Universidad Zaragoza

Práctica 2 y 3 Proyecto Hardware

Jorge Sanz Alcaine 680182

Resumen

Para la implementación del sudoku sobre la placa se han creado librerías con las que medir el tiempo de forma precisa, gestionar excepciones, escribir en memoria datos de depuración, y conseguir el comportamiento deseado en los botones.

Para medir el tiempo se han habilitado las interrupciones en algunos timers. El timer 0 se ha configurado para que tenga una frecuencia menor a la del resto, por lo que es menos preciso. Cada una de las librerías asociadas a los timers contienen los metodos empezar y leer. El método leer devuelve el tiempo en milisegundos transcurrido desde la ultima vez que se llamó al método empezar.

Se han gestionado las excepciones de software, undefined, dAbort y pAbort de forma que cuando se producen se muestran los códigos 1, 2, 3 y 4 respectivamente parpadeando en el 8led.

Para poder ver el valor de algunos datos durante la depuración, se ha desarrollado una pila en la que almacenar esos datos y así poder verlos en memoria. Esa pila tiene asociado el timer 0 y cada vez que se apila un dato se apila también el instante en el que se apiló. La pila tiene una capacidad de 256 Bytes y cuando se supera esa capacidad se sobrescribe el principio de la pila.

Se ha utilizado una librería que configura el comportamiento de los botones. Cuando se pulsa un botón, su interrupción viene acompañada de otras llamadas rebotes. Los rebotes no hay que tenerlos en cuenta y por ello se ignoran las interrupciones de botón generadas justo después de pulsar el botón y antes de soltarlo. Cada uno de los botones tiene un comportamiento deferente, el izquierdo sirve para aumentar una cuenta hasta que llega a cierto número y se reinicia. El derecho sirve para confirmar la cuenta marcada con el botón izquierdo.

Se ha utilizado otra pantalla con la que interaccionar con la pantalla lcd. Con ella se muestra por la pantalla el sudoku, las instrucciones y el tiempo de juego y calculo.

Se ha cargado el código en la memoria flash para que cuando el usuario encienda la placa se ejecute el juego.

Introducción

El objetivo de la práctica es implementar un programa sudoku a partir del código de la practica anterior sobre una placa arm real. El programa debe cargarse automáticamente al encender la placa y dispone de mecanismo de E/S con los que interaccionar con el usuario.

Tiempos obtenidos

Se han utilizado tres librerías diferentes para los tiempos, cada gestiona uno de los timers utilizados, en este caso 0, 2 y 3. Todas la librerías tienen la misma estructura, la única diferencia es la frecuencia con la que ocurren las interrupciones. Todas disponen de un atributo llamado timer0_num_int que se incrementa en cada interrupción. Este atributo está configurado para resetearse al llegar a una cifra indicada en el método init de la librería. Disponen también de otros dos métodos más, uno de ellos resetea el contador anterior y el otro interpreta su valor en función de la frecuencia de las interrupciones para devolver los milisegundos transcurridos.

Se ha utilizado otra librería que gestiona una pila de depuración, la librería dispone de un método llamado push_debug que apila tres atributos, los dos que le pasan por parametros y el tiempo que ha transcurrido desde el inicio de la ejecución. Para apilar el tiempo transcurrido es necesario que un timer haya sido inicializado solo al comienzo de la ejecución. Para medir los tiempos de forma precisa se ha utilizado el timer 2.

Los resultados obtenidos en milisegundos son :

	C	ARM
C	32	31.9
ARM	5.7	13
THUMB	7.5	7.3

Las columna de la izquierda representa el código utilizado en sudoku_candidatos y la 1ª fila representa el código utilizado en sudoku_recalcular.

Estructura del proyecto

En el trabajo se ha desarrollado bastante código, pero este es fácilmente separable en módulos. Podemos distinguir fácilmente el modulo de gestión del tiempo, el de gestión de las interrupciones en los botones, el de gestión de las excepciones, el de la interfaz gráfica y el del sudoku.

