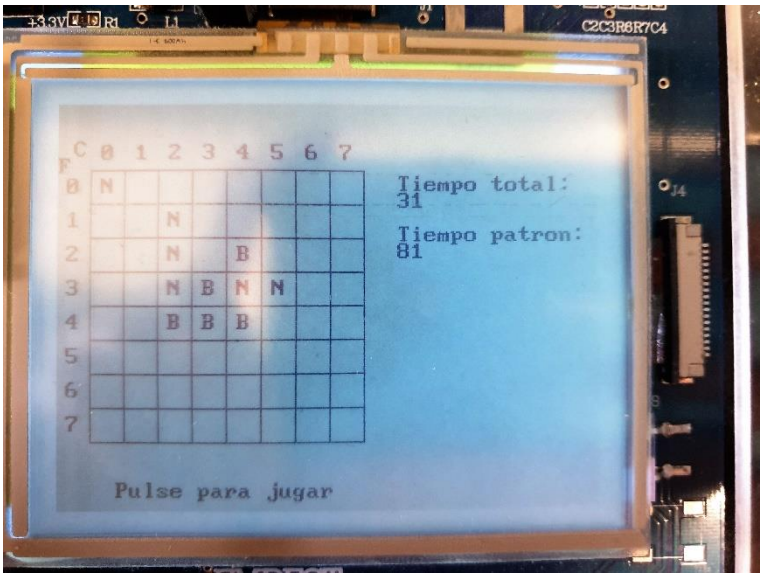
DISEÑO DE UNA PLATAFORMA INDEPENDIENTE



12/01/2019

Prácticas 2 y 3

Andrés Gavín Murillo 716358

Eduardo Gimeno Soriano 721615

Proyecto Hardware

Ingeniería Informática

Universidad de Zaragoza

DISEÑO DE UNA PLATAFORMA INDEPENDIENTE

PRÁCTICAS 2 Y 3

ÍNDICE

RESUMEN

En esta memoria se presenta el desarrollo de las prácticas 2 y 3 de la asignatura Proyecto Hardware, las cuales están centradas en la interacción con el usuario con la placa *Embest S3CE40 EVB*, concretamente se han utilizado los botones, LEDs, 8led, LCD y *Touch Pad*. Los objetivos son que el usuario pueda introducir una jugada mediante el uso de los botones (la forma en la que se usan estos botones varía entre las prácticas), poder interactuar y ver el resultado de su jugada en la pantalla.

La práctica 2 está centrada en la utilización de los botones de la placa para introducir una jugada, el botón izquierdo se utiliza para aumentar el valor de fila y columna (cada una en su debido momento) y el derecho para confirmar dicho valor, en el 8led se visualiza el valor según se incrementase. Esta funcionalidad se implementa en dos módulos, botones_antirebotes y jugada_por_botones. En el primero de ellos se gestiona la pulsación del botón, este al ser un dispositivo mecánico produce rebotes y si no son gestionados provocaría un comportamiento erróneo. En el segundo se gestiona la jugada en sí, se utiliza el módulo anterior para detectar una pulsación y se incrementa el valor correspondiente según el estado en que se encuentre (ambos módulos usan máquinas de estados).

Detectar fallos y saber qué los provoca en estos módulos es una tarea compleja, por ello, se han desarrollado una serie de módulos que faciliten esta tarea. Un módulo encargado del tratamiento de excepciones, llamado Excepcion, se encarga de capturar las excepciones *Data Abort*, *Undefined Instruction* y *Software Interrupt* e informar cuál de ellas tres ha sido provocada utilizando el 8led, además de introducir diversa información en una pila de depuración. Esta pila es otro módulo desarrollado, llamado Pila, que consiste en una lista circular ubicada en memoria después de las distintas pilas de los modos del procesador. El módulo Latido permite saber si la placa se encuentra activa y ejecutando el programa mediante el parpadeo del led izquierdo a 2 Hz. La función principal del Reversi8 ha sido modificada para incorporar la gestión de los nuevos periféricos y estructurar la forma de jugar. Por último, el módulo Simulación permite abstraer el hardware, sustituyendo el comportamiento de los periféricos, para poder trabajar sin tener la placa físicamente.

Como resultados de la realización de esta práctica se ha obtenido un juego, el cual permite al usuario introducir un movimiento mediante el uso de los botones, visualizar los valores en el 8led y la fácil detección de posibles errores.

La práctica 3 está centrada en que el juego funcione autónomamente en la placa, para ello el usuario introduce el movimiento a realizar utilizando los botones, se utilizan los mismos módulos que en la práctica anterior, aunque ahora jugada_por_botones (renombrado como Jugada) engloba los distintos pasos de una jugada, además de incrementar fila o columna se encarga de gestionar el tablero y la ficha del usuario en la pantalla. Para ello hace uso del módulo Pantalla, el cual ofrece una serie de operaciones, como mostrar el estado actual de la partida, la ayuda inicial, cancelar el movimiento, mostrar una serie de mensajes especiales al finalizar la partida, etc. Este módulo hace uso de los módulos Lcd y Tp, los cuales interactúan directamente con la placa. Para que el sistema sea completamente autónomo se ha *flasheado* el juego en la placa para ejecutarlo sin la necesidad del simulador, para ello se ha modificado el fichero *44binit* con el código proporcionado. Por último, al inicio de la ejecución se cambia a modo usuario. Como resultados de la realización de esta práctica se ha obtenido un juego el cual funciona autónomamente y permite al usuario una interacción completa con la placa.

Durante el desarrollo de ambas prácticas se ha utilizado la versión C_C de patrón_volteo, ya que es la que mejores resultados proporcionó en la práctica 1.

INTRODUCCIÓN

Este trabajo trata de ampliar el sistema previamente desarrollado en la Práctica 1, que consistía en el juego de Reversi8 jugado únicamente en *RAM*, es decir, para visualizar el juego y colocar una nueva ficha, había que ver y cambiar directamente la *RAM*.

En estas dos prácticas (2 y 3) se ha conseguido crear un sistema de juego de Reversi8 completamente autónomo para las placas *Embest S3CEV40*. Es decir, se ha cargado la imagen del juego desarrollado directamente en la memoria *Flash* de la placa de tal manera que, al encenderla, se carga directamente el juego, y, por medio de los botones y la pantalla, se puede jugar correctamente.

OBJETIVOS

Los objetivos propuestos para cumplir con el sistema de juego autónomo previamente mencionado son los siguientes:

- **Tratar el sistema de manera modular.** Consiste en la diferenciación de cada componente del sistema (controlador para dispositivo hardware, autómata para la gestión de diferentes eventos, comportamiento del juego, …). De esta manera, se puede portar y entender el sistema con mayor facilidad, además de poder simular los dispositivos hardware y así trabajar en entornos diferentes.
- **Tratar los estados de excepción.** Capturar las excepciones que se pueden producir en la ejecución (*Data Abort, Undefined, SWI*), para evitar situaciones inconsistentes en el sistema. Cuando se produzca una excepción, se mostrará por el 8LED su código de excepción.
- **Desarrollar una pila de depuración.** Almacenar los eventos producidos durante el juego, ya sean pulsaciones en los botones o excepciones producidas, para poder depurar el sistema con facilidad.
- **Desarrollar un latido LED.** Encender y apagar el LED de la izquierda a una frecuencia de 2 Hz cuando el juego se está ejecutando de manera normal. Con esto, se visualiza cuando el sistema funciona correctamente, o cuando ha comenzado a fallar.
- **Utilizar los botones.** Realizar las jugadas por medio de la pulsación de los dos botones, el izquierdo incrementa la columna de la ficha a colocar y el derecho la fila, siendo 0 la mínima a introducir y 7 la máxima. Para el correcto funcionamiento de los botones, se deben tratar los rebotes generados en cada pulsación.
- **Integrar la pantalla.** Visualizar cada estado del juego en la pantalla, diferenciando 3 principalmente: la pantalla de ayuda inicial (donde se espera a que el usuario inicie la partida pulsando la pantalla o los botones), la pantalla de jugada (donde se muestra el tablero y el usuario introduce la ficha por medio de los botones y la pantalla) y la pantalla de partida terminada (donde se muestra el resultado de la partida y se vuelve a la pantalla de ayuda).
- **Medir los tiempos de juego.** Cronometrar el tiempo total de la partida (en segundos) y el tiempo de ejecución de la función crítica de patrón volteo (en microsegundos), por medio del timer0 y el timer2 respectivamente, y mostrando los resultados por pantalla.
- **Comprobar que la ficha se introduce de manera correcta.** Realizar la comprobación de si la ficha que se desea introducir cumple con las reglas del juego, y en el caso de que no las cumpla denegar el movimiento.

