreto (260 bytes).

```
//Creacion del tipo de dato tipoNodo (representa el elemento a apilar)
typedef struct{
    int ID_evento; // Identificador del evento.
    int auxData; // Datos auxiliares
    uint32_t tiempo; // Momento de el que tuvo lugar la interrupción
} tipoNodo;

tipoNodo* lista = (tipoNodo *) 0xc7fedfc; // Puntero a la posicion inicial de la lista. (260 bytes)

void anadirElemento(int identificador, int dato, int tiempo);

// Indice que marca el ultimo elemento de la pila
static int indice = 0;

// Operacion que se encarga de añadir un nuevo elemento a la lista
void anadirElemento(int identificador, int dato, int tiempo){
    lista[indice].ID_evento = identificador;
    lista[indice].auxData = dato;
    lista[indice].tiempo = tiempo;
    indice++;

    //Resetea el indice de la lista si esta llega al tope
    if((int)&lista[indice]>0xc7fef00){
        indice = 0;
    }
}
```

7. MEMORIA FLASH:

Para conseguir que la plataforma sea autónoma y se pueda utilizar sin conectarse a ningún PC se ha cargado nuestro programa en la memoria Flash de la placa utilizando openOCD y JTAG. Para ello se siguieron los siguientes pasos:

1. Se generó el fichero binario a escribir en la memoria Flash a partir del fichero .elf del proyecto. Esto se realizó a través de la línea de comandos escribiendo lo siguiente:

**“C:/Program Files/EclipseARM/sourcery-g++-lite-arm-2011.03/bin/arm-none-eabi-obcopy -O binary
C:/Users/a680669/workspace/Practica3/Debug/Practica3.elf
C:/Users/a680669/workspace/Practica3/common/Practica3.bin”**

2. A continuación se introdujo el fichero binario en la memoria Flash de la placa con el siguiente comando:

**"C:/Program Files/EclipseARM/openocd-0.7.0/bin/openocd-0.7.0.exe -f
test/arm-fdi-ucm.cfg -c "program
C:/Users/a680669/workspace/Practica3/common/Pratica3.bin 0x00000000""**

3. Al encender la placa, el código ya está almacenado en la memoria Flash, sin embargo no se puede ejecutar correctamente porque las direcciones reales no cuadran con las que ha utilizado el linker. Para solucionar el problema se copió a la RAM. Para ello se añadió el código encargado de: inicializar el controlador de memoria, copiar el contenido de la ROM a la memoria RAM al comienzo de la ejecución, y el código encargado de saltar al Main recién copiado en la RAM.

4. Después de realizar todos los cambios el código queda de la siguiente manera:

```
#;*****
#;* Setup IRQ handler *
#;*****
ldr    r0,=HandleIRQ    /* This routine is needed */
ldr    r1,=IsrIRQ        /* if there isn't 'subs pc,lr,#4' at 0x18, 0x1c */
str    r1,[r0]

#*****
#* Copy and paste RW data/zero initialized data *
#*****
LDR    r0, =0 /* Get pointer to ROM data */
LDR    r1, =Image_RO_Base /* and RAM copy */
LDR    r3, =Image_ZI_Base
/* Zero init base => top of initialised data */

CMP    r0, r1 /* Check that they are different */
BEQ    F1
F0:
CMP    r1, r3 /* Copy init data */
LDRCC  r2, [r0], #4 /* --> LDRCC r2, [r0] + ADD r0, r0, #4 */
STRCC  r2, [r1], #4 /* --> STRCC r2, [r1] + ADD r1, r1, #4 */
BCC    F0
F1:
LDR    r1, =Image_ZI_Limit /* Top of zero init segment */
MOV    r2, #0
F2:
```

5. Para resetear la placa y dejarla en buen estado para que pudiese ser utilizada sin problemas por los demás grupos de alumnos se utilizaba el siguiente comando:

**"C:/Program Files/EclipseARM/openocd-0.7.0/bin/openocd-0.7.0.exe -f
test/arm-fdi-ucm.cfg -c "program
C:/Users/a680669/workspace/Practica3/common/backup flash.bin
0x00000000""**

8. CÓDIGO FUENTE:

8.1. BUTTON.C

```
/*
 * Archivo: button.c
 * Autores: Alejandro Guíu Perez - 680669 y Alvaro Juan Ciriaco - 682531
 * Descrip: Funciones de manejo de los pulsadores (EINT6-7)
 * Version:
 */

/*--- ficheros de cabecera ---*/
#include "44b1ib.h"
#include "44b.h"
#include "def.h"
#include "gestorSudoku.h"
#include "gestorRebotes.h"
#include "pilaDepuracion.h"
#include <inttypes.h>

enum {botonIzquierdo = 0, botonDerecho = 1};
enum {botonIzqMascara = 6, botonDerMascara = 7};
enum {interrupcionBoton = 10};

/*--- declaracion de funciones ---*/
void Eint4567_ISR(void) __attribute__((interrupt("IRQ")));
void Eint4567_init(void);

/*--- codigo de funciones ---*/
// Función que inicializa los pulsadores.
void Eint4567_init(void)
{
    /* Configuración del controlador de interrupciones. Estos registros están definidos en 44b.h */
    rI_ISPC = 0x3fffffff; // Borra INT_PND escribiendo 1s en I_ISPC
    rEXTINT_PND = 0xf; // Borra EXTINT_PND escribiendo 1s en el propio registro
    rINTMSK &= ~(BIT_EINT4567); // Enmascara todas las líneas excepto eint4567.

    /* Establece la rutina de servicio para Eint4567 */
    pISR_EINT4567 = (int)Eint4567_ISR;

    /* Configuración del puerto G */
    rPCONG = 0xffff; // Establece la función de los pines (EINT0-7)
    rPUPG = 0x0; // Habilita el "pull up" del puerto
    rEXTINT = rEXTINT | 0x22222222; // Configura las líneas de int. como de flanco de bajada

    /* Por precaución, se vuelven a borrar los bits de INT_PND y EXTINT_PND */
    rI_ISPC |= (BIT_EINT4567);
    rEXTINT_PND = 0xf;
}
```

```
/* Gestión de la interrupción de la pulsación de un botón
 * Si se pulsa el botón izquierdo o el botón derecho, informa
 * al gestor de rebotes y al gestor de sudokus de la acción.
 */
void Eint4567_ISR(void)
{
    int which_int = rEXTINT_PND;
    switch (which_int)
    {
        case 4: // En caso de que sea el botón izquierdo
            gestionarRebotesPulsacion(botonIzqMascara); //informa al gestor de rebotes de la pulsación
            gestionarSudoku(botonIzquierdo); //informa al gestor de sudoku de la pulsación
            push_debug(interrupcionBoton, botonIzquierdo); //Apilamos la interrupción de pulsación con botón izquierdo (4)
            break;
        case 8: // En caso de que sea el botón derecho
            gestionarRebotesPulsacion(botonDerMascara); //informa al gestor de rebotes de la pulsación
            gestionarSudoku(botonDerecho); //informa al gestor de sudoku de la pulsación
            push_debug(interrupcionBoton, botonDerecho); //Apilamos la interrupción de pulsación con botón derecho (8)
            break;
        default: // Si es cualquier otro botón, se ignora.
            break;
    }
    rINTMSK |= (BIT_EINT4567); //Desactivamos la ISR
    /* Finalizar ISR */
    rEXTINT_PND = 0xf; // Borra los bits en EXTINT_PND.
    rI_ISPC |= BIT_EINT4567; // Borra el bit pendiente en INT_PND.
}
```