Los tres primeros módulos se inicializan en el main, una vez hecho esto se lanza el método juegoSudoku que inicializa la interfaz gráfica y comienza el juego del sudoku.

Descripción de la interacción

El usuario puede comunicarse con el sistema por medio de los botones, el 8 led y la pantalla. Cuando se pulsa un botón, este genera una interrupción y esta es capturada para que el sistema responda en función del botón pulsado. Sin embargo, sucede que tras la primera interrupción aparecen otras no deseadas llamadas rebotes y lo mismo al soltarlo. Para evitar estos rebotes se han introducido tiempos en los que se deshabilitan las interrupciones de botones para así ignorar los rebotes, una vez desaparecen los rebotes se trata el comportamiento de los botones.

Cuando el usuario enciende la placa se muestra “Pulsa un botón para empezar”. Cuando el usuario pulsa un botón aparece el sudoku, los tiempos y el texto “Introduce la fila A para acabar”. El usuario elige la fila, la columna y el valor con los botones y el sudoku se actualiza. Cuando el usuario introduce la fila A, vuelve a aparecer el texto “Pulsa un botón para empezar”. Los candidatos a cada celda aparecen como cuadrados más pequeños en la posición que le corresponde. Las celdas que aparecen como pista aparecen en negro mientras que las que introduce el usuario aparecen en gris. Si el usuario introduce un número que no está en los candidatos, las celdas en conflicto se invierten para que el usuario lo corrija.

Gestión de excepciones

Se han tratado cuatro excepciones diferentes, para ello se ha utilizado una librería que dispone de dos métodos. El primero de esos métodos se encarga de actualizar la dirección de salto en cada una de las excepciones al segundo método. El segundo método parpadea el 8 led indefinidamente mostrando un código de error en función de la excepción producida, 1 software, 2 undefined, 3 dAbort y 4 pAbort. Para conseguir distinguir las excepciones se comprueba el modo del CPSR puesto que cada una tiene un código diferente.

Carga del código en la memoria RAM

Se ha modificado el fichero 44binit.asm para que inicialice la memoria, esto antes se realizaba mediante un script en el inicio. También se ha incluido un código para que se copie el código de la ROM a la RAM al inicio de la ejecución y otro para que salte luego al main y así empiece el sudoku.

Una vez hecho esto es necesario escribir el fichero binario generado con todo el código anterior en la memoria ROM. Para ello, se han ejecutado los siguientes comandos:

Para generar el fichero binario

arm-none-eabi-objcopy -O binary practica3.elf practica3.bin

Para copiar el fichero binario a la memoria ROM

openocd-0.7.0.exe -f test/arm-fdi-ucm.cfg -c "program practica3.bin 0x00000000"

Código fuente

Control del tiempo

Existen tres ficheros para cada uno de los timers,cada uno de ellos tiene la estructura que tiene el código de abajo, el del timer 0. La diferencia entre ellos es que los otros timers utilizan un valor de preescalado de 0, por lo que la frecuencia de las interrupciones es mayor y a la hora de sacar los milisegundos cambian algunas cifras. Para configurarlo, se habilitan las interrupciones, se establece la rutina de servicio, que será el método timer_ISR, se ajusta el preescalado y la entrada del multiplexor, que controlan la frecuencia del timer, asignar los valores iniciales y de comparación en la cuenta y activar primero el flag de manual e invertir y luego el auto-reload e inicio. Una vez estos bits se activen, la variable timer0_num_int se reiniciará una vez llegue al valor de comparación.

```
void timer_ISR(void){
	timer0_num_int++;
	switch_leds = 1;
	rI_ISPC = rI_ISPC | BIT_TIMER0;
}
void timer_init(void){
	rINTMOD = 0x0; // Configura las linas como de tipo IRQ
	rINTCON = 0x1; // Habilita int. vectorizadas y la linea IRQ (FIQ no)
	rINTMSK = ~(BIT_GLOBAL | BIT_TIMER0);

	pISR_TIMER0 = (unsigned) timer_ISR;

	rTCFG0 = 255; // ajusta el preescalado
	rTCFG1 = 0x0;
	rTCNTB0 = 65535;// valor inicial de cuenta (la cuenta es descendente)
	rTCMPB0 = 12800;// valor de comparación
	/* update=manual (bit 1) + inverter=on */
	rTCON = 0x2;
	/* iniciar timer (bit 0) con auto-reload (bit 3)*/
	rTCON = 0x09;
}
void Timer0_Empezar(void)
{
	rTCNTO0 = rTCNTB0;
	timer0_num_int=0;
}
int Timer0_Leer(void){
	long cuenta=rTCNTB0;
	cuenta-=rTCNTO0;
	cuenta+=(timer0_num_int*rTCNTB0);
	cuenta=cuenta*8;
	return cuenta;
}
```