- **Cambiar a modo usuario.** Comenzar en modo supervisor inicializando todos los dispositivos del sistema y, a continuación, cambiar a modo usuario para ejecutar el juego Reversi8.
- **Cargar el juego en memoria Flash.** Ejecutar el sistema de manera autónoma, de tal manera que cada vez que se encienda la placa se cargue directamente el juego. Para ello, hay que generar una imagen binaria del juego y cargarla en la placa.

METODOLOGÍA

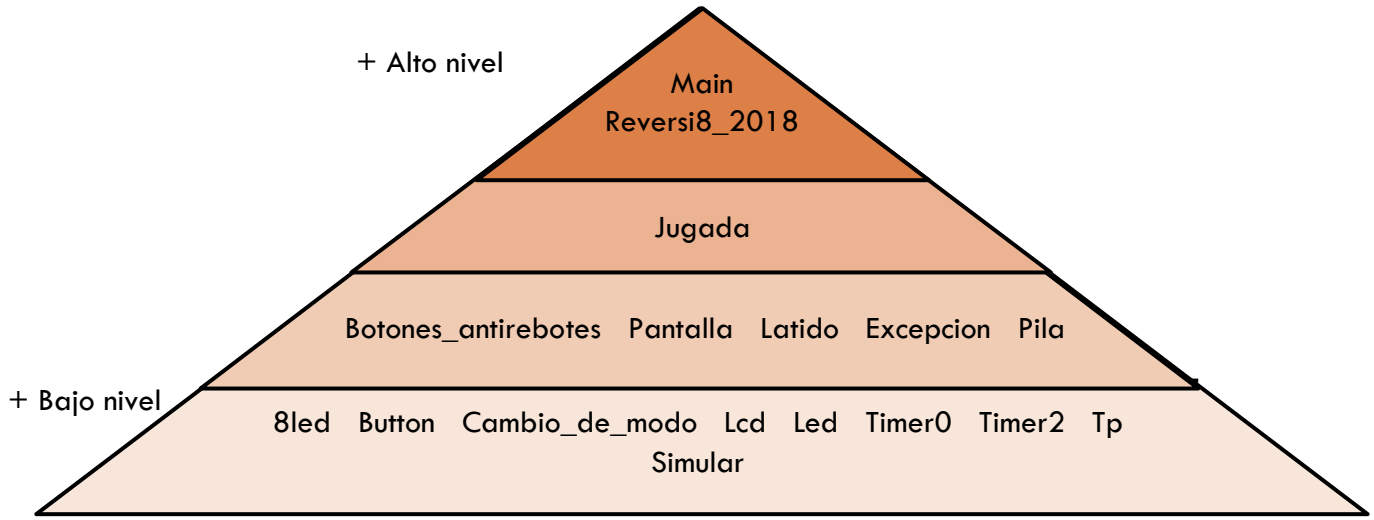
Para el desarrollo de este proyecto, se ha comenzado estudiando los fuentes proporcionados de las bibliotecas de la placa, en concreto los de tratamiento de interrupciones y periféricos. A continuación, se ha configurado la biblioteca para el timer0 y se ha definido el latido LED del juego. Después se han tratado el resto de los periféricos (solucionando los problemas explicados más adelante, como los rebotes) y las excepciones.

Ya con todos los controladores hardware funcionando, se ha procedido a crear una biblioteca de simulación para poder trabajar en diferentes entornos sin necesidad de usar la placa. Tras esto, se ha establecido toda la lógica de Reversi8 en una biblioteca aparte (reversi8_2018) y su sistema de juego en otro (jugada), y un fichero principal con la función *main* que inicializa todos los periféricos y pasa el control a Reversi8.

Finalmente se ha establecido el modo usuario antes de llamar a Reversi8, y, tras probar que funcionaba correctamente, se ha generado un binario para poder cargarlo en la memoria *Flash* de la placa y jugar de manera autónoma.

Estructura del proyecto

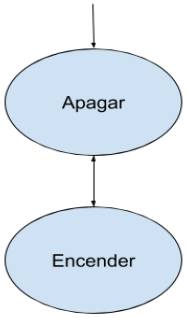
El proyecto está compuesto por distintos módulos enlazados de más alto a más bajo nivel, quedando una estructura como la que se muestra a continuación:



- **Main.** (main.c) Inicialización de los dispositivos, cambio a modo usuario y ejecución de Reversi8.
- **Reversi8_2018.** (reversi8_2018.h y reversi8_2018.c) Lógica del juego.
- **Jugada.** (jugada.h y jugada.c) Gestión de los eventos para el correcto funcionamiento del juego.
- **Botones_antirebotes.** (botones_antirebotes.h y botones_antirebotes.c) Gestión del estado de los botones, solucionando los posibles rebotes.
- **Pantalla.** (pantalla.h y pantalla.c) Gestión del estado de la pantalla y el touchpad, solucionando los posibles rebotes de este.
- **Latido.** (latido.h y latido.c) Gestión del apagado y encendido del LED izquierdo.
- **Excepcion.** (excepción.h y excepción.c) Tratamiento de las excepciones.
- **Pila.** (pila.h y pila.c) Tratamiento de la pila de depuración.
- **8led.** (8led.h y 8led.c) Controlador del 8led.
- **Button.** (button.h y button.c) Controlador de los botones derecho e izquierdo.
- **Cambio_de_modo.** (cambio_de_modo.asm) Cambio a modo usuario.
- **Lcd.** (lcd.h y lcd.c) Controlador de la pantalla LCD.
- **Led.** (led.h y led.c) Controlador de los LEDs.
- **Timer0.** (timer0.h y timer0.c) Controlador del timer0.
- **Timer2.** (timer2.h y timer2.c) Controlador del timer2.
- **Tp.** (tp.h y tp.c) Controlador del touchpad.
- **Simular.** (simular.h y simular.c) Biblioteca de simulación de los dispositivos hardware.

Gestión de entrada/salida

En la gestión de entrada/salida, se han configurado los LEDs para establecer un latido, y los botones y la pantalla para establecer una jugada de la siguiente manera:

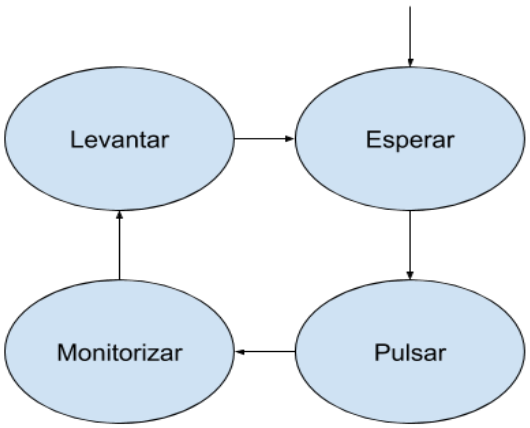


Latido. Se ha configurado el LED1 para que parpadee con una frecuencia de 2Hz, es decir, que se enchufe y se apague 2 veces por segundo. Para ello se establece un cambio de estado cada 250 milisegundos.

Botones_antirebotes. Se han configurado los botones para que cada vez que uno de ellos haya sido pulsado se ejecute una función de *callback* y se obtenga el estado actual de los botones.

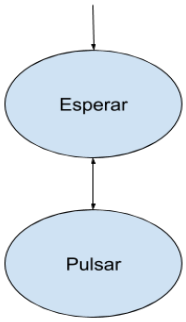
Para evitar los rebotes al pulsar, una vez ha sido pulsado un botón, se cambia a un estado pulsar, se deshabilitan las interrupciones y se establece un retardo *trp*, después se cambia a un estado de monitorización en el que, una vez se haya esperado el retardo, se monitoriza el botón hasta que se levante. Cuando el botón ha sido levantado, se establece un retardo *trd* y se cambia a un estado levantar en el que, una vez se haya esperado el retardo, se vuelven a habilitar las interrupciones y se reestablece el estado de espera inicial.