8.2. GESTIONEXCEPCIONES.C

```
/*
 * Archivo: gestionExcepciones.c
 * Autores: Alejandro Guiu Perez - 680669 y Alvaro Juan Ciriaco - 682531
 * Descrip: gestiona todas las excepciones de la placa
 */

/*--- ficheros de cabecera ---*/
#include "44b1ib.h"
#include "44b.h"
#include "stdio.h"
#include <inttypes.h>

/*--- variables globales ---*/
int NUM_ALERTAS = 20;

/*--- funciones externas ---*/
extern void D8Led_symbol(int value);
extern void push_debug(int ID_evento, int auxData);

/*--- declaracion de funciones ---*/
void avisoExcepcion(int tipo);
void init_excep();
void gestionGlobal(void) __attribute__((interrupt("ABORT")));
void gestionGlobal(void) __attribute__((interrupt("UNDEF")));
void gestionGlobal(void) __attribute__((interrupt("ABORT")));

/*--- codigo de funciones ---*/
/**
 * Funcion encargada de gestionar de forma global
 * las distintas excepciones que ocurran. Ademas
 * se encarga de diferenciar de que tipo es.
 */
void gestionGlobal(void){
    uint32_t direccion;
    uint32_t tipo;
    asm ("mov %0, lr;" : "=r" (direccion));
    asm ("mrs %0, cpsr;" : "=r" (tipo)); //tipo = cpsr
    switch(tipo&0x1F){ //compara con el modo (con los ultimos 5 bits)
        case 0x1B: //undefined
            push_debug(1,direccion);
            avisoExcepcion(1);
            break;
        case 0x17: //abort --> en data lr = lr+8, en prefetch lr = lr+4
            push_debug(2,direccion);
            avisoExcepcion(2);
            break;
    }
}
```

```
/**
 * Funcion encargada de mostrar por D8Led el numero
 * que corresponda a la excepcion que se ha gestionado.
 * El programa finaliza por completo.
 */
void avisoExcepcion(int tipo){
    int paridad = 0; //Cuando sea par mostrara el dato, impar se apagara
    while(1){
        if(paridad%2==0){ //Par = muestra codigo de error
            D8Led_symbol(tipo);
        }else{ //Impar = se apaga para parpadear
            D8Led_symbol(16);
        }
        paridad++;
        Delay(5000);
    }
}

void init_excep() {
    pISR_DABORT = (unsigned) gestionGlobal;
    pISR_UNDEF = (unsigned) gestionGlobal;
    pISR_PABORT = (unsigned) gestionGlobal;
}
```