Comportamiento de los botones

Cuando aparece una interrupción, se comprueba cual es el botón que la ha causado. Si es el izquierdo se prepara la gestión de los rebotes deshabilitando las interrupciones de botón y cambiando estado a 1. Además, se aumenta el contador para luego mostrarlo por pantalla y bajar la interrupción. Si es el botón derecho, se activa confirmación siempre que se haya pulsado antes el otro botón y se prepara para la gestión de los rebotes.

```
/*--- variables globales ---*/
unsigned int int_count = 0;
int estado=0;
int boton=0;
int confirmacion=0;
int empezar=0;
int comparacion=10;

void Eint4567_ISR(void)
{
	/* Identificar la interrupcion (hay dos pulsadores)*/
	int which_int = rEXTINTPND;
	switch (which_int){
		case 4:
			boton=1;
			rINTMSK =BIT_EINT4567 | rINTMSK; //deshabilitar boton
			Timer3_Empezar();
			estado=1;
			int_count++; // incrementamos el contador
			if(int_count==comparacion){
				int_count=reset;
			}
			D8Led_symbol(int_count); // sacamos el valor por pantalla (módulo 16)
			break;
		case 8:
			if(boton==1)
				confirmacion=1;
			boton=2;
			rINTMSK =BIT_EINT4567 | rINTMSK; //deshabilitar boton
			Timer3_Empezar();
			estado=1;
			break;
		default:
			break;
	}
	if(empezar==0)
		boton=2;
	empezar=1;
	/* Finalizar ISR */
	rEXTINTPND = 0xf;				// borra los bits en EXTINTPND
	rI_ISPC   |= BIT_EINT4567;		// borra el bit pendiente en INTPND
}
```

Gestión de los rebotes

Los rebotes se han gestionado en la rutina de servicio del timer 3. Cuando se pulsa un botón se guarda 1 en la variable estado, se inicia el contador del timer 3y se deshabilitan las interrupciones. Pasados 100 milisegundos, ya no se producen rebotes, por lo que se pasa al estado dos, donde cada 10 milisegundos se comprueba si el botón sigue pulsado. Si el botón sigue pulsado pasados 500 milisegundos, se aumenta la cuenta en uno y se muestra por el 8 led cada 300 milisegundos hasta que se suelte el botón. Cuando se sueta el botón, se esperan otros 100 milisegundos para evitar los rebote, se borra el bit en I_ISCP para desactiva las solicitudes de interrupción de los rebotes y por último se guarda 0 en el estado y se habilitan las interrupciones.

```
void Timer3_ISR(void){
	timer3_num_int++;
	if(estado==1){
		if(Timer3_Leer()>100000){
			estado=2;
			Timer3_Empezar();
			comparacionMant=510000;
		}
	}else if(estado==2){
		if(Timer3_Leer()>10000){
			if(boton==1){
				if(rPDATG & 0x40){
					estado=3;
				}else{
					comparacionMant=comparacionMant-Timer3_Leer();
					if(mantener){
						if(Timer3_Leer()>comparacionMant){
							comparacionMant=310000;
							int_count++;
							if(int_count==comparacion)
								int_count=reset;
							D8Led_symbol(int_count);
						}
					}else{
						if(Timer3_Leer()>comparacionMant){
							comparacionMant=310000;
							mantener=1;
						}
					}
				}
				Timer3_Empezar();
			}else if(boton==2){
				if(rPDATG & 0x80)
					estado=3;
			}
		}
```

```
}else if(estado==3){
		if(Timer3_Leer()>100000){
			mantener=0;
			estado=0;
			rEXTINTPND = 0xf;
			rI_ISPC   |= BIT_EINT4567;
			rINTMSK =(~(BIT_EINT4567)) & rINTMSK;
		}
	}
	/* borrar bit en I_ISPC para desactivar la solicitud de interrupción*/
	rI_ISPC |= BIT_TIMER3; // BIT_TIMER2 está definido en 44b.h y pone un uno en el bit
11 que correponde al Timer2
}
```