Cada vez que se produce una interrupción debida a un botón, entre otras cosas, se almacena en la pila de depuración el tiempo y el botón donde se ha producido la interrupción. Así, se han calculado los retardos *trp* y *trd*, es decir, sin deshabilitar las interrupciones, se han pulsado y levantado los botones distintas veces y comprobado en la pila de depuración los rebotes producidos.



Los retardos son una estimación de tiempo (sobreestimación) que se han calculado restando el tiempo del último rebote con el del primero, para que así mientras se producen los rebotes, no salten más interrupciones.

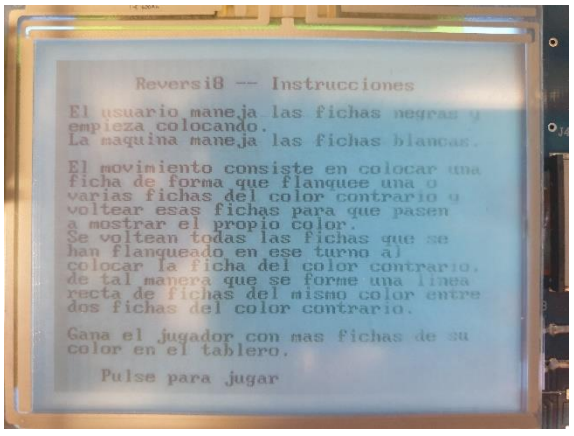
Pantalla. Se ha configurado el *touchpad* para que cada vez que sea pulsado se almacene en una variable del módulo Pantalla, y pueda ser leída con una función que devuelve el estado del *touchpad*.

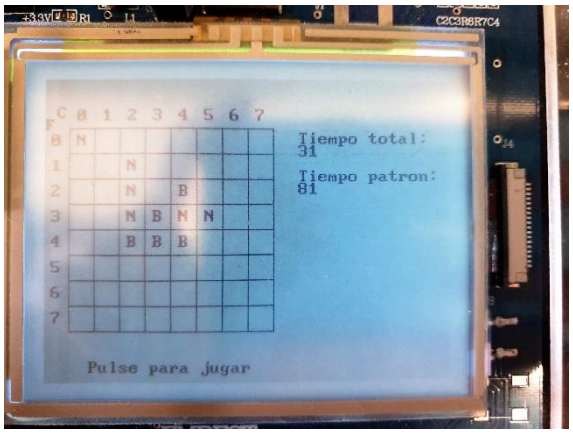


Para evitar los rebotes al pulsar, una vez se ha pulsado la pantalla, se establece un retardo (calculado de manera similar al de los botones) y se cambia a un estado pulsar. Una vez se haya esperado el retardo, se reestablece el estado inicial.

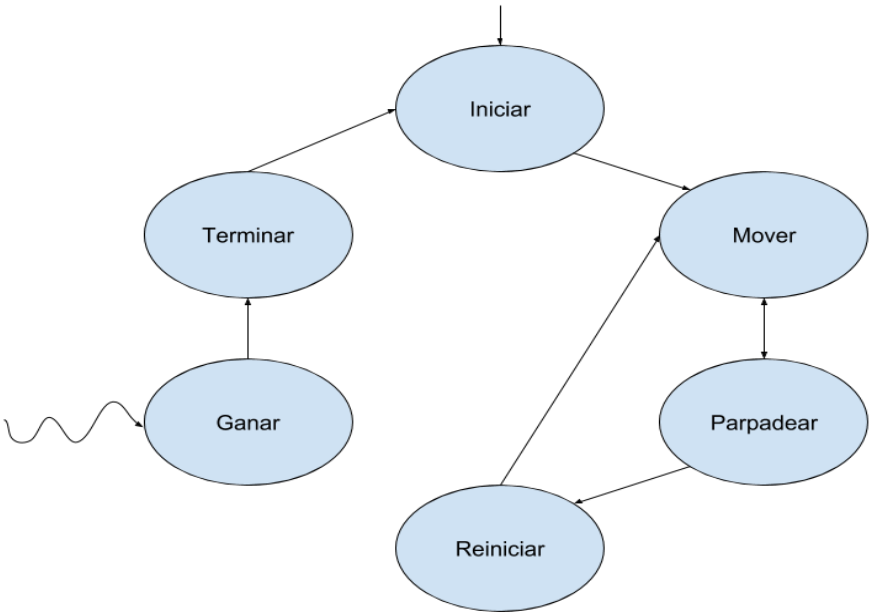
Para comprobar si el *touchpad* ha sido pulsado, la función *pantalla_pulsada()* devuelve si la pantalla ha sido pulsada desde la última vez que se ha llamado a la función, y reestablece el estado de la pantalla.

También se han configurado diferentes funciones para imprimir y borrar por pantalla, entre las que se encuentran una función que imprime una pantalla de ayuda (*pantalla_help*), otra que imprime el tablero con las fichas colocadas y el tiempo de patrón (*pantalla_mostrar_partida*), otra que se encarga de actualizar la ficha que se está moviendo (*pantalla_casilla*), otra que actualiza el tiempo de partida (*pantalla_tiempo*), y otras que muestran distintos mensajes (*pantalla_pulsar_continuar*, *pantalla_pulsar_cancelar*, *pantalla_ganador*).





Jugada. Se ha configurado un módulo Jugada que se encarga de unir la lógica del juego (Reversi8_2018) con los periféricos (pantalla, botones). Para ello se ha establecido el siguiente autómata de control:



- **Iniciar.** Estado inicial de espera hasta que comience una nueva partida. Mientras se permanece en este estado aparece la pantalla de ayuda. Cuando se pulsa la pantalla o uno de los botones, se reinician las variables de juego, se muestra la pantalla de la partida con el tablero y los tiempos, y se cambia al estado Mover.
- **Mover.** Estado en el que se mueve por el tablero la ficha a introducir, el botón izquierdo incrementa las columnas y el derecho las filas (ambas entre 0 y 7). Cuando se pulsa la pantalla, se cambia al estado Parpadear. Además del tablero, también se muestran los tiempos de patrón volteo y de partida.
- **Parpadear.** Estado en el que se espera durante 2 segundos mientras la ficha que se desea introducir parpadea. Si se pulsa alguno de los botones o la pantalla, se cancela el movimiento y se vuelve al estado Mover. Si acaba el tiempo de espera sin que haya habido ninguna pulsación, se confirma el movimiento y se cambia al estado Reiniciar. Además del tablero, también se muestran los tiempos de patrón volteo y de partida.
- **Reiniciar.** Estado en el que se reinician las variables de la jugada, se muestra la pantalla de la partida con el tablero, las fichas confirmadas y los tiempos, y se cambia al estado Mover.
- **Ganar.** Estado en el que se muestra la pantalla de fin de partida con la información del ganador y se cambia al estado Terminar. Para llegar a este estado se debe llamar a la función *jugada_ganador()*.
- **Terminar.** Estado en el que se espera durante 5 segundos mientras se muestra la pantalla de fin de partida. Cuando termina el tiempo de partida se reinicia el juego volviendo al estado Iniciar.

Gestión de excepciones

Se han capturado las excepciones *Data Abort*, *Undefined Instruction* y *Software Interrupt* (módulo Excepcion) de tal manera que, cuando se lance una de ellas, se ejecutará la función *excepcion_handler()*. En esta función se calcula la dirección de la instrucción que ha generado la excepción de la siguiente manera:

1. Se lee el registro LR donde se almacena la dirección de retorno, en este caso, el valor del PC en el momento en el que se ha producido la excepción.
2. Se lee el registro CPSR donde se almacena el código de la excepción que se ha producido.
3. Se comprueba la excepción que se ha producido.
    a) Si se ha producido una *Data Abort*, significa que se ha generado en la etapa EX del procesador, por lo que para calcular la dirección en la que se ha producido (si no acaba de ocurrir un salto) hay que restar 2 instrucciones (de 32 bits, como un *int*) a la dirección obtenida en el registro LR. Además, se muestra una D en el 8led.
    b) Si se ha producido una *Undefined Instruction*, significa que se ha generado en la etapa IF del procesador, por lo que el registro LR contiene la dirección de la instrucción que la ha producido. Además, se muestra una E en el 8led.
    c) Si se ha producido una *Software Interrupt*, significa que se ha generado en la etapa ID del procesador, por lo que para calcular la dirección en la que se ha producido hay que restar 1 instrucción (de 32 bits, como un *int*) a la dirección obtenida en el registro LR. Además, se muestra una A en el 8led.