8.3. GESTORPANTALLA.C

```
/*-----
* Fichero: gestorPantalla.c
* Autores: Alejandro Guiu P{rez - 680669 y {lvaro Juan Ciriaco - 682531
* Descrip: Gestiona lo que se muestra por pantalla para el caso concreto de un sudoku
*-----*/

/*--- ficheros de cabecera ---*/
#include "def.h"
#include "44b.h"
#include <stdlib.h>
#include "44b1ib.h"
#include "lcd.h"
#include "bmp.h"
#include <inttypes.h>
#include "sudoku_2015.h"
#include "itoa.h"
#include "timer2.h"

/**
 * Pantallas posibles:
 * pantallaJuego: Muestra los elementos principales del juego, el sudoku y los tiempos.
 * pantallaConfirmacionFinal: Solicita al usuario confirmaci3n del final de la partida.
 * pantallaVolverAJugar: Muestra la pantalla de juego pausada, esperando para volver a empezarla.
 * pantallaInstrucciones: Muestra las instrucciones de juego y se queda a la espera de empezar.
 */
enum {pantallaJuego = 0, pantallaConfirmacionFinal = 1, pantallaVolverAJugar = 2,
      pantallaInstrucciones = 3};

/*--- variables internas ---*/
const int margenIzquierdoLeyenda = 0; // Distancia desde el margen izquierdo hasta la leyenda
const int margenSuperiorLeyenda = 5; // Distancia desde el margen superior hasta la leyenda
const int margenIzquierdoSudoku = 15; // Distancia desde el margen izquierdo hasta el sudoku
const int margenSuperiorSudoku = 30; // Distancia desde el margen superior hasta el sudoku
const int sizeOfCell = 22; // Tamafio de una celda
int idSiguientePantalla = 0; // Identificador de la siguiente pantalla a mostrar
char* leyenda; // Texto correspondiente a la leyenda que se va a mostrar
char tiempoCalculoEnPantalla[15]; // Coste del {ltimo calculo en formato de texto
char tiempoTotalEnPantalla[15]; // Tiempo desde que se ha iniciado la partida en formato de texto
```

```
/**
 * Pantallas posibles:
 * pantallaJuego: Muestra los elementos principales del juego, el sudoku y los tiempos.
 * pantallaConfirmacionFinal: Solicita al usuario confirmaci3n del final de la partida.
 * pantallaVolverAJugar: Muestra la pantalla de juego pausada, esperando para volver a empezarla.
 * pantallaInstrucciones: Muestra las instrucciones de juego y se queda a la espera de empezar.
 */
enum {pantallaJuego = 0, pantallaConfirmacionFinal = 1, pantallaVolverAJugar = 2,
      pantallaInstrucciones = 3};

/*--- variables internas ---*/
const int margenIzquierdoLeyenda = 0; // Distancia desde el margen izquierdo hasta la leyenda
const int margenSuperiorLeyenda = 5; // Distancia desde el margen superior hasta la leyenda
const int margenIzquierdoSudoku = 15; // Distancia desde el margen izquierdo hasta el sudoku
const int margenSuperiorSudoku = 30; // Distancia desde el margen superior hasta el sudoku
const int sizeOfCell = 22; // Tamafio de una celda
int idSiguientePantalla = 0; // Identificador de la siguiente pantalla a mostrar
char* leyenda; // Texto correspondiente a la leyenda que se va a mostrar
char tiempoCalculoEnPantalla[15]; // Coste del {ltimo calculo en formato de texto
char tiempoTotalEnPantalla[15]; // Tiempo desde que se ha iniciado la partida en formato de texto

/**
 * Funcion encargada de actualizar el contenido de la pantalla, leyendo
 * el identificador de la pantalla que se debe mostrar y actualizando
 * la leyenda si fuera necesario.
 */
void actualizarPantalla(){
    if(idSiguientePantalla == pantallaJuego) {
        leyenda = "Introduzca Fila A para acabar la partida";
        dibujar_sudoku();
    } else if (idSiguientePantalla == pantallaConfirmacionFinal) {
        mostrarConfirmacionFinal();
    } else if (idSiguientePantalla == pantallaVolverAJugar) {
        leyenda = "Pulse un boton para jugar";
        dibujar_sudoku();
    } else if (idSiguientePantalla == pantallaInstrucciones) {
        mostrarInstrucciones();
    }
}
```

```

/**
 * Dibuja por pantalla el sudoku, incluidas las l neas, las numeraciones,
 * los datos de las celdas, los distintos colores de cada celda.
 */
void dibujar_sudoku(){
    /* clear screen */
    Lcd_Clr();
    Lcd_Active_Clr();

    #ifdef Eng_v
    Lcd_DspAscII8x16(10,0,DARKGRAY,"Embest S3CEV40 ");
    #else
    #endif

    Lcd_DspAscII8x16(margenIzquierdoLeyenda,margenSuperiorLeyenda,BLACK,leyenda);
    int x0 = margenIzquierdoSudoku, x1, y0 = margenSuperiorSudoku, y1=235;
    //posiciones para colocar numeracion del 1 al 9
    int margenIzquierdoNumeros = x0-10;
    int margenSuperiorNumeros = y0-10;
    int i;
    INT8U* numero = "";
    //Coloca los numeros para las columnas y filas
    for (i = 1; i<10; i++) {
        numero[0]=i+'0';
        Lcd_DspAscII8x16(x0+8,margenSuperiorNumeros,BLACK,numero);
        Lcd_DspAscII8x16(margenIzquierdoNumeros,y0+8,BLACK,numero);
        x0 += sizeofCell;
        y0 += sizeofCell;
    }
    //Se dibujan las lineas horizontales del sudoku:
    x0=margenIzquierdoSudoku; y0=margenSuperiorSudoku+4; x1=215;
    for(i=0; i<10; i++){
        if(i%3==0){
            Lcd_Draw_HLine(x0,x1,y0,15,3);
        } else {
            Lcd_Draw_HLine(x0,x1,y0,15,1);
        }
        y0 += sizeofCell;
    }
    //Se dibujan las lineas verticales del sudoku:
    x0=margenIzquierdoSudoku; y0=margenSuperiorSudoku+4;
    y1=232;
    for(i=0; i<10; i++){
        if(i%3==0){
            Lcd_Draw_VLine(y0,y1,x0,15,3);
        } else {
            Lcd_Draw_VLine(y0,y1,x0,15,1);
        }
        x0 += sizeofCell;
    }
    //Muestra los titulos de los respectivos tiempos
    Lcd_DspAscII8x16(margenIzquierdoSudoku+210,margenSuperiorSudoku+4,BLACK,"T. total:");
    Lcd_DspAscII8x16(margenIzquierdoSudoku+210,margenSuperiorSudoku+50,BLACK,"T. calculo:");
}

```

```

//Se completan las celdas con los datos correspondientes
escribirDatos();
//
if(idSiguientePantalla == pantallaVolverAJugar){
    actualizarTiempoTotalEnPantalla();
}
//Se actualiza la pantalla con la nueva informacion
Lcd_Dma_Trans();
}

/**
 * Escribe en las celdas la informaci3n que corresponda en ese momento,
 * haciendo que su color varie segun si son pista, errores, o datos
 * que ha puesto el usuario. Tambi3n actualiza el tiempo de calculo en pantalla.
 */
void escribirDatos(){
    int i,j;
    INT8U* numero = "";
    for(i=0; i<9; i++){
        for(j=0; j<9; j++){
            int valor = obtenerValor(i,j);
            numero[0]=valor+'0';
            int x0 = margenIzquierdoSudoku+(j)*sizeofCell;
            int y0 = margenSuperiorSudoku+(i)*sizeofCell;
            if(valor==0){
                //No hay valor, poner candidatos
                colocarCandidatos(i,j);
            } else if(esPista(i,j)){
                //Si es pista, se dibuja de color gris
                if(esError(i,j)) {
                    //En caso de ser error y pista, se pinta de negro la celda
                    LcdClrRect(x0, y0+4, x0+sizeofCell, y0+4+sizeofCell, BLACK);
                    Lcd_DspAscII8x16(8+x0,8+y0,DARKGRAY, numero);
                } else {
                    Lcd_DspAscII8x16(8+x0,8+y0,DARKGRAY, numero);
                }
            } else if(esError(i,j)){
                //Se pinta de negro la celda
                LcdClrRect(x0, y0+4, x0+sizeofCell, y0+4+sizeofCell, BLACK);
                Lcd_DspAscII8x16(8+x0,8+y0,WHITE, numero);
            } else{ //Se dibuja un valor normal
                Lcd_DspAscII8x16(8+x0,8+y0,BLACK, numero);
            }
        }
    }
    //Actualiza el tiempo del ultimo calculo en pantalla
    Lcd_DspAscII8x16(margenIzquierdoSudoku+215,margenSuperiorSudoku+65,BLACK,tiempoCalculoEnPantalla);
}