Pila de depuración

Se ha creado una pila implementada como una lista circular, de forma que cuando esta alcanza cierto tamaño los datos se apilan al principio de la pila sobrescribiendo a los anteriores. La pila tiene asociada el timer0, de forma que cuando se apilan sus dos parámetros, también se apila el momento en el que se hizo (suponiendo que el timer se haya inicializado al comienzo de la ejecución).

```
int inicioPila=_ISR_STARTADDRESS-0xf00-256;
int limite=_ISR_STARTADDRESS-0xf00;
int *actual;


void pila_Init(){
	actual=_ISR_STARTADDRESS-0xf00-256;
}


void push_debug(int ID_evento, int auxData){
	if(actual+3>limite){
		actual=inicioPila;
	}
	*actual = Timer0_Leer();
	actual ++;

	*actual = ID_evento;
	actual++;

	*actual = auxData;
	actual++;
}
```

Gestion de excepciones

Para inicializar la gestion de excepciones, solo es necesario cambiar las direcciones de salto en la tabla de vectores para las cuatro interrupciones a la direccion de la rutina de tratamiento de excepcion, que en este caso es exception_ISR. La rutina de tratamiento de excepción no recive parametros, por lo que tiene que conprobar el modo de ejecución para saber cual el la excepcion provocada. Una vez sabe cual es la excepcion lo muestra en el 8led parpadeando.

```
void exception_ISR(void)
{
	uint32_t code=0;
	asm volatile("mrs %[code], CPSR\n\t"
		     "and %[code], %[code], #0x1F\n\t"
		     : [code] "+r" (code));
	switch(code){
		case 19:
			code=1;
			break;
		case 23:
			code=2;
			break;
		case 27:
			code=3;
			break;
		default:
			code=4;
		break;
	}
	while(1){
		Delay(4000);
		D8Led_symbol(16);
		Delay(4000);
		D8Led_symbol(code);
	}
}

void exception_init(void)
{
	pISR_UNDEF = (unsigned) exception_ISR;
	pISR_SWI = (unsigned) exception_ISR;
	pISR_PABORT = (unsigned) exception_ISR;
	pISR_DABORT = (unsigned) exception_ISR;
}
```

Función Main

Es el primero en ejecutarse y por ello su función es inicializar las librerías que se van a utilizar más adelante y lanzar el programa del sudoku.

```
void Main(void)
{
      /* Inicializa controladores */
      exception_init();
      sys_init
      timer_init();         Eint4567_init();

      D8Led_init(); // inicializamos el 8led
      Timer2_Inicializar(); /* Configura el Timer2 */
      Timer3_Inicializar(); /* Configura el Timer2 */
      pila_Init();

      juegoSudoku();
}
```