A continuación, se almacena en la pila de depuración el tipo de excepción generada y la dirección de la instrucción que la ha provocado. Finalmente se ejecuta un bucle infinito para cancelar la ejecución del juego.

Para comprobar la correcta gestión de las excepciones, se han forzado excepciones de los 3 tipos mencionados, en los ficheros *ex_data_abort.asm*, *ex_swi.asm* y *ex_undefined.asm*.

Flasheado del código

Se ha generado un binario del juego para poder ser cargado en la memoria *Flash* de la placa, y que sea persistente a los reinicios. Para ello, se ha modificado el fichero *44binit.asm* con el siguiente fin:

1. Se recalculan las direcciones de memoria para poder utilizar desde la dirección 0x0 de la *RAM*, ya que en Eclipse se utilizaban desde la dirección 0xc000000.
2. Se copia el contenido de la memoria *Flash* a la *RAM*.
3. Se inicializa con 0s el resto de la memoria *RAM* que no haya sido copiada.

A continuación, para generar el binario correspondiente (una vez compilado desde Eclipse) se ha lanzado el siguiente comando desde el ordenador del laboratorio:

```
arm-none-eabi-objcopy -O binary reversi.elf reversi.bin
```

Finalmente, se ha copiado el binario a la memoria Flash de la placa con el siguiente comando:

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

Problemas encontrados

Los principales problemas han surgido al utilizar la configuración O3, ya que el compilador eliminaba partes de código importantes. En el módulo tp, en la función encargada de gestionar la interrupción se utiliza la función DelayTime, la cual es una cuenta atrás implementada mediante un bucle. Esta función era descartada por el compilador, lo cual provocaba problemas a la hora de utilizar el Touch Pad. Para solucionarlo se cambió por la función Delay. Además, se declararon ciertas variables como volatile, como por ejemplo las variables encargadas de guardar el estado de los distintos autómatas, con el mismo propósito.

Respecto a la pantalla de la placa, se tuvieron algunos problemas en cuanto al solapamiento de distintos elementos. Esto obligó a guardar el elemento mostrado en una variable para cuando llegara el momento de actualizar escribir primero dicho valor en blanco y de esta forma "borrarlo".

En cuanto a flashear el juego en la placa, se tuvieron algunos problemas debido a que no se colocaron algunas partes del código proporcionado en el lugar incorrecto en el fichero 44binit y esto provocaba que aún se configuraran algunos aspectos referentes al simulador.

RESULTADOS

Como resultado de la realización de estas dos prácticas se ha obtenido un sistema completamente autónomo, el cual permite al usuario introducir su movimiento mediante el uso de los botones, utilizando el izquierdo para las filas y el derecho para las columnas, confirmar su movimiento tocando la pantalla (Touch Pad), pudiendo cancelarlo durante dos segundos, y viendo el resultado de dicho movimiento (LCD). El juego funciona completamente en la placa sin necesidad de uso del simulador y se ejecuta en modo usuario. Al comienzo de cada partida el usuario es informado de las reglas del juego y al finalizar se informa de quien es el ganador de la partida. De esta forma se consigue una experiencia plena de uso, sumado además a una buena jugabilidad.

CONCLUSIONES

Tras la realización de estas dos prácticas se entiende la gran importancia que tiene el correcto funcionamiento de los distintos dispositivos que pueden intervenir en un sistema, tanto para que el sistema vaya bien, como para la buena experiencia de uso que pueda obtener el usuario. Además, realizar máquinas de estados claras y concisas que faciliten la resolución de ciertas tareas.

Destacar también la importancia de realizar una buena documentación del código y el uso de las distintas documentaciones para la fácil comprensión de los elementos de, en este caso, la placa a utilizar como parte fundamental de la asignatura.

Estimación de horas:

La siguiente tabla muestra una estimación de las horas dedicadas por alumno:

- Estudio de los fuentes: 6h
- Diseño y programación: 45h
- Depuración y ajustes: 60h
- Memoria: 8h

BIBLIOGRAFÍA Y REFERENCIAS

Documentos consultados a lo largo de la realización de las dos prácticas

- ARM-Interrupts
- um_s3c44box
- P3-LEC
- P4-LEC
- Entradasalida
- p2-ec
- p3-ec
- P6_ARM

ANEXO

44binit.asm

```
[…]
    ldr          r0,=(SMRDATA-0xc000000) /* Se resta 0xc000000 ya que utilizando eclipse, las direcciones comenzaban a partir de esa dirección, pero al flashear la imagen de reversi8 en la placa, se utiliza desde la dirección 0 */
    ldmia   r0,{r1-r13}
    ldr          r0,=0x01c80000
    stmia   r0,{r1-r13}

    LDR r0,=0x0
    LDR r1,=Image_RO_Base
    LDR r3,=Image_ZI_Base
LoopRw: /* Copia el contenido de las antiguas direcciones de Eclipse a las nuevas de la placa */
    cmp         r1, r3
    ldrcc       r2, [r0], #4
    strcc       r2, [r1], #4
    bcc         LoopRw

    LDR r0, =Image_ZI_Base
    LDR r1, =Image_ZI_Limit
    mov r3, #0
LoopZI: /* Inicializa la memoria RAM con ceros */
    cmp r0, r1
    strcc r3, [r0], #4
    bcc LoopZI

[…]

    ldr pc,=Main

[…]

.equ   DebugStack,  _ISR_STARTADDRESS-0xf00+256*7    /* Define la primera dirección para la pila de Debug, que podrá almacenar 128 palabras */

[…]
```

==========

cambio_de_modo.asm

```
/* Cambio a modo User */

cambio_de_modo:
       .equ MODEMASK, 0x1f /* Para selección de M[4:0] */
       .equ USERMODE, 0x10 /* Código de modo User */
       .equ UserStack,     0xc7ff000

       mov r5, lr

       mrs r0, cpsr /* Llevamos el registro de estado a r0 */
```

```
	bic r0, r0, #MODEMASK /* Borramos los bits de modo de r0 */
	orr r1, r0, #USERMODE /* Añadimos el código del modo User y copiamos en r1 */
	msr cpsr_cxsf, r1 /* Escribimos el resultado en el registro de estado, cambiando de éste los bits de del campo de control, de extensión, de estado y los de flag. */
	ldr sp, =UserStack /* Una vez en modo User copiamos la dirección de comienzo de la pila */

	bx r5
```

==========

ex_data_abort.asm

```
/* Provoca una excepción data abort */
ex_data_abort:
	ldr r0, =0x0a333333 /* Al ser impar, no es un dato alineado y provoca la excepción */
	str r0, [r0]
	mov pc, lr
```

==========

ex_swi.asm

```
/* Provoca una excepción swi */
ex_swi:
	swi #0 /* Fuerza la excepción */
	mov pc, lr
```

==========

ex_undefined.asm

```
/* Provoca una excepción undefined */
ex_undefined:
	.word 0xE6000010 /* Dirección no definida */
	mov pc, lr
```

==========

main.c

```
void Main(void)
{
	/* Inicializa controladores */
	sys_init();		// Inicialización de la placa, interrupciones y puertos
	button_iniciar();	// inicializamos los pulsadores.
	D8Led_init();		// inicializamos el 8led
	timer0_inicializar();
	timer2_inicializar();
	init_excepcion();
	Lcd_Init();
	TS_init();

	/* Inicializa dispositivos */
```

```
	leds_off();
	pila_init();
	timer0_empezar();
	timer2_empezar();
	botones_antirebotes_init();
	latido_init();
	jugada_init();

	/* Descomentar para provocar excepciones */
	//ex_undefined();
	//ex_data_abort();
	//ex_swi();

	cambio_de_modo(); // Cambio a modo usuario

	reversi_main();
}
```