```

```

/**
 * Funci3n que informa por pantalla de los candidatos de una celda concreta
 * cuya posici3n viene definida por parametros.
 * Para mostrar la informaci3n, se divide la celda en 9 partes, pintando
 * de negro las que correspondan de manera n3merica y ascendente a un candidato
 * Ejemplo:      (Siendo N negro, y B blanco)
 * N N B
 * B B B
 * B N B
 * Los candidatos del ejemplo serian: 1, 2 y 8.
 */
void colocarCandidatos(int x, int y) {
    int i, j;
    int candidato = 1;
    for(i=0; i<3; i++) {
        for(j=0; j<3; j++) {
            /* Tomando como punto de referencia la esquina superior izquierda del sudoku
             * se desplaza hacia abajo y hacia la derecha tantas veces como la posici3n
             * x e y tengan, avanzando en cada caso el tama3o de una celda.
             * Despu3s hace lo mismo dentro de la celda, moviendose por las 9 partes que tiene.
             */
            int x0 = 1+margenIzquierdoSudoku+(y)*sizeofCell+j*sizeofCell/3;
            int y0 = 1+margenSuperiorSudoku+(x)*sizeofCell+i*sizeofCell/3;
            /* En caso de que el posible candidato que se este revisando, sea finalmente candidato,
             * se pinta un cuadrado negro en su posici3n correspondiente y calculada anteriormente.
             */
            if(esCandidato(x, y, candidato)) {
                LcdClrRect(x0, y0+4, x0+(sizeofCell/3), y0+4+(sizeofCell/3), BLACK);
            }
            candidato++; // Se revisa el siguiente posible candidato
        }
    }
}

/**
 * Muestra por pantalla un mensaje preguntando al usuario si desea finalizar,
 * y lo que debe pulsar segun su respuesta.
 */
void mostrarConfirmacionFinal(){
    /* clear screen */
    Lcd_Clr();
    Lcd_Active_Clr();

    #ifdef Eng_v
    Lcd_DspAscII8x16(10,0,DARKGRAY,"Embest S3CEV40 ");
    #else
    #endif
    Lcd_DspAscII8x16(0,45,BLACK,"Esta seguro de que desea finalizar?");
    Lcd_DspAscII8x16(30,75,BLACK,"Boton izquierdo - Cancelar");
    Lcd_DspAscII8x16(30,95,BLACK,"Boton derecho - Aceptar");
    //Se actualiza la pantalla con la nueva informacion
    Lcd_Dma_Trans();
}

```

```

/**
 * Muestra la pantalla de instrucciones con lo basico que hay que saber
 */
void mostrarInstrucciones(){
    /* initial LCD controller */
    Lcd_Init();
    /* clear screen */
    Lcd_Clr();
    Lcd_Active_Clr();

    #ifdef Eng_v
    Lcd_DspAscII8x16(10,0,DARKGRAY,"Embest S3CEV40 ");
    #else
    #endif
    Lcd_DspAscII8x16(0,25,BLACK,"Instrucciones para jugar:");
    Lcd_DspAscII8x16(0,55,BLACK,"Utilice el boton izquierdo para");
    Lcd_DspAscII8x16(0,70,BLACK,"seleccionar filas, columnas y valores");
    Lcd_DspAscII8x16(0,105,BLACK,"Utilice el boton derecho para");
    Lcd_DspAscII8x16(0,120,BLACK,"confirmar los datos introducidos");
    Lcd_Draw_HLine(0,320,140,15,3);
    Lcd_DspAscII8x16(0,175,BLACK,"Pulse un boton para jugar");
    //Se actualiza la pantalla con la nueva informacion
    Lcd_Dma_Trans();
}

/**
 * Funcion que establece el identificador de la siguiente pantalla a mostrar
 * cuando se actualice.
 */
void setPantalla(int idPantalla){
    idSiguientePantalla = idPantalla;
}

/**
 * Establece el tiempo de calculo que se debe mostrar al actualizar la pantalla
 */
void setTiempoCalculo(uint32_t tiempo){
    itoa(tiempo, tiempoCalculoEnPantalla);
}

```

```

/**
 * Muestra por pantalla el tiempo de juego que el usuario lleva en un momento concreto.
 */
void actualizarTiempoTotalEnPantalla() {
    /* Se comprueba si la pantalla que hay en este momento no es la de confirmacion,
    * ya que en ese caso el tiempo de juego sigue avanzando, pero no debe mostrarse.
    */
    if (idSiguientePantalla != pantallaConfirmacionFinal) {
        // Borra el contenido del area en la que se encuentra el tiempo de juego
        LcdClrRect(margenIzquierdoSudoku+215, margenSuperiorSudoku+18, LCD_XSIZE, margenSuperiorSudoku+40, WHITE);
        // Lee el tiempo que el usuario lleva, y lo guarda en formato de texto
        itoa(leerTiempoTimer4(), tiempoTotalEnPantalla);
        // Muestra el tiempo que acaba de leer.
        Lcd_DspAscII8x16(margenIzquierdoSudoku+215, margenSuperiorSudoku+20, BLACK, tiempoTotalEnPantalla);
        //Se actualiza la pantalla con la nueva informacion
        Lcd_Dma_Trans();
    }
}

