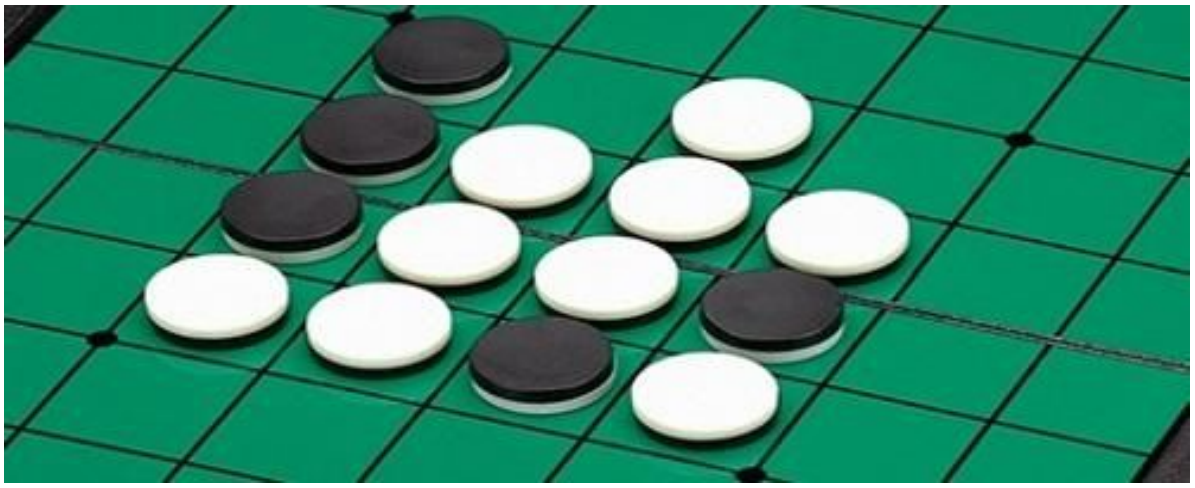
PROYECTO HARDWARE

REVERSI 8



Autores:
Daniel Cay Delgado (741066)
Alejandro Gutiérrez Bolea (735089)

3º ingeniería informática
Enero 2019

ÍNDICE

RESUMEN

En referencia a lo realizado en la **práctica 2**, comienza el desarrollo del soporte necesario para jugar al Reversi directamente desde la placa. Para ello, se elabora la entrada salida básica mediante el uso de los dos botones y el display de 8 segmentos de la placa. En primer lugar, se implementa el tratamiento de las excepciones, puesto que durante la ejecución del código se pueden dar casos en los que el procesador detecta un error. Frente a esto, se ha realizado una función de tratamiento que capture dichas excepciones y que se encargue de hacer entender al usuario que está utilizando la placa qué tipo de excepción se ha producido (mediante el encendido de la led izquierda, derecha o ambas al mismo). Además,se ha desarrollado una pila de depuración circular con el objetivo de introducir aquellos eventos que resulten útiles durante el desarrollo e implementación del programa (eliminación de rebotes). También ha comenzado la incorporación de diversos periféricos con los que interactuar, consiguiendo a la finalización de la práctica el colocar la ficha en una determinada posición del tablero mediante la pulsación de los dos botones que posee la placa, además de observar la posición actual de la misma en el 8led. Para comprobar que el programa no se ha "caído", se ha elaborado un latido, cuya funcionalidad es encender y apagar el led de la izquierda a una velocidad asociada a generar 50 interrupciones por minuto y a parpadear a 2 Hz. Por último en lo que respecta a esta práctica, se han eliminado los rebotes producidos por los botones, para ello, como ya se ha comentado, se ha hecho uso de la pila de depuración.

En cuanto a la **práctica 3**, esta supone la finalización del proyecto. En primer lugar, se ha programado la pantalla LCD táctil con el fin de visualizar el tablero y la disposición de las fichas en un instante determinado de la partida, así como demás información (fila y columna en la que se encuentra la ficha a colocar, duración hasta el momento de la partida, coste temporal de la ejecución de la función patrón volteo…).Otro aspecto es que ahora el programa se ejecuta en modo usuario (parte de él en modo supervisor), para ello, ha sido necesario modificar el procesador. El último aspecto realizado ha sido el de posibilitar la ejecución del juego directamente sobre la placa, independientemente del entorno de programación (en este caso *Eclipse*) que se ha estado utilizando (se ha modificado el archivo *44binit* de forma que se copie el código presente en la memoria Flash a la RAM, posibilitando así la correcta ejecución sobre la propia placa).

Para terminar, cabe mencionar que tras el cursado de la asignatura, se ha sido capaz de cumplir con el objetivo de lograr jugar al reversi sobre la placa directamente, con total autonomía del entorno de desarrollo del juego, logrando así entender la compleja interacción entre el código desarrollado y los periféricos vinculados a este.
Cabe destacar que el código realizado tiene como propósito su reutilización, por lo que la modularidad que presenta ha ido dirigida a dicho aspecto (separación de las funciones que interactúan con los periféricos a bajo nivel de las que gestionan el comportamiento dichos componentes).

INTRODUCCIÓN

Se presenta la memoria de las prácticas 2 y 3 de la asignatura de *Proyecto Hardware* de 3º de *Ingeniería Informática* de la *Universidad de Zaragoza (EINA)* a fecha de enero de 2019.
Dichas prácticas se sitúan en el contexto de que ya ha sido desarrollada la primera práctica de esta asignatura, en la que se ha comenzado con la realización de un proyecto que tiene por objetivo el poder jugar al juego de mesa *Reversi* sobre la placa Embest S3CEV40*,* es decir, hacer uso de los recursos de los que dispone la placa (pantalla táctil, botones, 8led…) para hacer de la ejecución del juego una mejor experiencia.
La situación antes de la realización de la práctica 2 y 3, es que se ha optimizado parte de código del juego (función *patron_volteo* y *ficha_valida*) con el fin de que su ejecución sea más eficiente. El es que este se puede jugar pero sobre el entorno de programación exclusivamente. La tarea de las siguientes prácticas será adaptar el programa para que acepte el uso de periféricos logrando el objetivo comentando anteriormente.
A continuación, se va a explicar el procedimiento seguido con el fin de lograr los objetivos planteados por las prácticas pendientes de realizar, explicando qué se ha hecho, por y para qué.

OBJETIVOS

El objetivo de la práctica 2 es desarrollar el soporte necesario para jugar al *Reversi* directamente desde la placa, por lo que se crearán los elementos de entrada salida básicos (uso de los botones y del 8led).
En cuanto a la práctica 3, se finalizará el proyecto en el que se ha estado trabajando a lo largo de las diferentes sesiones, logrando un sistema empotrado autónomo de los ordenadores del laboratorio con el que se pueda jugar directamente. Para ello se deberá añadir el uso de la pantalla *LCD* táctil, así como cargar el código desarrollado 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.

MEMORIA TÉCNICA

ESTRUCTURA DEL PROYECTO

A continuación se muestra la estructura del proyecto final, es decir, juntando todos los ficheros de la práctica 2 y de la práctica 3.

***8led.c y 8led.h*:** Funciones para gestionar el funcionamiento del display de 8 segmentos led.

- void D8Led_init(void);
- void D8Led_clear(void);
- void D8Led_symbol(int value);
- void parpadear(const int valor, const int tOn, const int tOff, const int times);

***Bmp.c y Bmp.h*:** Funciones de control y visualización del LCD y definición de los mapas de bits.

- void CursorInit(void);
- void CursorPush(INT16U x, INT16U y);
- void CursorPop(void);
- void BitmapPop(INT16U x, INT16U y, STRU_BITMAP Stru_Bitmap);
- void BitmapView(INT16U x, INT16U y, STRU_BITMAP Stru_Bitmap);
- void CursorView(INT16U x, INT16U y);

***botones_antirebote.c*:** Funciones para el control y gestión de los rebotes.

- void inicializar();
- void callback(estado_button boton);
- void button_empezar();
- void gestion(estado_button *boton);

***button.c y button.h*:** Funciones para el control y gestión de los botones.

- void Eint4567_init(void);
- void button_iniciar();
- estado_button button_estado();
- void cambiarBoton(estado_button e);
- void activarInterrupciones();

***excepciones.c*:** Funciones encargadas de capturar y gestionar las distintas excepciones.

- void ISR_Exception(void);
- void exception_inicializar(void);

lcd.c y lcd.h: Funciones para el control y visualización de elementos en el lcd.