==========

reversi8_2018.c

```
[…]

/* Calcula el patrón de volteo con la función más eficiente en O3 y el tiempo que ha
tardado */
int patron_volteo_test(char tablero[][DIM], int *longitud, char FA, char CA, char SF,
char SC, char color)
{
	// En 03 es mejor el compilador
	tiempo_patron = timer2_leer();
	int result = patron_volteo(tablero, longitud, FA, CA, SF, SC, color);
	tiempo_patron = timer2_leer() - tiempo_patron;
	return result;
}

[…]

/* Inicializa el tablero de candidatas */
void reversi_iniciar_candidatas(char candidatas[][DIM]) {
	int i, j;
	for (i = 0; i < DIM; i++) {
		for (j = 0; j < DIM; j++) {
			candidatas[i][j] = NO;
		}
	}
}

/* Copia el tablero de entrada al tablero de salida */
void reversi_copiar_tablero(char entrada[][DIM], char salida[][DIM]) {
	int i, j;
	for (i = 0; i < DIM; i++) {
		for (j = 0; j < DIM; j++) {
			salida[i][j] = entrada[i][j];
```

```
			}
		}
}

/* Comprueba si la partida ha terminado, y en su caso devuelve también el ganador. La
partida ha terminado si sólo hay fichas de un color, o si no hay casillas vacías en el
tablero */
char reversi_terminado(char tablero[][DIM], char *ganador) {
	int blancas = 0, negras = 0;
	char fin = PARTIDA_ACABADA;

	int i, j;
	for (i = 0; i < DIM; i++) {
		for (j = 0; j < DIM; j++) {
			if (tablero[i][j] == CASILLA_VACIA)
				fin = PARTIDA_SIGUE;
			else if (tablero[i][j] == FICHA_BLANCA)
				blancas++;
			else
				negras++;
		}
	}

	if (blancas == 0 || negras == 0)
		fin = PARTIDA_ACABADA;

	/* Gana el jugador que más fichas tenga */
	if (blancas > negras)
		*ganador = FICHA_BLANCA;
	else
		*ganador = FICHA_NEGRA;

	return fin;
}

/* Función principal del juego, donde se calculan los movimientos */
void reversi_main() {
	// Variables utilizadas
	uint32_t ahora;
	volatile char fila, columna;
	char reiniciar = PARTIDA_ACABADA;
	int done;	// la máquina ha conseguido mover o no
	int move = 0; // el humano ha conseguido mover o no
	int blancas, negras; // número de fichas de cada color
	char ganador;
	char __attribute__ ((aligned (8))) candidatas[DIM][DIM];
	char __attribute__ ((aligned (8))) tablero_aux[DIM][DIM];

	// Iniciar tablero
	reversi_iniciar_candidatas(candidatas);
	init_table(tablero, candidatas);

	while(1) {
		ahora = timer0_leer();
```

```
            latido_gestionar(ahora);

            // Lógica del juego
            if (jugada_gestionar(ahora, tiempo_patron, &fila, &columna, tablero) != 0) {
             // si la fila o columna son 8 asumimos que el jugador no puede mover
             if (((fila) < DIM) && ((columna) < DIM) && tablero[fila][columna] ==
CASILLA_VACIA)
             {
                  reversi_copiar_tablero(tablero, tablero_aux);
                tablero_aux[fila][columna] = FICHA_NEGRA;
                move = actualizar_tablero(tablero_aux, fila, columna, FICHA_NEGRA);

                if (move == PATRON_ENCONTRADO) {
                 reversi_copiar_tablero(tablero_aux, tablero);
                       actualizar_candidatas(candidatas, fila, columna);
                }
             }

             // escribe el movimiento en las variables globales fila columna
             done = elegir_mov(candidatas, tablero, &fila, &columna);
             if (done != -1)
             {
                tablero[fila][columna] = FICHA_BLANCA;
                actualizar_tablero(tablero, fila, columna, FICHA_BLANCA);
                actualizar_candidatas(candidatas, fila, columna);
             }

             reiniciar = reversi_terminado(tablero, &ganador);

                  if (reiniciar == PARTIDA_ACABADA) {
                        reiniciar = PARTIDA_SIGUE;

                        // Iniciar tablero
                        reversi_iniciar_candidatas(candidatas);
                        init_table(tablero, candidatas);
                        jugada_ganador(ganador);
                        tiempo_patron = 0;
                  }
            }
      }

      contar(tablero, &blancas, &negras); // Si blancas > negras, ganan las blancas,
sino las negras
}
```

==========

botones_antirebotes.c

```
typedef enum estado_automata {boton_esperar, boton_pulsar, boton_monitorizar,
boton_levantar} estado_automata;

#define retardo 20 // milisegundos
```

```
#define trp 100 // milisegundos
#define trd 100 // milisegundos

/*--- variables globales ---*/
static uint32_t siguiente; // milisegundos
static volatile estado_automata estado_auto;
static volatile estado_button estado_botones;

/* Función de callback, que es llamada cuando se ha producido un evento en los botones */
void callback(estado_button estado)
{
	// Se ha pulsado un botón
	estado_botones = estado;
	estado_auto = boton_pulsar;
}

/* Inicializa el módulo */
void botones_antirebotes_init()
{
	siguiente = 0;
	estado_botones = button_none;
	estado_auto = boton_esperar;
	button_empezar(&callback); // se inicializa button y estado_botones
}

/* Autómata principal del módulo, que gestiona las pulsaciones en los botones */
void botones_antirebotes_gestionar(uint32_t ahora) // milisegundos
{
	switch (estado_auto) {
	case boton_esperar:
		break;

	case boton_pulsar:
		// Se ha pulsado un botón
		if (estado_botones != button_none) {
			estado_auto = boton_monitorizar;
			siguiente = ahora + trp;
		}
		else {
			estado_botones = button_estado();
		}

		break;

	case boton_monitorizar:
		// Se eliminan los rebotes y se espera a que se levante el botón
		if (ahora >= siguiente) {
			estado_botones = button_estado();

			if (estado_botones == button_none) {
				estado_auto = boton_levantar;
				siguiente = ahora + trd;
			}
			else {
```

```
                        siguiente += retardo;
                }
        }

        break;

    case boton_levantar:
        // Se eliminan los rebotes y se espera habilitan interrupciones
        if (ahora >= siguiente) {
            estado_auto = boton_esperar;
            button_habilitar_interr(); // habilitar interrupcion button
        }

        break;
    }
}

/* Devuelve el estado de los botones */
boton botones_antirebotes_pulsado()
{
    switch (estado_botones) {
    case button_none:
        return b_ninguno;
    case button_dr:
        return b_derecho;
    case button_iz:
        return b_izquierdo;
    }
}
```