```

8.4. GESTORREBOTES.C

```
/* *****
* Fichero:      gestorRebotes.c
* Autores:     Alejandro Guiu Perez - 680669 y Alvaro Juan Ciriaco - 682531
* Descrip:     funciones de control del timer0 del s3c44b0x
* ***** */

/*--- ficheros de cabecera ---*/
#include "44b.h"
#include "44blib.h"
#include "gestorSudoku.h"

enum {botonIzquierdo = 0, botonDerecho = 1};
enum {botonIzqMascara = 6, botonDerMascara = 7};
enum {SinPulsar = 0, Pulsado = 1, Soltando = 2};
enum {false = 0, true = 1};

/*--- variables globales ---*/
int hallegadoA500ms = false;           //vale true si se ha mantenido el botón izquierdo 500ms
int veces10ms = 0;                     //Lleva la cuenta de las veces que han pasado 10ms
int estado = SinPulsar;                 //Estado actual de la maquina de estados del gestor
int botonPulsado;                       //Guarda el botón pulsado en la máscara
int tiempoAlPulsar_Soltar = 10000;     //Equivale a 250 ms
int tiempoSiguePulsado = 1290;         //Equivale a 10 ms

/*--- declaracion de funciones ---*/
// Para definir la rutina de captura de la interrupción del timer.
void timer0_ISR(void) __attribute__((interrupt("IRQ")));
void timer0_init(int);

/*--- codigo de las funciones ---*/
// Rutina de captura de una interrupción del timer.
void timer0_ISR(void)
{
    /* Si el botón se estaba soltando, actualiza el estado como sinPulsar
     * y reinicia los valores del timer además de los botones.
     * Si esta pulsado, solicita al timer0 que le avise en [tiempoSiguePulsado] ms
     */
    if(estado == Soltando){             //estado2
        estado = SinPulsar;             //va a estado0
        //Se reinician las variables empleadas para el control de la opción de
        //mantener pulsado los botones
        hallegadoA500ms = false;
        veces10ms = 0;
        //Se reinician los botones
        rINTMSK    &= ~(BIT_EINT4567); // Enmascara todas las líneas excepto eint4567.
        /* Por precaucion, se vuelven a borrar los bits de INTPND y EXTINTPND */
        rI_ISPC    |= (BIT_EINT4567);
        rEXTINTPND = 0xf;
    }
    //Si se ha soltado el botón
    /*else estado es Pulsado*/
    else if(rPDATG & (1<<botonPulsado)){
        timer0_init(tiempoAlPulsar_Soltar);
        estado = Soltando;
    }
}
```

```

//Si se mantiene pulsado
else if(!(rPDATG & (1<< botonPulsado))){ //estado1
//Si se pulsa el botón de selección y se esta manteniendo
if(botonPulsado==botonIzqMascara){
    veces10ms++;
    //Si lleva 500ms así se informa al gestor del sudoku
    if(!hallegadoA500ms && veces10ms==50){
        hallegadoA500ms = true;
        gestionarSudoku(botonIzquierdo);
        veces10ms = 0;
    }
    //Si ha alcanzado los 500 ms, aumenta cada 300ms
    else if(hallegadoA500ms && veces10ms==30){
        veces10ms=0;
        gestionarSudoku(botonIzquierdo);
    }
}
timer0_init(tiempoSiguePulsado);
}
/* borrar bit en I_ISPC para desactivar la solicitud de interrupción*/
rI_ISPC |= BIT_TIMER0;
}

/**
 * Funcion encargada de guardar una referencia al boton que esta
 * pulsado (izquierdo, derecho...) e inicializa el timer correspondiente.
 */
void gestionarRebotesPulsacion(int boton){
    estado = Pulsado; //Avanzamos de estado
    botonPulsado = boton; //Indicamos que boton esta pulsado
    timer0_init(tiempoAlPulsar_Soltar); //Inicializamos el timer
}

// Inicializa los valores del timer.
void timer0_init(int tiempo)
{
    /* Configuración controlador de interrupciones */
    rINTMSK &= ~(BIT_TIMER0); // Emascara el bit del timer 0.

    /* Establece la rutina de servicio para TIMER0 */
    pISR_TIMER0 = (unsigned) timer0_ISR;

    /* Configura el Timer0 */
    rTCFG0 |= 0x000000FF; // ajusta el preescalado a 255
    rTCFG1 &= 0xFFFFFFF0; // selecciona la entrada del mux que proporciona el reloj. La 00 corresponde a un divisor de 1/2.
    rCNTB0 = tiempo; // valor inicial de cuenta (la cuenta es descendente)
    rCMPB0 = 0; // valor de comparación
    /* establecer update=manual (bit 1) + inverter=on (¿? será inverter off un cero en el bit 2 pone el inverter en off)*/
    rTCON = (rTCON & 0xFFFFFFF0) | 0x2;
    /* iniciar timer (bit 0) con auto-reload (bit 3)*/
    rTCON = (rTCON & 0xFFFFFFF0) | 0x9;
}

```

8.5. GESTORSUDOKU.C

```
/* ****
 * Fichero:      gestorSudoku.c
 * Autores:     Alejandro Guiu Perez - 680669 y Alvaro Juan Ciriaco - 682531
 * Descrip:     gestor encargado del funcionamiento del sudoku
 * **** */

/* --- ficheros de cabecera --- */
#include "8led.h"
#include "sudoku_2015.h"
#include "gestorPantalla.h"
#include "timer2.h"
#include "timer4.h"

enum {sinEmpezar = 0, mostrarF = 1, elegirFila = 2, mostrarC = 3,
      elegirColumna = 4, mostrarD = 5, elegirValorDato = 6, confirmarFinal=7};
enum {botonIzquierdo = 0, botonDerecho = 1};
enum {pantallaJuego = 0, pantallaConfirmacionFinal = 1, pantallaVolverAJugar = 2,
      pantallaInstrucciones = 3};

/* --- variables globales --- */
int estadoSudoku = sinEmpezar; // Variable que marca estado actual del sudoku
int datoMostradoPorPantalla = 0; // Indica el numero actual que se muestra por el 8Led
int fila, columna; // Indican fila y columna del dato a introducir o modificar
int pintar = 0; // Indica si hay que pintar la pantalla

/**
 * Gestiona el juego del sudoku, actualizando los estados
 * y mostrando las salidas correspondientes
 * por la pantalla 8Led.
 */
void gestionarSudoku(int botonPulsado) {
    /* ESTADOSUDOKU = SIN EMPEZAR */
    if (estadoSudoku == sinEmpezar) {
        timer4_reset(); // Inicializamos timer que indica el tiempo de juego total
        timer4_init();
        estadoSudoku = mostrarF;
        datoMostradoPorPantalla = 15; // Mostramos una F por 8Led
        iniciarArmThumb(); // Iniciamos algoritmo (ARM-THUMB)
        setPantalla(pantallaJuego); // Indicamos la pantalla a dibujar
        refrescarPantalla(1); // Refrescamos la pantalla
    }

    /* ESTADOSUDOKU = MOSTRAR F */
    else if (estadoSudoku == mostrarF && botonPulsado == botonIzquierdo) {
        datoMostradoPorPantalla = 1; // Mostramos en 8Led un 1
        estadoSudoku = elegirFila; // Avanzamos de estado
    }
}
```



```

/* ESTADOSUDOKU = ELEGIR FILA */
else if (estadoSudoku == elegirFila) {
    if (botonPulsado == botonIzquierdo) {
        datoMostradoPorPantalla++;
        if (datoMostradoPorPantalla == 11) {
            datoMostradoPorPantalla = 1;
        }
    } else if (botonPulsado == botonDerecho) {
        fila = datoMostradoPorPantalla;
        // Si seleccionamos una "A" vamos a la pantalla de confirmarFinal
        if (fila == 10) {
            estadoSudoku = confirmarFinal;
            setPantalla(pantallaConfirmacionFinal);
            refrescarPantalla(1);
        } else {
            // Avanzamos siguiente estado (mostrar C)
            estadoSudoku = mostrarC;
            datoMostradoPorPantalla = 12;
        }
    }
}

/* ESTADOSUDOKU = MOSTRAR C */
else if (estadoSudoku == mostrarC && botonPulsado == botonIzquierdo) {
    datoMostradoPorPantalla = 1; // Mostramos un 1 por el 8Led
    estadoSudoku = elegirColumna;
}

/* ESTADOSUDOKU = ELEGIR COLUMNA */
else if (estadoSudoku == elegirColumna) {
    if (botonPulsado == botonIzquierdo) {
        datoMostradoPorPantalla++;
        if (datoMostradoPorPantalla == 10) {
            datoMostradoPorPantalla = 1;
        }
    } else if (botonPulsado == botonDerecho) {
        // Avanzamos siguiente estado (mostrar D)
        columna = datoMostradoPorPantalla;
        estadoSudoku = mostrarD;
        datoMostradoPorPantalla = 13;
    }
}

/* ESTADOSUDOKU = MOSTRAR D */
else if (estadoSudoku == mostrarD && botonPulsado == botonIzquierdo) {
    datoMostradoPorPantalla = 1;
    estadoSudoku = elegirValorDato;
}