- INT8U LCD_GetPixel(INT16U usX, INT16U usY);
- void Lcd_Init(void);
- void Lcd_Active_Clr(void);
- void Lcd_Clr(void);
- void Lcd_Test(void);
- void Lcd_Dma_Trans(void);
- void LcdVirtualToTrue(void);
- void LcdClrRect(INT16 usLeft, INT16 usTop, INT16 usRight, INT16 usBottom, INT8U ucColor);
- void Lcd_Draw_Box(INT16 usLeft, INT16 usTop, INT16 usRight, INT16 usBottom, INT8U ucColor);
- void Lcd_Draw_Line(INT16 usX0, INT16 usY0, INT16 usX1, INT16 usY1, INT8U ucColor, INT16U usWidth);
- void Lcd_Draw_HLine(INT16 usX0, INT16 usX1, INT16 usY0, INT8U ucColor, INT16U usWidth);
- void Lcd_Draw_VLine(INT16 usY0, INT16 usY1, INT16 usX0, INT8U ucColor, INT16U usWidth);
- void Lcd_Anti_Disp(INT16U usX0, INT16U usY0, INT16U usX1, INT16U usY1);
- void Lcd_DisplayChar(INT16U usX0, INT16U usY0, INT8U ucChar);
- void Lcd_DisplayString(INT16U usX0, INT16U usY0, INT8U *pucStr);
- void Lcd_DisplayShort(INT16 sX, INT16 sY, INT16U usInt);
- void Zdma0Done(void) __attribute__((interrupt("IRQ")));
- void Lcd_DspAscII6x8(INT16U usX0, INT16U usY0, INT8U ForeColor, INT8U *pucChar);
- void Lcd_DspAscII8x16(INT16U x0, INT16U y0, INT8U ForeColor, INT8U *s);
- void Lcd_DspHz16(INT16U x0, INT16U y0, INT8U ForeColor, INT8U *s);
- void ReverseLine(INT32U ulHeight, INT32U ulY);
- INT8U LCD_GetPixel(INT16U usX, INT16U usY);

led.c y led.h**:** Funciones para el control de los leds de la placa.

- void leds_on(); // todos los leds encendidos
- void leds_off(); // todos los leds apagados (*)
- void led1_on(); // led 1 encendido (*)
- void led1_off(); // led 1 apagado
- void led2_on(); // led 2 encendido
- void led2_off(); // led 2 apagado
- void leds_switch(); // invierte el valor de los leds (*)
- void Led_Display(int LedStatus); // control directo del LED

main.c**:** Es el programa principal del proyecto. Tiene funciones para gestionar la máquina de estados, calibrar la pantalla, seleccionar casillas con la pantalla táctil e iniciar todos los elementos de la placa.

***pantalla.c*:** Conjunto de funciones encargadas de mostrar por pantalla todos los elementos necesarios para la ejecución del juego.

- void mostrarInicio();
- void mostrarFila(int xpos);
- void mostrarColumna(int ypos);
- void auxMT(char t, int pos);
- void mostrarTiempo(int tiempo);
- void auxPatron(char t, int pos);
- void mostrarTiempoPatron(int tiempo;
- void mostrarTablero();
- void PulseJugar();
- void PulseCancelar();
- void rellenarTablero();
- void colocarPosibleFicha(int x, int y);
- void parpadeo(int x, int y, int cambio);
- void BlancasGanan();
- void NegrasGanan();
- void Empate();

***reversi8_2017.c, reversiASM-PASO3.asm y reversiASM-PASO4.asm*:** Funciones encargadas del funcionamiento del juego reversi.

void limpiarPila();
void push_debug(uint8_t ID_evento, uint32_t auxData);
- void init_table(char tablero[][DIM], char candidatas[][DIM]);
- void esperar_mov(char *ready);
- char ficha_valida(char tablero[][DIM], char f, char c, int *posicion_valida);
- int patron_volteo(char tablero[][DIM], int *longitud, char FA, char CA, char SF, char SC, char color);
- int patron_volteo_test(signed char tablero[][DIM], int *longitud, signed char f, signed char c, signed char SF, signed char SC, signed char color)
- void voltear(char tablero[][DIM], char FA, char CA, char SF, char SC, int n, char color);
- int actualizar_tablero(char tablero[][DIM], char f, char c, char color);
- int elegir_mov(char candidatas[][DIM], char tablero[][DIM], char *f, char *c);
- void contar(char tablero[][DIM], int *b, int *n);
- void actualizar_candidatas(char candidatas[][DIM], char f, char c);
- char fichaPosicion(int fila, int columna);
- int puedoColocar(int xpos, int ypos);
- int fichaPosicionCandidatas(int fila, int columna);
- int reversi8(char fila, char columna, int *tiempoPatron, int *fin);
- void reversi_main();

timer2.c**:** Funciones para el control del timer2, encargado de medir el tiempo y utilizado para mostrar el tiempo de juego y el tiempo de cálculos.

- void timer2_ISR(void);
- void timer2_inicializar(void);
- void timer2_empezar(void);
- unsigned int timer2_leer(void);
- unsigned int timer2_parar(void);

tp.c y tp.h**:** Funciones para el control de las pulsaciones en el panel táctil de la placa.

- void TS_Test(void);
- void TS_init(void);
- void TSInt(void);
- void TS_close(void);
- void Lcd_TC(void);
- void DesignREC(ULONG tx, ULONG ty);
- void Check_Sel(void);

Subdirectorio ./common**:** Contiene los ficheros utilizados para la inicialización y configuración de la placa y sus distintos elementos.

PRÁCTICA 2

ABSTRACCIÓN DE HARDWARE

Con el fin de poder llevar a cabo el desarrollo del proyecto sin utilizar siempre la placa se ha realizado una abstracción del Hardware. Esto es necesario realizarlo debido a la limitación de placas en el laboratorio o en caso de querer trabajar fuera de los horarios de utilización de las mismas, por lo que será necesario el uso del emulador. Para su realización se ha creado una condición mediante #ifdef USAR_EMULACION. De esta manera, todas las definiciones, funciones o llamadas, que tienen que ver con el emulador, se han introducido dentro de una condición como la mencionada anteriormente.
Ejemplo de uso:
Si no se encuentra comentado se guardará el estado del led en dos variables diferentes.

```
//#define USAR_EMULACION
int on1=0;
int on2=0;

/*--- codigo de las funciones públicas ---*/
void leds_on()
{
    #ifdef USAR_EMULACION
        on1=1;
        on2=1;
    #else
        Led_Display(0x3);
    #endif
}
```

TRATAMIENTO DE EXCEPCIONES

De acuerdo al enunciado, es necesario capturar las excepciones de tipo *Data Abort, Undefined Instruction* y *Software Interrupt.* Para ello, se ha creado el fichero *excepciones.c*. Dentro de este, la función de interrupción encargada de gestionar las excepciones es *ISR_Exception* (sea cual sea la excepción de las nombradas que provoque el salto de la interrupción, se saltará a dicha función). Su comportamiento es el de leer la variable de estado *CPSR* (para ello, se hace uso del método existente en *C* denominado *asm*, que permite ejecutar instrucciones correspondientes a nivel ensamblador en dicho lenguaje) *y* comparar del valor leído con el que se debería corresponder para saber si ha saltado una excepción u otra, de ahí que el valor se compare con 23 (*Data Abort*), 19 (*SWI*) y 27(*Undef*). Además de leer el estado, almacenamos la instrucción causante de la excepción. Tras conocer qué tipo de excepción ha saltado, se procede a mostrar a través de los led el código correspondiente (led izquierdo iluminado para *Data Abort*, led derecho para *SWI* y ambos leds encendidos para *Undef)*. Por último, sea el caso que sea de los 3, se detiene la ejecución del programa (bucle infinito mostrando a través de los led la razón del error producido).