==========

button.c

```
// bit 7 es EINT7 (botón derecho) y bit 6 es EINT6 (botón izquierdo)
#define bit_botonD 7
#define bit_botonI 6
#define boton_pulsado 0

/*--- variables globales del módulo ---*/
static estado_button estado = button_none; // estado del botón
static function_ptr *cb_ptr = NULL; // puntero a función de callback

/* 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 ---*/
/* Devuelve el bit posición del número n */
int devuelve_bit(int n, int posicion) {
    return (n >> posicion) & 0x1;
}
```

```
/* Función de tratamiento de interrupción */
void Eint4567_ISR(void)
{
	// Comentar si se van a medir tiempos:
	rINTMSK |= BIT_EINT4567; // deshabilitamos interrupcion linea eint4567 en vector de mascaras

	int botones = rPDATG;

	if (devuelve_bit(botones, bit_botonD) == boton_pulsado) {
		estado = button_dr;
		push_debug(p_boton_der, timer0_leer());
	}
	else if (devuelve_bit(botones, bit_botonI) == boton_pulsado) {
		estado = button_iz;
		push_debug(p_boton_izq, timer0_leer());
	}
	else {
		estado = button_none;
		push_debug(p_boton_none, timer0_leer());
	}

	/* Finalizar ISR */
	rEXTINTPND = 0xf;		// borra los bits en EXTINTPND
	rI_ISPC   |= BIT_EINT4567;	// borra el bit pendiente en INTPND

	cb_ptr(estado); // ejecuta la función de callback con el estado del botón actual
}

/* Inicializa el módulo */
void button_iniciar(void)
{
	/* Configuracion del controlador de interrupciones. Estos registros están definidos en 44b.h */
	rI_ISPC    |= 0x200000;	// 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)

	/* 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 precaución, se vuelven a borrar los bits de INTPND y EXTINTPND */
	rI_ISPC    |= (BIT_EINT4567);
	rEXTINTPND = 0xf;
}

/* Asigna la función de callback y habilita las interrupciones */
```

```
void button_empezar(function_ptr *callback)
{
	cb_ptr = callback;
	rINTMSK &= ~(BIT_EINT4567); // habilitamos interrupción línea eint4567 en vector
de máscaras
}

/* Habilita las interrupciones */
void button_habilitar_interr() {
	/* Finalizar ISR */
	rEXTINTPND = 0xf;		// borra los bits en EXTINTPND
	rI_ISPC   |= BIT_EINT4567;	// borra el bit pendiente en INTPND

	rINTMSK &= ~(BIT_EINT4567); // habilitamos interrupción línea eint4567 en vector
de máscaras
}

/* Devuelve el estado de los botones */
estado_button button_estado()
{
	int botones = rPDATG;
	if (devuelve_bit(botones, bit_botonD) == boton_pulsado) estado = button_dr;
	else if (devuelve_bit(botones, bit_botonI) == boton_pulsado) estado = button_iz;
	else estado = button_none;

	return estado;
}
```

==========

excepcion.c

```
/*--- variables globales ---*/
#define cod_DA 23		// código modo de ejecución Data Abort
#define cod_U 27		// código modo de ejecución Undefined Instruction
#define cod_SWI 19		// código modo de ejecución Software Interrupt
#define led_DA 13		// Encender 8led por Data Abort
#define led_U 14		// Encender 8led por Undefined Instruction
#define led_SWI 10		// Encender 8led por Software Interrupt

static uint32_t dir_inst = 0;	// Instruction que ha causado la interrupción
/* declaración de función que es rutina de servicio de interrupción
 * https://gcc.gnu.org/onlinedocs/gcc/ARM-Function-Attributes.html */
void excepcion_handler(void) __attribute__((interrupt("DABORT")));
void excepcion_handler(void) __attribute__((interrupt("UNDEF")));
void excepcion_handler(void) __attribute__((interrupt("SWI")));


/* Devuelve el valor de los últimos 5 bits del registro CPSR */
unsigned int CPSR_leer(void)
{
	uint32_t retVal;
	asm("MRS %[res], cpsr" : [res] "=r" (retVal));
	asm("AND %[res], %[input], #0x1F" : [res] "=r" (retVal) : [input] "r" (retVal));
```

```
	return retVal;
}

/* Función de tratamiento de interrupción */
void excepcion_handler(void)
{
	asm("MOV %[res], lr" : [res] "=r" (dir_inst));

	long instr = dir_inst;

	volatile uint32_t cod_excp = CPSR_leer();
	volatile uint8_t id_pila = p_desconocida;

	/* Se calcula la dirección de la instrucción que ha generado la excepción, para
ello se ajusta restando direcciones de instrucción según el estado donde se haya
producido */
	if (cod_excp == cod_DA) {
		dir_inst -= 2;
		id_pila = p_ex_dabort;
		D8Led_symbol(led_DA);
	}
	else if (cod_excp == cod_U) {
		id_pila = p_ex_undef;
		D8Led_symbol(led_U);
	}
	else if (cod_excp == cod_SWI) {
		dir_inst--;
		id_pila = p_ex_swi;
		D8Led_symbol(led_SWI);
	}

	push_debug(id_pila, dir_inst);

	while(1) {}
}

/* Inicializa el módulo de excepciones */
void init_excepcion(void)
{
	pISR_DABORT = (unsigned) excepcion_handler;
	pISR_UNDEF = (unsigned) excepcion_handler;
	pISR_SWI = (unsigned) excepcion_handler;
}
```

==========

jugada.c

```
typedef enum estado_automata {iniciar, reiniciar, mover, parpadear, ganar, terminar}
estado_automata;

#define max_numero R_DIM-1
#define periodo_parpadeo 250 // milisegundos
#define periodo_estado 2000 // milisegundos
```

```
#define periodo_ganador 5000 // milisegundos
#define FICHA_NEGRA 2
#define CASILLA_VACIA 0
#define ESCRIBIR 1
#define BORRAR 0

/*--- variables globales ---*/
static boton boton_antes;
static char f, c, casilla;
static uint32_t siguiente_parpadeo, siguiente_estado, antes; // milisegundos
static uint32_t tiempo_partida, tiempo_partida_antes; // segundos
static boton boton_ahora;
static estado_automata estado_auto;
static pantalla panta_ahora;
static char ganador;

/* Inicializa el módulo y muestra por pantalla la ayuda */
void jugada_init()
{
	pantalla_init(); // Inicializa el módulo de pantalla

	boton_antes = b_ninguno;
	estado_auto = iniciar;
	siguiente_parpadeo = 0;
	siguiente_estado = 0;
	antes = 0;
	pantalla_help();
}

/* Establece el ganador */
void jugada_ganador(char g) {
	estado_auto = ganar;
	ganador = g;
}

/* Autómata principal de la jugada, gestiona el estado de la partida a través de los
botones y de la pantalla, además va escribiendo por la pantalla el estado de la partida.
Devuelve 1 si se ha terminado la jugada, además de la fila y la columna introducidas, y 0
en otro caso. */
char jugada_gestionar(uint32_t ahora, uint32_t tiempo_patron, char *fila, char *columna,
char tablero[][R_DIM]) // milisegundos
{
	pantalla_gestionar(ahora);
	botones_antirebotes_gestionar(ahora);

	boton_ahora = botones_antirebotes_pulsado();
	panta_ahora = pantalla_pulsada();

	/* Gestión del tiempo transcurrido en la partida */
	tiempo_partida += (ahora-antes);
	antes = ahora;
	if (estado_auto != iniciar && estado_auto != ganar && estado_auto != terminar
				&& tiempo_partida >= (tiempo_partida_antes + 1000)) {
		tiempo_partida_antes = tiempo_partida;
```

```
			pantalla_tiempo(tiempo_partida / 1000);
		}

		switch (estado_auto) {
		case iniciar:
			/* Espera hasta que se inicie la partida y, entonces, inicializa todos los
		elementos de la partida y muestra el tablero por pantalla */
			if (boton_ahora != boton_antes || panta_ahora == p_pulsada) {
				boton_antes = boton_ahora;
				estado_auto = mover;
				tiempo_partida = 0;
				tiempo_partida_antes = 0;
				antes = ahora;
				f = 0;
				c = 0;
				casilla = tablero[f][c];
				pantalla_mostrar_partida(tiempo_patron, tablero);
			}

			return 0;

		case reiniciar:
			/* Inicializa todos los elementos para la siguiente jugada y muestra el
		tablero actualizado */
			estado_auto = mover;
			f = 0;
			c = 0;
			casilla = tablero[f][c];
			pantalla_mostrar_partida(tiempo_patron, tablero);

			return 0;

		case mover:
			/* Espera a que se introduzca una fila y una columna por medio de los
botones */
			if (boton_ahora != boton_antes) {
				boton_antes = boton_ahora;
				if (boton_ahora == b_izquierdo) {
					/* Cuando se pulsa el botón izquierdo, se incrementa la
				columna y se actualiza la ficha en pantalla */
					pantalla_casilla(f, c, CASILLA_VACIA, p_negro);
					pantalla_casilla(f, c, casilla, p_negro);
					if (c == max_numero) c = 0;
					else c++;
					casilla = tablero[f][c];
				}
				else if (boton_ahora == b_derecho) {
					/* Cuando se pulsa el botón derecho, se incrementa la fila y
				se actualiza la ficha en pantalla */
					pantalla_casilla(f, c, CASILLA_VACIA, p_negro);
					pantalla_casilla(f, c, casilla, p_negro);
					if (f == max_numero) f = 0;
					else f++;
					casilla = tablero[f][c];
```

```
                }
            }
            pantalla_casilla(f, c, FICHA_NEGRA, p_gris);
            pantalla_pulsar_continuar(ESCRIBIR);

            /* Cuando la pantalla es pulsada, comienza el periodo para cancelar la
        jugada o pasar a la siguiente */
            if (panta_ahora == p_pulsada) {
                estado_auto = parpadear;
                pantalla_pulsar_continuar(BORRAR);
                pantalla_pulsar_cancelar(ESCRIBIR);
                siguiente_parpadeo = ahora;
                siguiente_estado = ahora + periodo_estado;
            }
            return 0;

        case parpadear:
            /* Espera a que se cancele la jugada, o se valide */
            if (ahora >= siguiente_parpadeo) { // Parpadea la ficha mientras se espera
                siguiente_parpadeo += periodo_parpadeo;
                if (casilla == CASILLA_VACIA) {
                    casilla = FICHA_NEGRA;
                }
                else {
                    casilla = CASILLA_VACIA;
                }
                pantalla_casilla(f, c, casilla, p_gris);
            }

            if (boton_ahora != boton_antes || panta_ahora == p_pulsada) { // Se cancela
la jugada
                boton_antes = boton_ahora;
                estado_auto = mover;
                pantalla_pulsar_cancelar(BORRAR);
                casilla = tablero[f][c];
                pantalla_casilla(f, c, CASILLA_VACIA, p_negro);
                pantalla_casilla(f, c, casilla, p_negro); // Restablecer la ficha que
había
                return 0;
            }
            else if (ahora >= siguiente_estado) { // Se valida la jugada
                *fila = f;
                *columna = c;
                estado_auto = reiniciar;
                return 1;
            }
            else
                return 0;

        case ganar:
            /* Termina la partida y se muestra el ganador */
            pantalla_ganador(ganador);
            estado_auto = terminar;
            siguiente_estado = ahora + periodo_ganador;
```

```
		return 0;

	case terminar:
		/* Tras 5 segundos, se reinicia el juego volviendo a mostrar la pantalla de
ayuda */
		if (ahora >= siguiente_estado)
			jugada_init(); // reinicio
		return 0;
	}
}
```