```

```

/* ESTADOSUDOKU = ELEGIR VALOR DATO */
else if (estadoSudoku == elegirValorDato) {
    if (botonPulsado == botonIzquierdo) {
        datoMostradoPorPantalla = (datoMostradoPorPantalla+1)%10;
    } else if (botonPulsado == botonDerecho) {
        // Insertamos en el sudoku el valor dado por el usuario
        int celdasVacias = insertValor(fila, columna, datoMostradoPorPantalla);
        if (celdasVacias == 0) {
            // Si ya no hay celdas vacías es que el sudoku se ha realizado con éxito
            estadoSudoku = sinEmpezar;
            // Reiniciamos maquina de estados y mostramos por pantalla los resultados
            setPantalla(pantallaVolverAJugar);
            timer4_stop();
        } else {
            // Si el sudoku no ha acabado volvemos a realizar el mismo proceso
            estadoSudoku = mostrarF;
            datoMostradoPorPantalla = 15;
        }
        refrescarPantalla(1);
    }
}

/* ESTADOSUDOKU = CONFIRMAR EL FINAL DEL JUEGO */
else if (estadoSudoku == confirmarFinal){
    if (botonPulsado == botonIzquierdo) {
        // Cancelamos la finalización del juego
        estadoSudoku = mostrarF;
        datoMostradoPorPantalla = 15; // Mostramos una F por pantalla
        setPantalla(pantallaJuego);
    } else if (botonPulsado == botonDerecho) {
        // Terminamos de jugar y se muestra por pantalla el resultado actual
        timer4_stop();
        estadoSudoku = sinEmpezar;
        setPantalla(pantallaVolverAJugar);
    }
    refrescarPantalla(1);
}

// Se muestra por el 8Led el valor de la variable
D8Led_symbol(datoMostradoPorPantalla);
}

/**
 * Devuelve el valor de la variable [pintar]
 */
int hayQueRefrescar(){
    return pintar;
}
}

```

8.6. LISTACIRCULAR.C

```

/*****
 * Fichero: listaCircularEstatica.c
 * Autores: Alejandro Guiu Perez - 680669 y Alvaro Juan Ciriaco - 682531
 * Descrip: lista circular estatica para la pila
 * Version:
 *****/
#include "44b1ib.h"
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>

//Creacion del tipo de dato tipoNodo (representa el elemento a apilar)
typedef struct{
    int ID_evento;           // Identificador del evento.
    int auxData;            // Datos auxiliares
    uint32_t tiempo;        // Momento de el que tuvo lugar la interrupciÃ³n
} tipoNodo;

tipoNodo* lista = (tipoNodo *) 0xc7fedfc; // Puntero a la posici3n inicial de la lista. (260 bytes)

void anadirElemento(int identificador, int dato, int tiempo);

// Indice que marca el ultimo elemento de la pila
static int indice = 0;

// Operacion que se encarga de aÃ±adir un nuevo elemento a la lista
void anadirElemento(int identificador, int dato, int tiempo){
    lista[indice].ID_evento = identificador;
    lista[indice].auxData = dato;
    lista[indice].tiempo = tiempo;
    indice++;

    //Resetea el indice de la lista si esta llega al tope
    if((int)&lista[indice]>0xc7fef00){
        indice = 0;
    }
}
}