PILA DE DEPURACIÓN

La implementación de dicha pila ha ido centrada en obtener información relevante que facilite el desarrollo del programa. Puesto que, como se comentará posteriormente, ha sido necesario gestionar los rebotes de los botones y se quiere evitar que el programa interprete un rebote como la pulsación de nuevo de un botón, el uso que se le ha dado a la pila de depuración ha sido el de almacenar el instante temporal en el que se ha producido la interrupción, así como el motivo de dicha interrupción (en este caso la pulsación de un botón, a la que se le ha asociado el código 3). Para conseguir este comportamiento, se ha implementado la función *push_debug(uint8_t ID_evento, uint32_t auxData)* situada en *reversi8_2018.c*, que es invocada desde la función *Eint4567_ISR()* perteneciente al fichero *button.c* que se llama como resultado de la interrupción generada al pulsar un botón. Esta función apila dos enteros, el primero incluirá en el byte más significativo el campo *ID_evento*, que permita identificar qué interrupción ha saltado o qué evento se ha producido, concatenado con los 24 bits menos significativos del campo *auxData* (datos auxiliares del evento, por ejemplo el botón pulsado, número de ticks, etc).
Para obtener el instante en el que salta la interrupción del botón, se ha utilizado la función *timer0_leer()* presente en el fichero *timer.c*.
La pila se han situado antes de las correspondientes de los distintos modos de usuario, además se trata de una pila circular, con el fin de evitar sobreescribir las pilas de dichos modos y su tamaño es suficiente como para guardar la información de 10 interrupciones (la información de cada una ocupa 8 bytes).

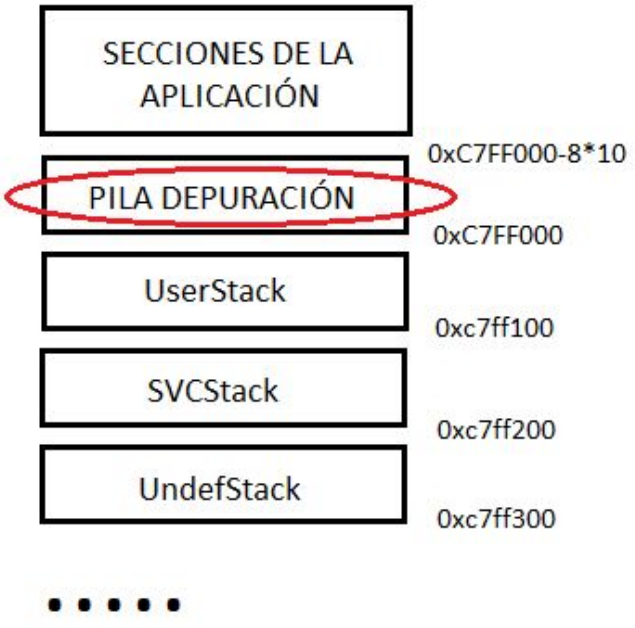


Foto 1: Disposición de la pila de depuración en la memoria del programa

LATIDO

Para saber si el programa sigue vivo se ha implementado un latido con los leds de la placa con la ayuda del timer0. Se ha programado dicho timer para que genere 50 interrupciones por segundo y el led de la izquierda parpadee (on/off) a 2 Hz. Para ello se ha utilizado la siguiente fórmula:

$$F = MCLK/\left((\text{valor de pre-escalado} + 1)(\text{valor del divisor})\right)$$

Como pre-escalado se ha usado 256 y como divisor se selecciona la entrada 00 del mux corresponde a un divisor de ½ y el MCLK es igual a 64*10^6. Realizando la fórmula, el resultado del valor inicial de F es 31250 teniendo en cuenta que el comparador es 0.

ELIMINACIÓN DE LOS REBOTES EN LOS PULSADORES

Cuando se pulsa un botón, este genera rebotes, provocando que salte la interrupción generada debido a la pulsación de dicho botón repetidas veces. Para evitar que esto afecte a la ejecución del programa (no se quiere que cada vez que se pulse un botón, el programa crea que se ha pulsado más veces), se ha utilizado la pila de depuración ya comentada. Su uso ha consistido en pulsar un botón, para después parar la ejecución del programa y comprobar cuántas interrupciones se han guardado en la pila debido a los rebotes. Con el fin de evitar los rebotes, se ha calculado el tiempo que ha pasado entre la primera interrupción generada debido a la pulsación real del botón, y a la última debido a un rebote.
Una vez se posee esta información, cabe describir la función *gestion()*, presente en el fichero *botones_antirebotes.c*, ya que en este se encuentran aquellas funciones que tienen que ver con los botones, pero que no interactúan con el hardware directamente (esta distribución es debida a la búsqueda de la modularidad del código, así como su reutilización.
Antes de profundizar sobre ella, cabe indicar que cuando se pulsa un botón, el comportamiento de la rutina que es llamada consiste en desactivar las interrupciones y, mediante la función de *callback()* hacer saber a *botones_antirebotes* qué botón se ha pulsado además de poner en marcha el autómata asociado.
Volviendo a la función *gestion()*, su comportamiento consiste en esperar el tiempo que se ha calculado para evitar los rebotes, para después estar comprobando si se ha soltado ya el botón (tras el paso de 20ms que es cada cuanto hay que realizar la comprobación de acuerdo al enunciado). Cuando esto ocurre, se procede a esperar otro *x* tiempo para evitar los rebotes de bajada, y por último, se reactivan de nuevo las interrupciones.

Por último, para conocer si un botón está pulsado y en caso de que así sea, cuál, se ha utilizado el registro *rPDATG* dentro de la función *button_estado()* del fichero *button.c* (el bit 6 ó 7 a 1 indican que se ha pulsado el botón izquierdo o el derecho respectivamente).

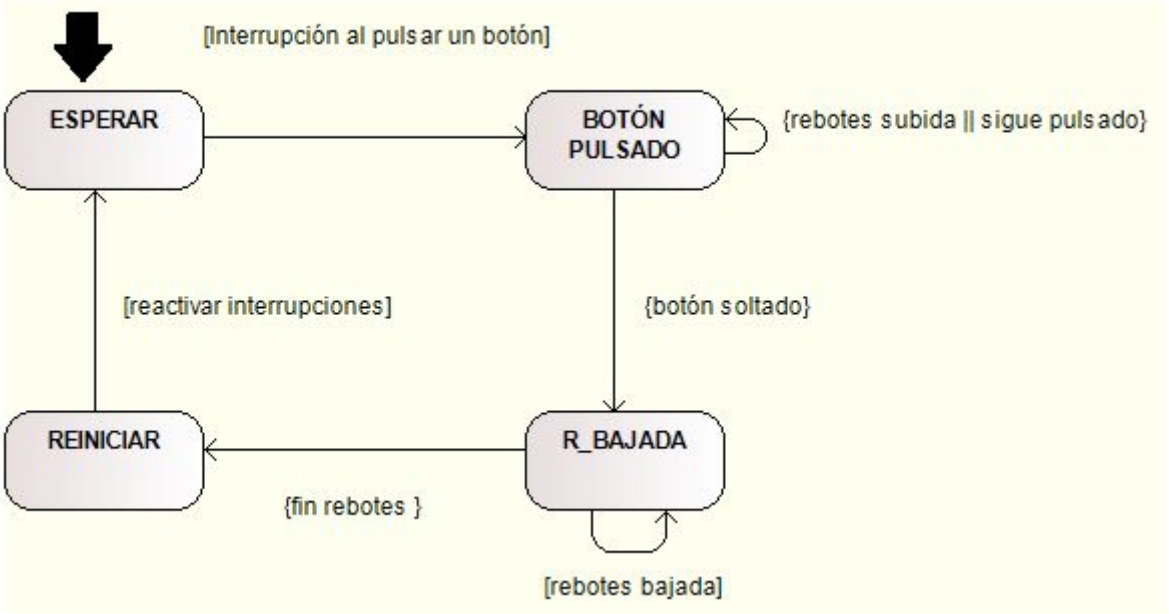


Foto 2: Autómata que expresa el comportamiento de la función *callback()* para evitar rebotes

PRÁCTICA 3

PANTALLA LCD TÁCTIL

Uno de los puntos a realizar en esta práctica, ha sido el de introducir el uso de la pantalla táctil de la que dispone la placa en la ejecución del juego. Permitirá al jugador observar el estado de la partida además de información relativa a esta (tiempo que lleva la partida, coste temporal de la función *patron_volteo()*, posición actual de la ficha a colocar…). A parte, el usuario podrá pulsar sobre ella tanto para calibrar la propia pantalla, como para empezar a jugar y colocar la ficha una vez esté en la posición deseada.
En primer lugar, toda lo que se observa por la pantalla en algún momento de la partida se encuentra en una serie de funciones presentes en el fichero *pantalla.c*, en el cual se ha utilizado *Lcd_DspAscII8x16()* para mostrar todo tipo de mensajes asociados a letras y números y *Lcd_Draw_Box()* para dibujar tanto las fichas como el tablero. Tras utilizar estas funciones, es obligatorio llamar a *Lcd_Dma_Trans()*, ya que es la encargada de transmitir lo que se ha "dibujado en memoria virtual con las funciones anteriores" a la pantalla propiamente dicha.
Para gestionar su comportamiento como pantalla táctil, se ha gestionado la interacción del usuario con la pantalla mediante la rutina de interrupción *TSInt(void)* presente en el fichero *tp.c*. Se supone que la pantalla tiene dos estados, pulsada o no pulsada, y cada vez que salte dicha rutina de interrupción, se cambiará su estado a pulsada, permitiendo así que desde el *main()*, donde está implementado el autómata del programa, se sea consciente de que se ha pulsado la pantalla. También se guardará la coordenada *x* e *y* asociadas al punto de la pantalla en el que se ha pulsado, puesto que cuando el jugador desee colocar la ficha en una posición, deberá pulsar en el centro de la pantalla, por lo que será necesario saber dónde ha pulsado (*pulsaCentro()*). Para modificar cierta información de la que aparece en el *LCD*, se ha decidido no borrar la pantalla entera mediante *Lcd_Clr()*, sino que se ha utilizado *LcdClrRect()* para eliminar el contenido de una zona en concreto.