Sudoku

Variables y definiciones utilizadas

Aparte de las ya utilizadas en la práctica anterior, se han creado nuevas variables y definiciones para traducir los dígitos y otros caracteres como el cuadrado de forma fácil, para poder reiniciar el sudoku si el usuario termina o pulsa la tecla a y para medir los tiempos de juego.

```
typedef INT8U TEXTO[2];

const TEXTO traduccion[]=
     {"0","1","2","3","4","5","6","7","8","9",{0x7,'\0'}};

const char digitos[]={'0','1','2','3','4','5','6','7','8','9'};


const CELDA original[NUM_FILAS][NUM_COLUMNAS]=

     {0x9800,0x6800,0x0000,0x0000,0x0000,0x0000,0x7800,0x0000,0x8800
,0,0,0,0,0,0,0,
     0x8800,0x0000,0x0000,0x0000,0x0000,0x4800,0x3800,0x0000,0x0000,
0,0,0,0,0,0,0,
     0x1800,0x0000,0x0000,0x5800,0x0000,0x0000,0x0000,0x0000,0x0000,
0,0,0,0,0,0,0,
     0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x1800,0x7800,0x6800,
0,0,0,0,0,0,0,
     0x2800,0x0000,0x0000,0x0000,0x9800,0x3800,0x0000,0x0000,0x5800,
0,0,0,0,0,0,0,
     0x7800,0x0000,0x8800,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0,0,0,0,0,0,0,
     0x0000,0x0000,0x7800,0x0000,0x3800,0x2800,0x0000,0x4800,0x0000,
0,0,0,0,0,0,0,
     0x3800,0x8800,0x2800,0x1800,0x0000,0x5800,0x6800,0x0000,0x0000,
0,0,0,0,0,0,0,
     0x0000,0x4800,0x1800,0x0000,0x0000,0x9800,0x5800,0x2800,0x0000,
0,0,0,0,0,0,0};
int tiempoCalculos;
int tiempoTotal;
```

Pintar tablero

Para mostrar el tablero al usuario se utiliza la pantalla lcd. Cuando este método se usa la pantalla ya ha sido inicializado y el tablero ha sido recalculado, por ello, lo primero es limpiarla y dibujar la líneas del tablero en las posiciones indicadas. Una vez se han dibujado las líneas se comprueban las celdas para introducir los números. Si el valor es distinto de 0 y el bit de pista está activo se dibuja el número en negro, si es distinto de 0 pero no tiene el bit de pista se dibuja en gris. En cualquiera de los dos casos se comprueba después el bit de error y si esta activado se invierte la celda. Si el valor es igual a 0 se dibujan los candidatos para esa celda con cuadrados en negro. Una vez se ha rellenado el sudoku, se dibujan los números correspondientes a cada fila y columna, se dibuja el mensaje de "Introduce la fila A para acabar" y se dibujan los lugares donde irán los tiempos.

```
void pintarTablero(){

      int posicionX1=10;
      int posicionY1=30;

      int posicionX2=posicionX1+180;
      int posicionY2=posicionY1+180;

      Lcd_Clr();
      Lcd_Active_Clr();

      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+0,BLACK,3);
      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+20,BLACK,1);
      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+40,BLACK,1);
      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+60,BLACK,3);
      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+80,BLACK,1);
      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+100,BLACK,1);
      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+120,BLACK,3);
      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+140,BLACK,1);
      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+160,BLACK,1);
      Lcd_Draw_HLine(posicionX1,posicionX2,posicionY1+180,BLACK,3);


      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+0,BLACK,3);
      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+20,BLACK,1);
      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+40,BLACK,1);
      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+60,BLACK,3);
      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+80,BLACK,1);
      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+100,BLACK,1);
      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+120,BLACK,3);
      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+140,BLACK,1);
      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+160,BLACK,1);
      Lcd_Draw_VLine(posicionY1,posicionY2,posicionX1+180,BLACK,3);
```

```
int x=posicionX1+8;
      int i=0;
      int valor;
      int mascara;
      while(i<NUM_FILAS){
            int j=0;
            int y=posicionY1+4;
            while(j<NUM_FILAS){
                  valor=celda_leer_valor(cuadricula[i][j]);
                  if(valor!=0){
                        mascara=1<<11;
                        if(mascara & cuadricula[i][j]){
                              Lcd_DspAscII8x16(x,y,BLACK,traduccion[valor]);
                        }else{
                              Lcd_DspAscII8x16(x,y,DARKGRAY,traduccion[valor]);
                        }
                        mascara=1<<10;
                        if(mascara & cuadricula[i][j])
                              ReverseSquare(x-5, y-1, x+9, y+14);
                  }else{
                        int n=0;
                        while(n<9){
                              mascara=1<<n;
                              if(mascara & cuadricula[i][j])
                                    Lcd_Dot(x-3+5*(n%3),y+5*(n/3));
                              n++;
                        }
                  }
                  y+=20;
                  j++;
            }
            x+=20;
            i++;
      }
      i=0;
      int pos=posicionY1+4;
      while(i<NUM_FILAS){
            Lcd_DspAscII8x16(posicionX2+6,pos+i*20,BLACK,traduccion[i+1]);
            i++;
      }
      i=0;
      pos=posicionX1+8;
      while(i<NUM_FILAS){
            Lcd_DspAscII8x16(pos+i*20,posicionY2+4,BLACK,traduccion[i+1]);
            i++;
      }

      INT8U text1[]={"T. Total = "};
      Lcd_DspAscII8x16(10,5,BLACK,text1);
      INT8U text2[]={"T. Calculo = "};
      Lcd_DspAscII8x16(160,5,BLACK,text2);
      Lcd_Dma_Trans();
      INT8U text3[]={"Introduce"};
      INT8U text4[]={"la fila A"};
      INT8U text5[]={"para acabar."};
      Lcd_DspAscII8x16(230,80,BLACK,text3);
      Lcd_DspAscII8x16(230,100,BLACK,text4);
      Lcd_DspAscII8x16(230,120,BLACK,text5);

}
```