```

8.7. MAIN.C

```
/* *****
* Fichero: main.c
* Autores: Alejandro Guiu Perez - 680669 y Alvaro Juan Ciriaco - 682531
* Descrip: punto de entrada de C
* Version: <P4-ARM.timer-leds>
* ***** */

/*--- ficheros de cabecera ---*/
#include "44blib.h"
#include "44b.h"
#include "stdio.h"
#include "gestorSudoku.h"
#include "gestorPantalla.h"
#include <inttypes.h>
#include "timer2.h"
#include "timer4.h"

enum {pantallaJuego = 0, pantallaConfirmacionFinal = 1, pantallaVolverAJugar = 2,
      pantallaInstrucciones = 3};

typedef uint16_t CELDA;

/*--- funciones externas ---*/
extern void Eint4567_init();
extern void D8Led_init();
extern void init_excep();

/*--- declaracion de funciones ---*/
void Main(void);

/*--- codigo de funciones ---*/
void Main(void)
{
    /* Inicializa controladores */
    sys_init(); // Inicializacion de la placa, interrupciones y puertos.
    Eint4567_init(); // Inicializamos los pulsadores. Cada vez que se pulse se v
    D8Led_init(); // Inicializamos el 8led.
    init_excep(); // Inicializamos tratamiento de excepciones
    setPantalla(pantallaInstrucciones); // Mostrar la pantalla de instrucciones
    actualizarPantalla(); // Actualizar la pantalla
    while(1) {
        // Si hay que actualizar el tiempo de juego en la pantalla
        if(hayQueActualizarTiempo()) {
            // Dibujamos el nuevo tiempo de juego
            desactivarActualizacion();
            actualizarTiempoTotalEnPantalla();
        }
        // Si hay que pintar una nueva "pantalla de juego" en la pantalla
        if(hayQueRefrescar()){
            //Si es asi, desactiva la peticion de pintar, y muestra por
            //pantalla el sudoku actualizado.
            refrescarPantalla(0);
            actualizarPantalla();
        }
    }
}
```

8.8. PILADEPURACION.C

```
/* *****
* Fichero: pilaDepuracion.c
* Autores: Alejandro Guiu Perez - 680669 y Alvaro Juan Ciriaco - 682531
* Descrip: implementa la operacion encargada de añadir un nuevo registro de excepcion a la pila
* Version:
* ***** */

#include "44blib.h"
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "timer4.h"

/*--- declaracion de funciones ---*/
void push_debug(int ID_evento, int auxData);

// Funcion encargada de apilar un nuevo elemento en la pilaDepuracion
void push_debug(int ID_evento, int auxData){
    uint32_t tiempo = (leerTiempoTimer4()); // Tiempo en el que se ha producido la interrupción
    anadirElemento(ID_evento, auxData, tiempo);
}
```

8.9. TIMER2.C

```
*****
* Fichero: timer2.c
* Autores: Alejandro Gulu Perez - 680669 y Alvaro Juan Ciriaco - 682531
* Descrip: funciones de control del timer2
* Version:
*****/

/*--- ficheros de cabecera ---*/
#include "44b.h"
#include "44blib.h"
#include <inttypes.h>
#include "gestorPantalla.h"

/*--- variables globales ---*/
uint32_t timer2_num_int = 0; // Número de interrupciones generadas por el timer 2.

/*--- declaracion de funciones ---*/
void timer_ISR2(void) __attribute__((interrupt("IRQ")));
void Timer2_Inicializar();
void Timer2_Empezar();
uint32_t Timer2_Leer();

/*--- codigo de las funciones ---*/
void timer_ISR2(void)
{
    timer2_num_int++; //Aumentamos el número de interrupciones del timer2
    /* borrar bit en I_ISPC para desactivar la solicitud de interrupción*/
    rI_ISPC |= BIT_TIMER2;
}

void Timer2_Inicializar()
{
    /* Configuración controlador de interrupciones */
    // Enmascara el bit del timer 2.
    rINTMSK &= ~(BIT_TIMER2);

    /* Establece la rutina de servicio para TIMER2 */
    pISR_TIMER2 = (unsigned) timer_ISR2;

    /* Configura el Timer2 */
    rTCFG0 &= 0xFFFF00FF; // Preescalado a 0
    rTCFG1 &= 0xFFFF00FF; // Entrada del mux. 00 divisor 1/2
    rTCNTB2 = 65535; // Valor inicial de la cuenta del timer (valor máximo como se pide enunciado)
    rTCMPB2 = 0; // Valor con el que se comparará la cuenta del timer2
    /* Activamos los bits para iniciar la cuenta (12 y 13) */
    rTCON = (rTCON & 0xFFFF0FFF) | 0x2000; //Activamos manual update bit.
    rTCON = (rTCON & 0xFFFF0FFF) | 0x9000; //Activamos start/stop y desactivamos manual update bit a la vez.
}

// Función encargada de reiniciar la cuenta de interrupciones del timer2
void Timer2_Empezar()
{
    timer2_num_int = 0;
}

// Devuelve el tiempo actual del timer 2 (tratando las condiciones de carrera).
uint32_t Timer2_Leer()
{
    uint32_t interrupcionesAntes, interrupcionesDesp, interrupcionesCuenta;
    // Leemos 2 veces las interrupciones totales del timer más una vez la cuenta actual
    interrupcionesAntes = timer2_num_int;
    uint32_t cuentaActual = rTCNT02;
    interrupcionesDesp = timer2_num_int;
    // Si las interrupciones coinciden, no ha habido problema al leer el tiempo
    if (interrupcionesAntes == interrupcionesDesp) {
        interrupcionesCuenta = interrupcionesDesp;
    } else {
        // Si las interrupciones no coinciden en las 2 mediciones
        interrupcionesCuenta = cuentaActual >= 65535/2 ?
            interrupcionesAntes : interrupcionesDesp;
    }
    // Devolvemos el tiempo actual que lleva el timer 2 en microsegundos
    return (interrupcionesCuenta*(65535/32)+((65535-cuentaActual)/32));
}
```