CALIBRACIÓN DEL LCD

Con el fin de saber cual es el centro de la pantalla, se ha elaborado una función *calibrar()* presente en el *main()* que pide al usuario pulsar en los extremos de la pantalla, a fin de saber sus dimensiones iniciales, pudiendo reducirlas a un rectángulo menor hasta obtener las medidas aproximadas del centro de la pantalla.
Para llevar a cabo la calibración se realizan 10 mediciones de cada punto en el que el usuario pulsa para cada una de las cuatro esquinas. Se calcula la media y se almacenan los valores en variables globales. Para el cálculo se han utilizado variables volátiles ya que la rutina de interrupción del LCD se encuentra en fichero *tp.c.*

MÁQUINA DE ESTADOS

En *main.c* se encuentra la máquina de estados que expresa el comportamiento del programa a lo largo de su ejecución. Su conducta es la siguiente:
En primer lugar, se accede a un estado inicial donde se muestran las instrucciones para jugar, esperando entonces a que el jugador pulse la pantalla o uno de los botones para comenzar la partida. Cuando ocurra una de las acciones anteriores, se mostrará el tablero con la distribución de las fichas así como la información sobre la partida ya comentadas. En este nuevo estado en el que se encuentra el programa, se estará comprobando que el jugador pulse la pantalla para indicar que quiere colocar en la posición actual la ficha, así como si se pulsa algún botón (función *gestion()*) para desplazar la ficha a lo largo del tablero, además de comprobar si ha pasado el tiempo seleccionado para provocar el parpadeo de la ficha candidata a colocar. Cuando el jugador pulse la pantalla, se pasará a un nuevo estado, en el que se esperarán dos segundos, durante los cuales, si se pulsa de nuevo dicha pantalla, se volverá al estado anterior (se ha decidido cancelar el colocar la ficha en la posición actual). Si no se pulsa, se saltará al estado encargado de realizar la jugada elegida invocando a la función *reversi8()*. Dicha función devolverá si la partida ha finalizado, y en caso de que sea así, cuál ha sido el resultado. De acuerdo a esto, el programa volverá al estado que permite colocar una nueva ficha (la partida no ha acabado) o al inicial (la partida si ha terminado).

A continuación se muestra la máquina de estados explicada:

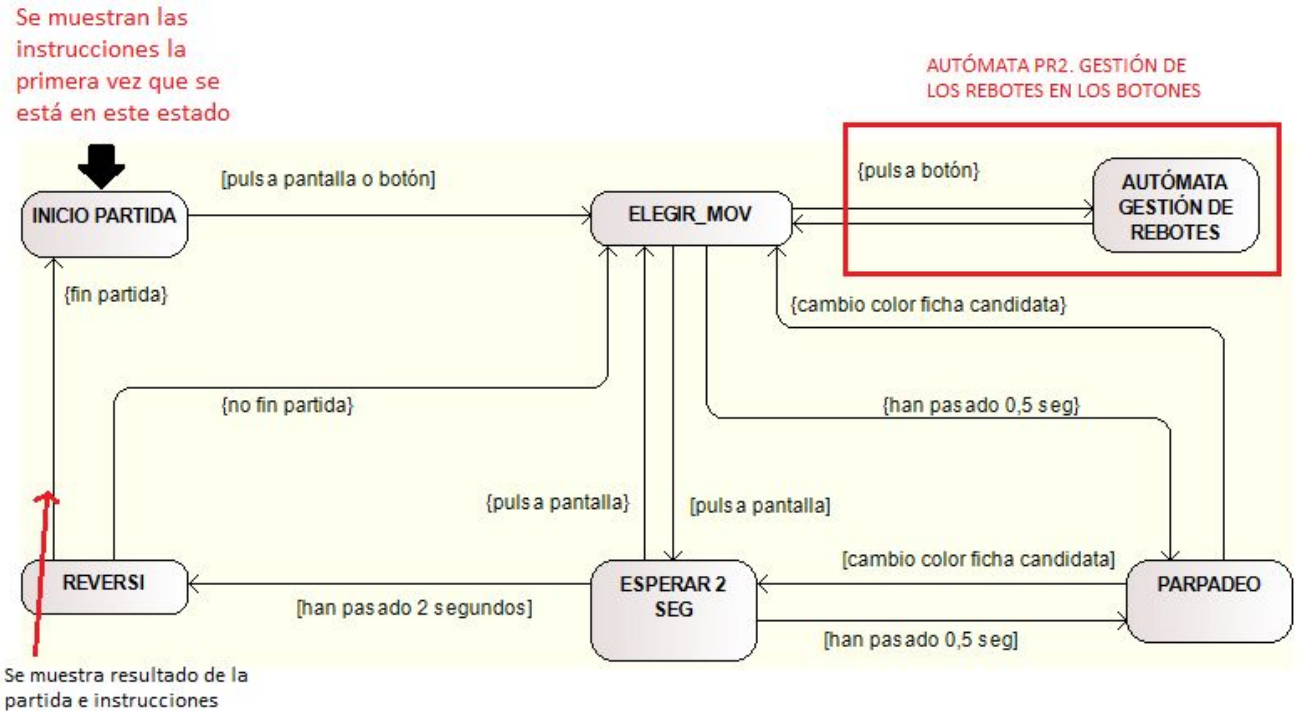


Foto3: Autómata que representa el comportamiento del programa

EJECUCIÓN EN MODO USUARIO

A lo largo de las prácticas, el programa se ha estado ejecutado en modo supervisor, en cambio, ahora se desea su ejecución en modo usuario. Para ello, ha sido necesario acceder al bajo nivel con el fin de modificar el modo del procesador (la instrucción *asm()* presente en *C* permite ejecutar instrucciones en lenguaje ensamblador en dicho lenguaje).

```
//Pasamos a modo USUARIO
asm(".equ MODEMASK, 0x1f");  //Para selección de M[4:0]
asm(".equ USERMODE, 0x10");  //Código del modo User
asm("mrs r0,cpsr");          //LLevamos el registro de estado a r0
asm("bic r0,r0,#MODEMASK");  //Borramos los bits de modo de r0
asm("orr r1,r0,#USERMODE");  //Añadimos el código del modo User y copiamos en r1
asm("msr cpsr_cxsf,r1");     //Escribimos el resultado en el registro de estado
asm("ldr sp, =0xc7ff000");   //Una vez en modo User copiamos la dirección de comienzo de la pila
```

CARGA EN MEMORIA RAM

Para no depender del ordenador y poder jugar directamente en la placa, debemos flashear el archivo .elf en la placa, convirtiéndolo a .bin previamente mediante *JTAG*. Con el código que teníamos hasta el momento no era posible hacerlo por lo que había que realizar alguna modificación. Dichas modificaciones se realizan sobre el fichero *44binit.asm* haciendo que el juego se ejecute directamente en las direcciones de memoria correspondientes a la memoria *RAM*.