==========

latido.c

```
#define periodo 250 // milisegundos

/*--- variables globales ---*/
static uint32_t siguiente; // milisegundos
static unsigned char estado_led;

/* Inicializa el módulo */
void latido_init()
{
	siguiente = 0;
	estado_led = 0;
}

/* Gestiona el estado del latido en los leds */
void latido_gestionar(uint32_t ahora) // milisegundos
{
	if (ahora >= siguiente) {
		siguiente += periodo;
		if (estado_led == 0) {
			estado_led = 1;
			led1_on();
		}
		else {
			estado_led = 0;
			led1_off();
		}
	}
}
```

==========

pantalla.c

```
#define minX_tablero 19
#define maxX_tablero 179
#define minY_tablero 39
#define maxY_tablero 199
#define xPan_inicial 41 // Primera fila tablero
#define yPan_inicial 25 // Primera columna tablero
#define pan_interval 20 // Intervalo entre cada fila/columna
```

```
#define retardo 200 // milisegundos
#define ESCRIBIR 1

// Estados de las casillas del tablero
enum {
CASILLA_VACIA = 0,
FICHA_BLANCA = 1,
FICHA_NEGRA = 2
};

typedef enum estado_automata {pantalla_esperar, pantalla_pulsar} estado_automata;

/*--- variables globales ---*/
static uint32_t siguiente; // milisegundos
static uint32_t tiempo_patron_antes = 0, tiempo_partida_antes = 0; // segundos
static volatile estado_automata estado_auto;
static pantalla estado_pant;
static int pulsacionAntes, pulsacionAhora;

/* Inicializa el módulo */
void pantalla_init() {
	/* clear screen */
	Lcd_Clr();
	Lcd_Active_Clr();

	pulsacionAntes = TS_puls();
	pulsacionAhora = pulsacionAntes;
	estado_auto = pantalla_esperar;
	estado_pant = p_no_pulsada;

	Lcd_Dma_Trans(); // dma transport virtual LCD screen to LCD actual screen
}

/* Muestra la pantalla de ayuda */
void pantalla_help() {
	/* clear screen */
	Lcd_Clr();
	Lcd_Active_Clr();

	Lcd_DspAscII8x16(10,10,BLACK,"      Reversi8 -- Instrucciones        ");
	Lcd_DspAscII8x16(10,30,DARKGRAY,"El usuario maneja las fichas negras y  "); //
longitud max 39
	Lcd_DspAscII8x16(10,40,DARKGRAY,"empieza colocando.                     ");
	Lcd_DspAscII8x16(10,50,DARKGRAY,"La maquina maneja las fichas blancas.  ");
	Lcd_DspAscII8x16(10,70,DARKGRAY,"El movimiento consiste en colocar una  ");
	Lcd_DspAscII8x16(10,80,DARKGRAY,"ficha de forma que flanquee una o      ");
	Lcd_DspAscII8x16(10,90,DARKGRAY,"varias fichas del color contrario y    ");
	Lcd_DspAscII8x16(10,100,DARKGRAY,"voltear esas fichas para que pasen    ");
	Lcd_DspAscII8x16(10,110,DARKGRAY,"a mostrar el propio color.            ");
	Lcd_DspAscII8x16(10,120,DARKGRAY,"Se voltean todas las fichas que se    ");
	Lcd_DspAscII8x16(10,130,DARKGRAY,"han flanqueado en ese turno al        ");
	Lcd_DspAscII8x16(10,140,DARKGRAY,"colocar la ficha del color contrario, ");
	Lcd_DspAscII8x16(10,150,DARKGRAY,"de tal manera que se forme una linea  ");
	Lcd_DspAscII8x16(10,160,DARKGRAY,"recta de fichas del mismo color entre ");
```

```
	Lcd_DspAscII8x16(10,170,DARKGRAY,"dos fichas del color contrario.       ");
	Lcd_DspAscII8x16(10,190,DARKGRAY,"Gana el jugador con mas fichas de su  ");
	Lcd_DspAscII8x16(10,200,DARKGRAY,"color en el tablero.                  ");

	pantalla_pulsar_continuar(ESCRIBIR);
}

/* Transforma x del tablero en x de pantalla */
int pantalla_xTab_xPan(char xTab) {
	return xTab * pan_interval + xPan_inicial;
}

/* Transforma y del tablero en y de pantalla */
int pantalla_yTab_yPan(char yTab) {
	return yTab * pan_interval + yPan_inicial;
}

/* Actualiza el estado de la casilla en el tablero con la fila y columna dadas */
void pantalla_casilla(char fila, char columna, char casilla, pantalla_color color) {
	int x = pantalla_xTab_xPan(fila);
	int y = pantalla_yTab_yPan(columna);
	if (casilla == FICHA_NEGRA) { // Actualiza la ficha del humano
		if (color == p_negro)
			Lcd_DspAscII8x16(y,x,BLACK,"N");
		else
			Lcd_DspAscII8x16(y,x,DARKGRAY,"N");
	}
	else if (casilla == FICHA_BLANCA) // Actualiza la ficha de la máquina
		Lcd_DspAscII8x16(y,x,BLACK,"B");
	else
		Lcd_DspAscII8x16(y,x,WHITE,"N"); // Borra la ficha del humano

	Lcd_Dma_Trans(); // dma transport virtual LCD screen to LCD actual screen
}

/* Devuelve la longitud de valor */
uint32_t longitud(uint32_t valor) {
	uint32_t l = 1;
	while (valor > 9) {
		l++;
		valor /= 10;
	}
	return l;
}

/* Transforma entero a cadena
 * longitud = longitud de entrada
 */
void atoi(uint32_t entrada, uint32_t longitud, char *salida) {
	while (longitud > 0) {
		salida[longitud-1] = (INT8U) (entrada % 10 + '0');
		entrada /= 10;
		longitud--;
	}
```

```
}

/* Muestra el tiempo cada segundo */
void pantalla_tiempo(uint32_t tiempo_partida) { // segundos
	// Borrar tiempo antes
	uint32_t lon = longitud(tiempo_partida_antes);
	volatile char t_antes[56] = "";
	atoi(tiempo_partida_antes, lon, &t_antes);

	Lcd_DspAscII8x16(199, 49, WHITE, &t_antes);
	Lcd_Dma_Trans(); // dma transport virtual LCD screen to LCD actual screen

	lon = longitud(tiempo_partida);
	volatile char t_ahora[56] = "";
	atoi(tiempo_partida, lon, &t_ahora);

	// Mostrar tiempo de partida
	tiempo_partida_antes = tiempo_partida;
	Lcd_DspAscII8x16(199, 39, BLACK, "Tiempo total:");
	Lcd_DspAscII8x16(199, 49, BLACK, &t_ahora);
	Lcd_Dma_Trans(); // dma transport virtual LCD screen to LCD actual screen
}

/* Muestra el tablero con sus fichas, y el tiempo del último patrón de volteo */