8.10. TIMER4.C

```
/* *****
 * Fichero: timer2.c
 * Autores: Alejandro Guio Perez - 680669 y Alvaro Juan Ciriaco - 682531
 * Descrip: funciones de control del timer4
 * Version:
 * ***** */

/*--- ficheros de cabecera ---*/
#include "44b.h"
#include "44blib.h"
#include <inttypes.h>
#include "gestorPantalla.h"

/*--- declaracion de funciones ---*/
void timer4_ISR(void) __attribute__((interrupt("IRQ")));
void timer4_init(void);
void timer4_stop(void);

/*--- declaracion de variables ---*/
volatile int actualizarTiempo = 0; // Indica si hay que actualizar el tiempo de la pantalla
volatile uint32_t tiempoTotal=0; // lleva la cuenta de las veces que ha interrumpido

void timer4_ISR(void)
{
    tiempoTotal++; // Aumentamos el numero de interrupciones del timer4
    actualizarTiempo = 1; // Marcamos que hay que actualizar el tiempo dibujado en la pantalla
    rI_ISPC |= BIT_TIMER4; // limpiar bit de pendiente del timer4
}

/**
 * Funcion encargada de parar la gestion de interrupciones del timer
 * con la finalidad de dejar el timer bloqueado con la varibale tiempoTotal
 * sin resetear, para asi poder mostrarla de forma indefinida en la pantalla final del sudoku
 */
void timer4_stop(void){
    rINTMSK |= BIT_TIMER4; // Deshabilitamos interrupciones de timer4
    rTCON &= ~(0x100000); // Paramos el reloj del timer
}

/* Funcion encargada de resetear la cuenta del timer 4 */
void timer4_reset(void) {
    tiempoTotal = 0;
}
```

```
void timer4_init(void){

    // Configuracion del timer4
    rINTMSK &= ~(BIT_TIMER4);
    pISR_TIMER4=(unsigned)timer4_ISR;

    rTCFG0 |= ~(0x00FFFF); // Marcamos el preescalado a 255
    rTCFG1 &= 0xFFFFFFF; // Marcamos el divisor a 1/4
    rTCFG1 |= 0x10000;
    rTCNTB4 = 64516; // Valor inicial de cuenta
    rTCMPB4 = 0; // Valor del dato a comparar
    rTCON |= 0x200000; // Activamos manual update bit.

    rTCON |= 0x900000; // Activa auto-reload
    rTCON &= ~(0x200000); // Desactiva manual update bit.
}

/* Devuelve el valor de la variable [tiempoTotal] */
uint32_t leerTiempoTimer4() {
    return tiempoTotal;
}

/* Devuelve el valor de la variable [actualizarTiempo] */
int hayQueActualizarTiempo() {
    return actualizarTiempo;
}

/* Desactiva (pone a 0) la variable [actualizarTiempo] */
void desactivarActualizacion() {
    actualizarTiempo = 0;
}
```

9. PROBLEMAS ENCONTRADOS:

Durante las sesiones de prácticas se han encontrado varios problemas, algunos más costosos de solucionar que otros. Durante el desarrollo de la práctica 2 la programación del primer Timer2 trajo varios problemas. Aún no se dominaba y se entendía el funcionamiento de la placa y los manuales no dejaban claro cómo configurar el timer para que funcionase de la forma correcta. El principal problema fue encontrar y activar los bits necesarios para la inicialización del mismo.

La eliminación de los rebotes en los pulsadores también presentó algunos problemas. En un principio no se sabía cómo indicarle al programa qué botón había pulsado el usuario y cuando se había soltado o seguía pulsado. Para solucionar el problema se buscó información en los manuales y se preguntaron algunas dudas a los profesores.

Respecto a la práctica 3, el problema principal fue entender cómo funcionaban las operaciones que dibujaban en la pantalla de la placa. Una vez entendidas, fue más o menos sencillo el diseñar las distintas pantallas del sudoku. Otro problema fue el funcionamiento del timer2 que se utiliza para calcular el tiempo de ejecución de la operación de recalcular (como se pide en el pdf). Una vez programado daba errores, como que mostraba el tiempo pero negativo. Esto era porque no estaban gestionada las condiciones de carrera. Después de preguntar a varios profesores y con la ayuda de otros compañeros se consiguió solucionar el problema y se dejó el timer 2 funcionando correctamente.

Por último, el principal problema de las prácticas fue la escasez de placas. A pesar de asistir a todas las horas que correspondían se necesitó una gran cantidad de horas extra para poder terminar las prácticas dentro de plazo y con todo lo que se pedía funcionando de forma correcta. Al final se tenía que ir la mayoría de días de la semana por la mañana a la universidad para coger placa o simplemente para entrar en el sorteo de alguna para los días posteriores.

10. CONCLUSIONES:

Durante las distintas sesiones de prácticas se han ido cumpliendo todos los objetivos previos marcados en los guiones de las prácticas. Con el paso de las sesiones se ha adquirido un mayor conocimiento de la estructura del procesador, se ha depurado código con varias fuentes de interrupción activas, se ha aprendido a desarrollar en C rutinas de tratamiento de interrupción, se han programado temporizadores internos de la placa y teclado y se ha utilizado la pantalla LCD de la placa para visualizar el tablero de juego.

Para el desarrollo de las prácticas se ha elegido el código ARM-THUMB para ejecutar la operación de recalcular candidatos. Se ha elegido este porque era el código más complejo y con el que más problemas se tuvieron en la primera práctica. Además, este es uno de los códigos más rápidos y con menos instrucciones, tal y como se presentó en la memoria de la primera práctica.

TIEMPO (MS)	INSTRUCCIONES EJECUTADAS
2,2	21.095

La optimización y presentación del código de la práctica 2 ha sido mejorado de forma considerable tras la sesión de presentación de dicha práctica. Se han modificado tanto la máquina de estado de los pulsadores como la máquina de estados del sudoku para mejorar el entendimiento de ambas. Para ello se crearon dos ficheros denominados “gestorSudoku.c” (encargado de gestionar los estados del sudoku) y “gestorRebotes.c” (encargado de gestionar los rebotes de los distintos botones de la placa), se modificaron los nombres de las variables y de los estados (para facilitar el entendimiento del código) y se eliminaron variables innecesarias. Además, se modificó el archivo “button.c” simplificando la rutina de interrupción de los botones tal y como se pedía en el guion de la práctica.

En general, el desarrollo de las prácticas de la asignatura ha supuesto un trabajo complejo y con muchas horas de dedicación, sin embargo son unas prácticas completamente distintas a las realizadas en cualquier otra asignatura y esto supone una motivación especial para llevarlas a cabo. A pesar de haber trabajado anteriormente con el lenguaje ensamblador y en C, en ninguna de las anteriores asignaturas se había podido poner en práctica y ver unos resultados “visibles”. Es gratificante ver como con el paso de las prácticas el sudoku iba funcionando y mejorando, hasta llegar al punto final, cuando puedes jugar al sudoku directamente en una placa independiente al ordenador.

Sin embargo, muchas veces las prácticas han supuesto un gran problema, sobre todo por algunos fallos ajenos a nosotros (problemas con las placas,..) y en especial a la poca disposición de las placas para poder realizar pruebas e ir avanzando con las prácticas. Aun habiendo seguido las instrucciones de los profesores (preparar el código en casa e ir ya a las sesiones de prácticas con el código desarrollado) la práctica 2 se hizo bastante dura.