En primer lugar, se modifica la dirección *SMRDATA* restándole a su valor por defecto *0xc000000*, trabajando directamente, como se ha indicado anteriormente, sobre la memoria *RAM* de la placa. Esta resta se debe a que, aunque el *ELF* se ha creado colocando el código a partir de la dirección 0x0C000000 (con el fichero de configuración del enlazador), al bajarlo a la placa se está copiando a partir de la dirección *0x00000000*. Como consecuencia aunque el código se posicione en la memoria flash a partir de la dirección *0x00000000* todos los símbolos y etiquetas resueltos por el enlazador toman el valor como si el código comenzase en la dirección *0x0C000000*. Posteriormente, se ha cambiado la parte que escribe ceros en la memoria *RAM* antes de saltar al *main()*. Este cambio se ha realizado siguiendo los ficheros proporcionados por el fabricante. Además, dado que con esta solución teníamos algún problema, se ha modificado ligeramente la llamada a Main. En vez de hacer un salto poniendo directamente en pc la dirección del Main, en primer lugar se guarda en “LR” la posición actual del *PC* y posteriormente se salta al *main()*. Por último, para generar el fichero binario (*.bin*) y flashearlo en la placa, se siguen los siguientes pasos. En primer lugar se ejecuta la instrucción *arm-none-eabi-objcopy -O binary practica3.elf practica3.bin* generando así el fichero binario. Posteriormente, para transferirlo a la memoria flash de la placa, se utiliza la instrucción *openocd-0.7.0.exe -f test/arm-fdi_ucm.cfg -c “program practica3.bin 0x00000000*.

PROBLEMAS ENCONTRADOS Y SOLUCIONES

Práctica 2:El principal problema que surgió en esta práctica consistió en que al capturar la información deseada en la pila de depuración, saltaba una de las excepciones programadas. Tras investigar el problema y preguntar a los profesores, se descubrió que cuando se habían almacenado un número determinado de elementos se comenzaba a pisar la pila de uno de los modos de ejecución, cosa que no puede ocurrir. Por ello, se tuvo que adaptar el tamaño de la pila a la distribución de la memoria del programa, con el fin de evitar pisar zonas de memoria que no deben se sobreescritas.
Otro problema fue el de lograr entender cómo debía ser el autómata a implementar, puesto que no se era del todo consciente de las partes de las que debía disponer el programa (*main()* que implemente el autómata que va interaccionando con los periféricos de la placa con el fin de lograr la correcta interacción entre el código del juego y dichos periféricos).

Práctica 3: En esta práctica no se pueden destacar grandes problemas puesto que todas las dudas que han ido surgiendo se han solucionado sin grandes dificultades mediante un mejor estudio de la documentación junto con el apoyo necesario por parte de los profesores. Esto se debe a que tras la realización de las dos prácticas anteriores, la práctica en cuestión no aportaba un concepto muy distinto a lo realizado en las prácticas anteriores a diferencia de lo que ocurría con estas (comenzar a trabajar con la placa combinando además alto y bajo nivel, e implementar las primeras interacciones entre los periféricos de la placa y el juego implementado).

CORRECCIONES

La única corrección que ha sido necesario realizar ha tenido lugar tras la práctica 2, puesto que la distribución del programa no era la adecuada, es decir, el autómata que rige el comportamiento del juego, debe estar en su totalidad en el *main*, por lo que no deben existir esperas activas en otro lugar que no sea este. En cambio, el autómata que controla los rebotes, realizaba esperas activas, teniendo por lo tanto que ser estas trasladadas a dicho *main.*

RESULTADOS Y CONCLUSIONES

Debido a la complejidad del proyecto realizado, se considera que se han adquirido gran cantidad de conocimientos con una gran utilidad para el futuro profesional.
Por un lado, se ha comprendido la importancia de la organización, es decir, diferenciar unas versiones de otras, así como tener un código legible, puesto que en proyectos de tal tamaño, si no se mantiene un cierto orden, se complica mucho el sistema de trabajo. Además, al trabajar con un periférico real (no haciendo todo mediante un emulador), se ha sido consciente de la gran cantidad de problemas que pueden surgir, y que dichas complicaciones van más allá de que el código desarrollado esté o no bien escrito (realizar

una correcta comunicación entre el programa realizado y el periférico que utiliza, en este caso, la placa). Para evitar estos problemas, ha sido muy importante estudiar la documentación disponible proporcionada por el fabricante de la placa, comprendiendo que cierta información no se puede encontrar en internet y que debe ser uno mismo el que estudiando la documentación disponible, resuelva las dificultades que van surgiendo (junto con la ayuda del profesorado, claro está).

Un punto muy positivo ha sido el poder probar aquello que íbamos programando en un dispositivo real, cosa que hasta ahora no se había realizado, y que ha hecho mucho más llevaderas tanto la práctica 2 como la 3.

Por último, pese a las dificultades que han aparecido, tras la realización de la práctica 1, y al empezar a entender cuál era una metodología más adecuada de trabajo así como la interacción adecuada con la placa, todo ha sido mucho más sencillo (siempre dentro de la gran dificultad que ha supuesto la asignatura).

ANEXO: CÓDIGO

botones_antirebote.c

```
/*--- ficheros de cabecera ---*/
#include "button.h"
#include "8led.h"
#include "44blib.h"
#include "44b.h"
#include "def.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

typedef estado_button;
enum estado_button {button_none, button_iz, button_dr};
typedef void(function_ptr)(estado_button);
static estado_button boton_pulsado;
typedef estado;
enum estado {inicial,rsubida,encuestar,rbajada,reiniciar};
static volatile estado estadoActual=inicial;
int ahora=0;

/* inicializa el button */
void inicializar(){
    button_iniciar();
}

/* Funcion de callback que almacena el estado del boton */
void callback(estado_button boton){
    ahora=timer0_leer();
    estadoActual=encuestar;
    boton_pulsado=boton;
}

/* Inicializa la funcion callback */
void button_empezar(){
   void (*ptr_func_1)(estado_button)=&callback;
}

/* Gestiona los rebotes del boton */
void gestion(estado_button *boton){
    static int cambiar=0;
    static int esperar;
    *boton=button_none;
    switch(estadoActual){
        case encuestar:
            if((timer0_leer()-ahora)>=50000){
                if (cambiar==0){
                    esperar=timer2_leer();
                    cambiar=1;
                }
                if((timer2_leer()-esperar)>=20){
                        cambiar=0;
                        if(button_estado()!=boton_pulsado){
                                estadoActual=rbajada;
                                ahora=timer0_leer();
                                *boton=boton_pulsado;
                        }
                }
            }

        break;

        case rbajada:
            if((timer0_leer()-ahora)>=50000){
                estadoActual=reiniciar;
            }
        break;

        case reiniciar:
            estadoActual=inicial;
            activarInterrupciones();
        break;
    }
}
```

button.c

```
/*--- ficheros de cabecera ---*/
#include "button.h"
#include "8led.h"
#include "44blib.h"
#include "44b.h"
#include "def.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

/*--- variables globales del módulo ---*/

//Guarda estado del boton
typedef estado_button;
enum estado_button {button_none, button_iz, button_dr};
void (*ptr_func_1)(estado_button)=NULL;

estado_button estadoBoton=button_none;

/* declaración de función que es rutina de servicio de interrupción
 * https://gcc.gnu.org/onlinedocs/gcc/ARM-Function-Attributes.html */
void Eint4567_ISR(void) __attribute__((interrupt("IRQ")));

/*--- codigo de funciones ---*/
/* Rutina de interrupción */
void Eint4567_ISR(void)
{
    rINTMSK |= BIT_EINT4567;
    //push_debug(3,timer0_leer());
    /* Identificar la interrupcion (hay dos pulsadores)*/
    int which_int = rEXTINTPND;
    switch (which_int)
    {
        case 4:
            callback(button_iz);
            estadoBoton=button_iz;
            break;
        case 8:
            callback(button_dr);
            estadoBoton=button_dr;
            break;
        default:
            break;
    }
    /* Finalizar ISR */
    rEXTINTPND = 0xf;               // borra los bits en EXTINTPND
    rI_ISPC   |= BIT_EINT4567;      // borra el bit pendiente en INTPND
}

/* Inicializa el boton */
void button_iniciar()
{
    /* Configuracion del controlador de interrupciones. Estos registros están definidos en 44b.h */
    rI_ISPC    = 0x3ffffff; // Borra INTPND escribiendo 1s en I_ISPC
    rEXTINTPND = 0xf;       // Borra EXTINTPND escribiendo 1s en el propio registro
    rINTMOD    = 0x0;       // Configura las linas como de tipo IRQ
    rINTCON    = 0x1;       // Habilita int. vectorizadas y la linea IRQ (FIQ no)
    rINTMSK    &= ~(BIT_EINT4567); // habilitamos interrupcion linea eint4567 en vector de mascaras
    /* Establece la rutina de servicio para Eint4567 */
    pISR_EINT4567 = (int) Eint4567_ISR;
    /* Configuracion del puerto G */
    rPCONG  = 0xffff;               // Establece la funcion de los pines (EINT0-7)
    rPUPG   = 0x0;                  // Habilita el "pull up" del puerto
    rEXTINT = rEXTINT | 0x22222222;   // Configura las lineas de int. como de flanco de bajada
    /* Por precaucion, se vuelven a borrar los bits de INTPND y EXTINTPND */
    rEXTINTPND = 0xf;
    rI_ISPC    |= (BIT_EINT4567);
    D8Led_symbol(0 & 0x000f);

}
```

```
/* Devuelve el estado del boton en ese momento */
estado_button button_estado(){
    int bit6 = rPDATG & 0x40;
    int bit7 = rPDATG & 0x80;
    if(bit6==0){
        return button_iz;
    }
    else if(bit7==0){
        return button_dr;
    }
    else{
        return button_none;
    }
}

/* Devuelve el estado del boton en ese momento */
estado_button buttonEstado(){
    return estadoBoton;
}

/* Cambia el estado del boton*/
void cambiarBoton(estado_button e){
    estadoBoton=e;
}

/* Activa las interrupciones */
void activarInterrupciones(){
    rEXTINTPND = 0xf;
    rI_ISPC    |= (BIT_EINT4567);
    rINTMSK    &= ~(BIT_EINT4567);
}
```