Actualizar hora

La función recibe como parámetros dos enteros que representan los tiempos total y de ejecución, sin embargo, el primero viene dado en segundos, mientras que el segundo en milisegundos. Utilizando las variables definidas anteriormente resulta fácil traducir los números a cadenas de caracteres. Primero se borra la hora anterior de la pantalla con la función LcdClearRect. El tiempo total se se divide en minutos y segundos, se dibuja con dos dígitos que representan los minutos seguidos de : y de otros dos dígitos que representan los minutos. El juego no debería tardar más de 99 minutos y por eso cuando los minutos llegan a 100, se reinician. El tiempo de ejecución dispone de dos formatos, si el tiempo es menor a 10000 se muestra como cuatro dígitos en milisegundos, de lo contrario se muestra con el formato anterior.

```
void actualizarHora(int hora1,int hora2){
	LcdClrRect(100, 5, 150, 25, WHITE);
	int min=(hora1/60)%100;
	int seg=hora1%60;

	INT8U
text1[]={digitos[min/10],digitos[min%10],':',digitos[seg/10],digitos[se
g%10],'\0'};
	Lcd_DspAscII8x16(100,5,BLACK,text1);

	LcdClrRect(260, 5, 320, 25, WHITE);

	if(hora2>9999){
		hora2=hora2/1000;
		min=(hora2/60)%100;
		seg=hora2%60;

		INT8U
text2[]={digitos[min/10],digitos[min%10],':',digitos[seg/10],digitos[se
g%10],'\0'};
		Lcd_DspAscII8x16(260,5,BLACK,text2);

	}else{
		INT8U
text2[]={digitos[hora2/1000],digitos[(hora2/100)%10],digitos[(hora2/10)
%10],digitos[hora2%10],'\0'};
		Lcd_DspAscII8x16(260,5,BLACK,text2);
	}

	Lcd_Dma_Trans();
}
```

Reiniciar tablero

Si el usuario termina el sudoku o pulsa la tecla A se lanza el método reiniciarTablero. Este método copia las celdas de la variable original a cuadricula para que recuperen su valor inicial.

```
void reiniciarTablero(CELDA cuadricula1[NUM_FILAS][NUM_COLUMNAS],CELDA
cuadricula2[NUM_FILAS][NUM_COLUMNAS]){
      int i=0;
      while(i<NUM_FILAS){
            int j=0;
            while(j<NUM_FILAS){
                  cuadricula2[i][j]=cuadricula1[i][j];
                  j++;
            }
            i++;
      }

}
```