excepciones.c

```
#include "8led.h"
#include "button.h"
#include "led.h"
#include "timer.h"
#include "44blib.h"
#include "44b.h"
/*--- variables globales ---*/
static int ERRORES=0;
int resultado, instruccion;

void ISR_Exception(void) __attribute__ ((interrupt ("SWI")));
void ISR_Exception(void) __attribute__ ((interrupt ("UNDEF")));
void ISR_Exception(void) __attribute__ ((interrupt ("ABORT")));
/* Rutina de interrupción */
void ISR_Exception(void)
{
        ERRORES++;
        asm("mrs %[resultado], cpsr" : [resultado]  "=r" (resultado) : );
        asm("sub %[instruccion], r14, #8" : [instruccion] "=r" (instruccion) : );
        resultado &= 0x1F;
        if(resultado==23){ //Data abort
            led1_on();
        }
        else if(resultado==19){ //SWI
            led2_on();
        }
        else if(resultado==27){ //Undef
            leds_on();
        }
        while(1);
}

/* Inicializa las excepciones */
void exception_inicializar(void)
{
    /* Establece la rutina de servicio para las excepciones */
    pISR_SWI = (unsigned) ISR_Exception;
    pISR_DABORT = (unsigned) ISR_Exception;
    pISR_UNDEF = (unsigned) ISR_Exception;

}
```

pantalla.c

```
/*--- ficheros de cabecera ---*/
#include "8led.h"
#include "button.h"
#include "led.h"
#include "timer.h"
#include "44blib.h"
#include "44b.h"
#include "tp.h"
#include "def.h"
#include "lcd.h"
#include "Bmp.h"

//Guarda estado de la pantalla
typedef estado_pantalla;
enum estado_pantalla {pulsada, no_pulsada};

/* Muestra el inicio del juego en el LCD */
void mostrarInicio(){
    Lcd_DspAscII8x16(25,25,BLACK,"Toque la pantalla para jugar");
    Lcd_DspAscII8x16(25,40,BLACK,"----------------------------");
    Lcd_DspAscII8x16(25,55,BLACK,"       INSTRUCCIONES");
    Lcd_DspAscII8x16(25,70,BLACK,"----------------------------");
    Lcd_DspAscII8x16(25,85,BLACK,"Boton izq: ->");
    Lcd_DspAscII8x16(25,100,BLACK,"Boton dr: |");
    Lcd_DspAscII8x16(25,112,BLACK,"           v");
    Lcd_DspAscII8x16(25,130,BLACK,"Pulsar pantalla: Fijar ficha");
    Lcd_Dma_Trans();
}

/* Muestra la fila en el LCD */
void mostrarFila(int xpos){
    xpos+=1;
    char xpos2=xpos + '0';
    Lcd_DspAscII8x16(220,60,BLACK,"Fila:");
    Lcd_DspAscII8x16(260,60,BLACK,&xpos2);
    Lcd_Dma_Trans();
}
/* Muestra la columna en el LCD */
void mostrarColumna(int ypos){
    ypos+=1;
    char ypos2=ypos + '0';
    Lcd_DspAscII8x16(220,80,BLACK,"Columna:");
    Lcd_DspAscII8x16(285,80,BLACK,&ypos2);
    Lcd_Dma_Trans();
}

void auxMT(char t, int pos){
    Lcd_DspAscII8x16(220,20,BLACK,"Time(s):");
    Lcd_DspAscII8x16(pos,20,BLACK,&t);
}

/* Muestra el tiempo real del juego en el LCD */
void mostrarTiempo(int tiempo){
    tiempo=tiempo/1000000;
    int t=0;
    int auxT;
    char tiempo2;
    int pos=290;
    int i=0;
    int vec[10];
    while (tiempo!=0){
        auxT=tiempo%10;
        tiempo=tiempo/10;
        tiempo2=auxT + '0';
        vec[i]=tiempo2;
        i++;
    }
    while(i>0){
        auxMT(vec[i-1],pos);
        pos+=8;
        i--;
    }
    Lcd_Dma_Trans();
}
```

```
void auxPatron(char t, int pos){
    Lcd_DspAscII8x16(220,40,BLACK,"P_V(ms):");
    Lcd_DspAscII8x16(pos,40,BLACK,&t);
}

/* Muestra el tiempo de la funcion patrón volteo en el LCD */
void mostrarTiempoPatron(int tiempo){
    if(tiempo==0){
        char tiempo2=tiempo + '0';
        auxPatron(tiempo2,285);
    }
    else{
        int t=0;
        int auxT;
        char tiempo2;
        int pos=290;
        int i=0;
        int vec[10];
        while (tiempo!=0){
            auxT=tiempo%10;
            tiempo=tiempo/10;
            tiempo2=auxT + '0';
            vec[i]=tiempo2;
            i++;
        }
        while(i>0){
            auxPatron(vec[i-1],pos);
            pos+=8;
            i--;
        }
    }
    Lcd_Dma_Trans();
}

/* Muestra el tablero vacio en el LCD */
void mostrarTablero(){
    Lcd_DspAscII6x8(205,0,BLACK,"1");
    Lcd_DspAscII6x8(205,25,BLACK,"2");
    Lcd_DspAscII6x8(205,50,BLACK,"3");
    Lcd_DspAscII6x8(205,75,BLACK,"4");
    Lcd_DspAscII6x8(205,100,BLACK,"5");
    Lcd_DspAscII6x8(205,125,BLACK,"6");
    Lcd_DspAscII6x8(205,150,BLACK,"7");
    Lcd_DspAscII6x8(205,175,BLACK,"8");

    Lcd_DspAscII6x8(0,205,BLACK,"1");
    Lcd_DspAscII6x8(25,205,BLACK,"2");
    Lcd_DspAscII6x8(50,205,BLACK,"3");
    Lcd_DspAscII6x8(75,205,BLACK,"4");
    Lcd_DspAscII6x8(100,205,BLACK,"5");
    Lcd_DspAscII6x8(125,205,BLACK,"6");
    Lcd_DspAscII6x8(150,205,BLACK,"7");
    Lcd_DspAscII6x8(175,205,BLACK,"8");
    /* draw rectangle pattern */
    int x=0;
    while(x<200){
        Lcd_Draw_Box(0,x,25,x+25,15);
        Lcd_Draw_Box(25,x,50,x+25,15);
        Lcd_Draw_Box(50,x,75,x+25,15);
        Lcd_Draw_Box(75,x,100,x+25,15);
        Lcd_Draw_Box(100,x,125,x+25,15);
        Lcd_Draw_Box(125,x,150,x+25,15);
        Lcd_Draw_Box(150,x,175,x+25,15);
        Lcd_Draw_Box(175,x,200,x+25,15);
        x=x+25;
    }
    Lcd_Dma_Trans();
}

/* Muestra mensaje "Pulse para jugar" en el LCD */
void PulseJugar() {
    Lcd_DspAscII8x16(50,215,BLACK,"Pulse para jugar");
    Lcd_Dma_Trans();
}
```

```
/* Muestra mensaje "Pulse para cancelar" en el LCD */
void PulseCancelar() {
    Lcd_DspAscII8x16(50,215,BLACK,"Pulse para cancelar");
    Lcd_Dma_Trans();
}

/* Muestra las fichas existentes en el LCD */
void rellenarTablero(){
    char ficha;
    int i=0; int j=0;
    for(i=0;i<8;i++) {
        for(j=0;j<8;j++) {
            ficha=fichaPosicion(i,j);
            if(ficha==2) {
                LcdClrRect(j*25+5,i*25+5,(j+1)*25-5,(i+1)*25-5,BLACK);      //Escribimos un cuadrado negro
            }
            else if(ficha==1){
                Lcd_Draw_Box(j*25+5,i*25+5,(j+1)*25-5,(i+1)*25-5,BLACK);        //Escribimos un cuadrado gris
            }
        }
    }
    Lcd_Dma_Trans();
}

/* Muestra la ficha a colocar en ese momento en el LCD */
void colocarPosibleFicha(int x, int y) {
    LcdClrRect(y*25+5,x*25+5,(y+1)*25-5,(x+1)*25-5,0xb);
    Lcd_Dma_Trans();
}

/* Muestra la ficha a colocar cambiando de color en el LCD */
void parpadeo(int x, int y, int cambio) {
    if(cambio==0) {
        LcdClrRect(y*25+5,x*25+5,(y+1)*25-5,(x+1)*25-5,WHITE);
    }
    else {
        LcdClrRect(y*25+5,x*25+5,(y+1)*25-5,(x+1)*25-5,BLACK);
    }
    Lcd_Dma_Trans();
}
/* Muestra victoria de fichas blancas e inicio en el LCD */
void BlancasGanan() {
    Lcd_DspAscII8x16(25,0,BLACK,"[BLANCAS HAN GANADO!!]");
    Lcd_DspAscII8x16(25,25,BLACK,"Toque la pantalla para jugar");
    Lcd_DspAscII8x16(25,40,BLACK,"----------------------------");
    Lcd_DspAscII8x16(25,55,BLACK,"       INSTRUCCIONES");
    Lcd_DspAscII8x16(25,70,BLACK,"----------------------------");
    Lcd_DspAscII8x16(25,85,BLACK,"Boton izq: ->");
    Lcd_DspAscII8x16(25,100,BLACK,"Boton dr: |");
    Lcd_DspAscII8x16(25,112,BLACK,"          v");
    Lcd_DspAscII8x16(25,130,BLACK,"Pulsar pantalla: Fijar ficha");
    Lcd_Dma_Trans();
}

/* Muestra victoria de fichas negras e inicio en el LCD */
void NegrasGanan() {
    Lcd_DspAscII8x16(25,0,BLACK,"[NEGRAS HAN GANADO!!]");
    Lcd_DspAscII8x16(25,25,BLACK,"Toque la pantalla para jugar");
    Lcd_DspAscII8x16(25,40,BLACK,"----------------------------");
    Lcd_DspAscII8x16(25,55,BLACK,"       INSTRUCCIONES");
    Lcd_DspAscII8x16(25,70,BLACK,"----------------------------");
    Lcd_DspAscII8x16(25,85,BLACK,"Boton izq: ->");
    Lcd_DspAscII8x16(25,100,BLACK,"Boton dr: |");
    Lcd_DspAscII8x16(25,112,BLACK,"          v");
    Lcd_DspAscII8x16(25,130,BLACK,"Pulsar pantalla: Fijar ficha");
    Lcd_Dma_Trans();
}

/* Muestra empate e inicio en el LCD */
void Empate() {
    Lcd_DspAscII8x16(25,0,BLACK,"[Empate!!]");
    Lcd_DspAscII8x16(25,25,BLACK,"Toque la pantalla para jugar");
    Lcd_DspAscII8x16(25,40,BLACK,"----------------------------");
    Lcd_DspAscII8x16(25,55,BLACK,"       INSTRUCCIONES");
    Lcd_DspAscII8x16(25,70,BLACK,"----------------------------");
    Lcd_DspAscII8x16(25,85,BLACK,"Boton izq: ->");
    Lcd_DspAscII8x16(25,100,BLACK,"Boton dr: |");
    Lcd_DspAscII8x16(25,112,BLACK,"          v");
    Lcd_DspAscII8x16(25,130,BLACK,"Pulsar pantalla: Fijar ficha");
    Lcd_Dma_Trans();
}
```

main.c

```
/*--- ficheros de cabecera ---*/
#include "8led.h"
#include "button.h"
#include "led.h"
#include "timer.h"
#include "44blib.h"
#include "44b.h"
#include "tp.h"
#include "lcd.h"
#include "Bmp.h"
#include <stdio.h>

//Guarda estado de la pantalla
typedef estado_pantalla;
enum estado_pantalla {pulsada, no_pulsada};
//Guarda estado del boton
typedef estado_button;
enum estado_button {button_none, button_iz, button_dr};
estado_button boton=button_none;
//Guarda estado del programa
typedef estado_programa;
enum estado_programa {inicio, elegirMovimiento, esperar_2s, reversi};
estado_programa estadoPrograma=inicio;

int primera=0;
int fin=0;

int xpos=0;
int ypos=0; //posicion ficha gris

int tiempoParpadeo=0;
int tiempo2s=0;
int tiempoPatron=0;
int cambio=1;
int estadoPartida=0;

volatile int tX=0;          //Indica la posición X de la pulsación de la pantalla
volatile int tY=0;          //Indica la posición Y de la pulsación de la pantalla
volatile int X_MIN;     //Valor mínimo de la X en el panel táctil para el tablero
volatile int Y_MIN;     //Valor mínimo de la Y en el panel táctil para el tablero
volatile int X_MAX;     //Valor máximo de la X en el panel táctil para el tablero
volatile int Y_MAX;     //Valor máximo de la Y en el panel táctil para el tablero
```

```
/* Funcion de calibracion de la pantalla */
void calibrar(){
    Lcd_Clr();              //Limpiamos la pantalla
    volatile int minX=0;    //Valor mínimo de la X en el panel táctil para el tablero
    volatile int minY=0;    //Valor mínimo de la Y en el panel táctil para el tablero
    volatile int maxX=0;    //Valor máximo de la X en el panel táctil para el tablero
    volatile int maxY=0;    //Valor máximo de la Y en el panel táctil para el tablero
    int i;                  //Variable que utilizará para comprobar n veces cada esquina del tablero
    int tiempo;
    cambiarEstado(no_pulsada);
    for (i=0; i<1; i++){                                //Hacemos 5 medidas
        Lcd_Clr();
        Lcd_DspAscII8x16(25,25,BLACK,"Superior izquierda");     //Indica la esquina a pulsar
        Lcd_DspAscII8x16(125,125,BLACK,"CALIBRAR");
        Lcd_Dma_Trans();
        cambiarEstado(no_pulsada);
        while(estadoActual()==no_pulsada){}                     //Esperamos que se pulse la pantalla tactil
        cambiarEstado(no_pulsada);
        minX+=tX;
        maxY+=tY;
        tiempo=timer2_leer();
        while(timer2_leer()-tiempo<=1000000){}

        Lcd_Clr();
        Lcd_DspAscII8x16(150,25,BLACK,"Superior derecha");  //Indica la esquina a pulsar
        Lcd_DspAscII8x16(125,125,BLACK,"CALIBRAR");
        Lcd_Dma_Trans();
        cambiarEstado(no_pulsada);
        while(estadoActual()==no_pulsada){}                         //Esperamos que se pulse la pantalla tactil
        cambiarEstado(no_pulsada);

        maxX+=tX;
        maxY+=tY;

        tiempo=timer2_leer();
        while(timer2_leer()-tiempo<=1000000){}

        Lcd_Clr();
        Lcd_DspAscII8x16(25,200,BLACK,"Inferior izquierda");        //Indica la esquina a pulsar
        Lcd_DspAscII8x16(125,125,BLACK,"CALIBRAR");
        Lcd_Dma_Trans();
        cambiarEstado(no_pulsada);

        cambiarEstado(no_pulsada);

        minX+=tX;
        minY+=tY;

        tiempo=timer2_leer();
        while(timer2_leer()-tiempo<=1000000){}


        Lcd_Clr();
        Lcd_DspAscII8x16(150,200,BLACK,"Inferior derecha");         //Indica la esquina a pulsar
        Lcd_DspAscII8x16(125,125,BLACK,"CALIBRAR");
        Lcd_Dma_Trans();
        cambiarEstado(no_pulsada);
        while(estadoActual()==no_pulsada){}                         //Esperamos que se pulse la pantalla tactil
        cambiarEstado(no_pulsada);

        maxX+=tX;
        minY+=tY;

        tiempo=timer2_leer();
        while(timer2_leer()-tiempo<=1000000){}

    }

    //Dado que se han hecho, en total, 10 mediciones de cada punto, dividimos entre 10 para obtener la media de cada punto
    X_MIN= minX/2;
    Y_MIN= minY/2;
    X_MAX= maxX/2;
    Y_MAX= maxY/2;
}
```

```
void Main(void)
{
    /* Inicializa controladores */
    sys_init();                     // Inicializacion de la placa, interrupciones y puertos
    timer_init();                   // Inicializacion del temporizador 0
    D8Led_init();                   // inicializamos el 8led
    inicializar();                  // inicializamos los pulsadores. Cada vez que se pulse se verá reflejado en el 8led
    button_empezar();
    exception_inicializar();        // Inicializacion de excepciones
    timer2_inicializar();           // Inicializacion del temporizador 2
    leds_off();
    limpiarPila();
    reversi_main();
    Lcd_Init();
    /* clear screen */
    Lcd_Clr();
    Lcd_Active_Clr();

    TS_init();

    timer2_empezar();

    //Pasamos a modo USUARIO
    asm(".equ MODEMASK, 0x1f");
    asm(".equ USERMODE, 0x10");
    asm("mrs r0,cpsr");
    asm("bic r0,r0,#MODEMASK");
    asm("orr r1,r0,#USERMODE");
    asm("msr cpsr_cxsf,r1");
    asm("ldr sp, =0xc7ff000");

    calibrar();

    while(1){
        switch(estadoPrograma){
            //Estado inicial
            case inicio:
                //Se muestra el menu de inicio

                //Se muestra el menu de inicio
                if(primera==0){
                    Lcd_Clr();
                    cambiarBoton(button_none);
                    cambiarEstado(no_pulsada);
                    mostrarInicio();
                    primera=1;
                }
                //Se cambia de estado si es pulsada la pantalla o un boton
                if(estadoActual()==pulsada || buttonEstado()==button_iz || buttonEstado()==button_dr){
                    Lcd_Clr();
                    cambiarBoton(button_none);
                    cambiarEstado(no_pulsada);
                    timer2_empezar();
                    mostrarTiempo(timer2_leer());
                    mostrarTiempoPatron(tiempoPatron);
                    mostrarFila(xpos);
                    mostrarColumna(ypos);
                    mostrarTablero();
                    rellenarTablero();
                    PulseJugar();
                    colocarPosibleFicha(xpos,ypos);
                    tiempoParpadeo=timer2_leer();
                    estadoPrograma=elegirMovimiento;
                }
                break;