Juego sudoku

Lo primero que hace es inicializar la pantalla. Una vez inicializada entra en un bucle infinito del juego sudoku en el que recalcula el tablero,dibuja la pantalla inicial y espera a que se pulse una tecla. Cuando el usuario pulsa una tecla reinicia, los timers 2 y 0, pinta el tablero, muestra una F en el 8led y espera a que el usuario introduzca un valor con los botones o inserte la letra A. Si el usuario pulsa la tecla A, se reinicia el tablero, las variables empezar y confirmación se ponen a 0 y se realiza otra iteración en el bucle. Si el usuario introduce un valor, lo guarda, actualiza tiempos, muestra una C en el 8led y espera a que el usuario introduzca un valor. Cuando el usuario introduce un valor se sigue el procedimiento anterior y se pide un valor para la celda. Cuando el usuario introduce un valor, se modifica la cuadricula, se recalcula el tablero y si ha terminado lo reinicia, en caso contrario lo vuelve a mostrar por pantalla.

```
void juegoSudoku(){
      Lcd_Init();

      while(1){
            Lcd_Clr();
            Lcd_Active_Clr();

            char ready='N';

            sudoku9x9_c_c(cuadricula,&ready);


            INT8U text[]={"Pulsa un boton para empezar"};
            Lcd_DspAscII8x16(50,50,BLACK,text);
            Lcd_Dma_Trans();

            while(empezar==0);


            Timer0_Empezar();
            Timer2_Empezar();

            tiempoCalculos=0;
            while(ready!='Y'){
                  pintarTablero();
                  D8Led_symbol(15);
                  int_count=0;
                  comparacion=11;
                  tiempoCalculos+=Timer2_Leer();
                  while(confirmacion==0){
                        if(tiempoTotal!=Timer0_Leer()/1000000){
                              tiempoTotal=Timer0_Leer()/1000000;
                              actualizar-
Hora(tiempoTotal,tiempoCalculos/1000);
                        }
                  }
```

```
Timer2_Empezar();
                int fila=int_count;
                if(fila!=10){
                      comparacion=10;
                      confirmacion=0;

                      D8Led_symbol(12);
                      int_count=0;
                      tiempoCalculos+=Timer2_Leer();
                      while(confirmacion==0){
                            if(tiempoTotal!=Timer0_Leer()/1000000){
                                  tiempoTotal=Timer0_Leer()/1000000;
                                  actualizar-
Hora(tiempoTotal,tiempoCalculos/1000);
                            }
                      }
                      Timer2_Empezar();
                      confirmacion=0;
                      int columna=int_count;
                      int_count=0;
                      reset=0;
                      D8Led_symbol(16);
                      tiempoCalculos+=Timer2_Leer();
                      while(confirmacion==0){
                            if(tiempoTotal!=Timer0_Leer()/1000000){
                                  tiempoTotal=Timer0_Leer()/1000000;
                                  actualizar-
Hora(tiempoTotal,tiempoCalculos/1000);
                            }
                      }
                      Timer2_Empezar();
                      confirmacion=0;
                      int valor=int_count;

                      celda_poner_valor(&cuadricula[columna-1][fila-
1],valor);


                      sudoku9x9_c_c(cuadricula,&ready);
                }else{
                      ready='Y';
                }
          }
          reiniciarTablero(original,cuadricula);
          empezar=0;
          confirmacion=0;
     }

}
```

Problemas encontrados

Tuve algunas dificultades para entender cómo funcionaban los timers y la pantalla lcd.

Cuando implemente la gestión de rebotes no tuve en cuenta que aunque las interrupciones de botón estaban deshabilitadas, se lanzaban las interrupciones pendientes una vez se rehabilitan. Para arreglarlo es necesario el bit correspondiente en I_ISCP para eliminar las solicitudes de interrupción.

Conclusiones

El diseño de programas en una placa real y a tan bajo nivel trae consigo muchos problemas independientes del programa a implementar, sin embargo también hace más fácil ajustar el programa a lo que deseas.

En esta práctica se han utilizado diferentes códigos cada uno con sus ventajas e inconvenientes. La elección del código a utilizar depende del proyecto a desarrollar, sin embargo para proyectos grandes resulta más adecuado utilizar lenguajes de alto nivel como C.