            //Estado elegir movimiento de nuestra ficha
            case elegirMovimiento:
                cambiarEstado(no_pulsada);
                cambiarBoton(button_none);
                //Si pulsa en el centro se cambia a estado de esperar2s
                if(estadoActual()==pulsada){
                    cambiarEstado(no_pulsada);
                    if(pulsaCentro()==1) {
                        estadoPrograma=esperar_2s;
                        Lcd_Clr();
                        cambiarBoton(button_none);
                        mostrarTiempo(timer2_leer());
                        mostrarTiempoPatron(tiempoPatron);
                        mostrarFila(xpos);
```

```
                mostrarColumna(ypos);
                mostrarTablero();
                rellenarTablero();
                PulseCancelar();
                colocarPosibleFicha(xpos,ypos);
                tiempo2s=timer2_leer();
                tiempoParpadeo=timer2_leer();
            }
        }
        //Se gestiona el boton pulsado, eliminando los rebotes
        gestion(&boton);
        //Si el boton es derecho, se mueve verticalmente
        if(boton==button_dr){
            Lcd_Clr();
            cambiarBoton(button_none);
            mostrarTiempo(timer2_leer());
            mostrarTiempoPatron(tiempoPatron);
            mostrarFila(xpos);
            mostrarColumna(ypos);
            mostrarTablero();
            PulseJugar();
            rellenarTablero();
            if(xpos==7) {
                xpos=0;
            }
            else {
                xpos+=1;
            }
            colocarPosibleFicha(xpos,ypos);
        }
        //Si el boton es izquierdo, se mueve horizontalmente
        else if(boton==button_iz){
            Lcd_Clr();
            cambiarBoton(button_none);
            mostrarTiempo(timer2_leer());
            mostrarTiempoPatron(tiempoPatron);
            mostrarFila(xpos);
            mostrarColumna(ypos);
            mostrarTablero();
            PulseJugar();
            rellenarTablero();
```

```
            if(ypos==7) {
                ypos=0;
            }
            else {
                ypos+=1;
            }
            colocarPosibleFicha(xpos,ypos);
        }
        //Cada medio segundo, se produce el parpadeo de la ficha
        if(timer2_leer()-tiempoParpadeo>=500000) {
            LcdClrRect(220,15,315,100,WHITE);
            mostrarTiempo(timer2_leer());
            mostrarTiempoPatron(tiempoPatron);
            mostrarFila(xpos);
            mostrarColumna(ypos);
            if (cambio==0){
                cambio=1;//Si el color es blanco
                parpadeo(xpos,ypos,cambio); //Ponemos la casilla negra
            }
            else{
                cambio=0;                       //Color = blanco
                parpadeo(xpos,ypos,cambio); //Ponemos la casilla blanca
            }
            tiempoParpadeo=timer2_leer();
        }
        break;
    //Estado esperar 2 segundos para aceptar o cancelar movimiento
    case esperar_2s:
        cambiarEstado(no_pulsada);
        //Si pasan los 2 segundos se cambia de estado para realizar el movimiento
        if(timer2_leer()-tiempo2s>=2000000) {
            estadoPrograma=reversi;
        }
        cambiarEstado(no_pulsada);
        //Si es pulsada la pantalla o un boton se vuelve al estado anterior para seguir realizando el movimiento
        if(estadoActual()==pulsada || buttonEstado()==button_iz || buttonEstado()==button_dr) {
            estadoPrograma=elegirMovimiento;
            Lcd_Clr();
            cambiarBoton(button_none);
            cambiarEstado(no_pulsada);
            mostrarTiempo(timer2_leer());
```

```
                mostrarTiempoPatron(tiempoPatron);
                mostrarFila(xpos);
                mostrarColumna(ypos);
                mostrarTablero();
                PulseJugar();
                rellenarTablero();
                colocarPosibleFicha(xpos,ypos);
                tiempoParpadeo=timer2_leer();
            }
            //Cada medio segundo, se produce el parpadeo de la ficha
            if(timer2_leer()-tiempoParpadeo>=500000) {
                LcdClrRect(220,15,315,100,WHITE);
                mostrarTiempo(timer2_leer());
                mostrarTiempoPatron(tiempoPatron);
                mostrarFila(xpos);
                mostrarColumna(ypos);
                if (cambio==0){
                    cambio=1;//Si el color es blanco
                    parpadeo(xpos,ypos,cambio); //Ponemos la casilla negra
                }
                else{
                    cambio=0;                       //Color = blanco
                    parpadeo(xpos,ypos,cambio); //Ponemos la casilla blanca
                }
                tiempoParpadeo=timer2_leer();
            }
            break;
        //Estado reversi, se realiza el movimiento seleccionado anteriormente
        case reversi:
            //Si no puede colocar, se pasa turno
            if(fichaPosicionCandidatas(xpos,ypos)!=1 || puedoColocar(xpos,ypos)!=1){
                xpos=8;
                ypos=8;
            }
            estadoPartida=reversi8(xpos,ypos,&tiempoPatron,&fin);
            //Si no se ha terminado todavia se vuelve al estado elegirMovimiento
            if(fin==0){
                    Lcd_Clr();
                    mostrarTiempo(timer2_leer());
                    mostrarTiempoPatron(tiempoPatron);
                    mostrarFila(xpos);

                    mostrarColumna(ypos);
                    mostrarTablero();
                    PulseJugar();
                    rellenarTablero();
                    estadoPrograma=elegirMovimiento;
                    xpos=0;
                    ypos=0;
                    colocarPosibleFicha(xpos,ypos);
            }
            //En caso de fin, se muestra por pantalla el resultado pudiendo empezar otra partida
            else{
                estadoPrograma=inicio;
                Lcd_Clr();
                if(estadoPartida==1){
                    BlancasGanan();
                }
                else if(estadoPartida==2){
                    NegrasGanan();
                }
                else{
                    Empate();
                }
                reversi_main();
                xpos=0;
                ypos=0;
            }
            break;
    }
  }
}
```