void pantalla_mostrar_partida(uint32_t tiempo_patron, char tablero[][tam_tablero]) {
	/* clear screen */
	Lcd_Clr();
	Lcd_Active_Clr();

	Lcd_DspAscII8x16(9,19,BLACK,"C");
	Lcd_DspAscII8x16(0,29,BLACK,"F");

	// Numeración filas
	Lcd_DspAscII8x16(5,41,BLACK,"0");
	Lcd_DspAscII8x16(5,61,BLACK,"1");
	Lcd_DspAscII8x16(5,81,BLACK,"2");
	Lcd_DspAscII8x16(5,101,BLACK,"3");
	Lcd_DspAscII8x16(5,121,BLACK,"4");
	Lcd_DspAscII8x16(5,141,BLACK,"5");
	Lcd_DspAscII8x16(5,161,BLACK,"6");
	Lcd_DspAscII8x16(5,181,BLACK,"7");

	// Numeración columnas
	Lcd_DspAscII8x16(25,21,BLACK,"0");
	Lcd_DspAscII8x16(45,21,BLACK,"1");
	Lcd_DspAscII8x16(65,21,BLACK,"2");
	Lcd_DspAscII8x16(85,21,BLACK,"3");
	Lcd_DspAscII8x16(105,21,BLACK,"4");
	Lcd_DspAscII8x16(125,21,BLACK,"5");
	Lcd_DspAscII8x16(145,21,BLACK,"6");
	Lcd_DspAscII8x16(165,21,BLACK,"7");

	// Dibuja el tablero
	int x, y;
```

```
	for (y = minY_tablero; y < maxY_tablero; y += 20) {
		for (x = minX_tablero; x < maxX_tablero; x += 20) {
			Lcd_Draw_Box(x, y+20, x+20, y, BLACK);
		}
	}

	// Muestra las fichas
	int i, j;
	for (i = 0; i < tam_tablero; i++) {
		for (j = 0; j < tam_tablero; j++) {
			if (tablero[i][j] != CASILLA_VACIA) {
				x = pantalla_xTab_xPan(i);
				y = pantalla_yTab_yPan(j);
				if (tablero[i][j] == FICHA_NEGRA)
					Lcd_DspAscII8x16(y,x,BLACK,"N");
				else
					Lcd_DspAscII8x16(y,x,BLACK,"B");
			}
		}
	}

	// Borrar tiempo patrón antes
	uint32_t lon = longitud(tiempo_patron_antes);
	volatile char t_antes[56] = "";
	atoi(tiempo_patron_antes, lon, &t_antes);

	Lcd_DspAscII8x16(199, 79, WHITE, &t_antes);
	Lcd_Dma_Trans(); // dma transport virtual LCD screen to LCD actual screen

	// Mostrar tiempo de patrón
	lon = longitud(tiempo_patron);
	volatile char t_ahora[56] = "";
	atoi(tiempo_patron, lon, &t_ahora);
	tiempo_patron_antes = tiempo_patron;

	Lcd_DspAscII8x16(199, 69, BLACK, "Tiempo patron:");
	Lcd_DspAscII8x16(199, 79, BLACK, &t_ahora);

	Lcd_Dma_Trans(); // dma transport virtual LCD screen to LCD actual screen
}

/* Gestiona el estado del touchpad */
void pantalla_gestionar(uint32_t ahora) { // milisegundos
	switch (estado_auto) {
		case pantalla_esperar:
			pulsacionAhora = TS_puls();
			if (pulsacionAhora != pulsacionAntes) {
				pulsacionAntes = pulsacionAhora;
				siguiente = ahora + retardo;
				estado_auto = pantalla_pulsar;
				estado_pant = p_pulsada;
			}
			break;
```

```
		case pantalla_pulsar:
			if (ahora >= siguiente) {
				// Reset pulsaciones
				pulsacionAhora = TS_puls();
				pulsacionAntes = pulsacionAhora;
				estado_auto = pantalla_esperar;
			}
			break;
	}
}

/* Devuelve el estado del touchpad */
pantalla pantalla_pulsada()
{
	pantalla antes = estado_pant;
	estado_pant = p_no_pulsada;
	return antes;
}

/* Muestra o borra el mensaje de “Pulse para jugar” */
void pantalla_pulsar_continuar(char escribir) {
	if (escribir == ESCRIBIR)
		Lcd_DspAscII8x16(10,220,BLACK,"   Pulse para jugar                    "); //
fila max
	else
		Lcd_DspAscII8x16(10,220,WHITE,"   Pulse para jugar                    "); //
fila max

	Lcd_Dma_Trans(); // dma transport virtual LCD screen to LCD actual screen
}

/* Muestra o borra el mensaje de “Pulse para cancelar” */
void pantalla_pulsar_cancelar(char escribir) {
	if (escribir == ESCRIBIR)
		Lcd_DspAscII8x16(10,220,BLACK,"   Pulse para cancelar                 "); //
fila max
	else
		Lcd_DspAscII8x16(10,220,WHITE,"   Pulse para cancelar                 "); //
fila max

	Lcd_Dma_Trans(); // dma transport virtual LCD screen to LCD actual screen
}

/* Muestra la pantalla de final de partida */
void pantalla_ganador(char ganador) {
	/* clear screen */
	Lcd_Clr();
	Lcd_Active_Clr();

	Lcd_DspAscII8x16(10,10,BLACK,"      Reversi8 -- Partida terminada    ");
	if (ganador == FICHA_BLANCA)
		Lcd_DspAscII8x16(10,40,BLACK,"Has perdido, lastima...               ");
	else
		Lcd_DspAscII8x16(10,40,BLACK,"Has ganado, enhorabuena               ");
```

```
	Lcd_DspAscII8x16(10,50,DARKGRAY,"Se va a reiniciar la partida           ");

	Lcd_Dma_Trans(); // dma transport virtual LCD screen to LCD actual screen
}
```

==========

pila.c

```
static uint32_t inicio = _ISR_STARTADDRESS-0xf00+256*7;    // Dirección de inicio declarada en 44binit.asm para la pila // 0xc7fee00
static uint32_t fin = _ISR_STARTADDRESS-0xf00+256*5;       // Dirección de fin de la pila, se le resta tres para que esté alineado con la dirección de inicio
static uint32_t mask_24bits = 0x00ffffff;

static uint32_t* pos;     // Puntero utilizado internamente para guardar datos en la pila

/* Inicializa el módulo */
void pila_init()
{
	pos = inicio;
}

/* Guarda en la pila los dos enteros de la siguiente manera:
 * ID_evento = bits 0-7
 * auxData = bits 0-23 */
void push_debug(uint8_t ID_evento, uint32_t auxData)
{
	uint32_t tiempo = timer2_leer(); // tiempo en microsegundos cuando ocurre

	if (pos == fin) {
		pos = inicio;
	}

	uint32_t valorA = ID_evento << 24;
	uint32_t valorB = auxData & mask_24bits;
	uint32_t valor = valorA | valorB;

	*pos = tiempo;
	pos--;              // Aumenta 32bits
	*pos = valor;
	pos--;
}
```

==========

timer0.c

```
/* declaración de función que es rutina de servicio de interrupción
 * https://gcc.gnu.org/onlinedocs/gcc/ARM-Function-Attributes.html */
void timer0_ISR(void) __attribute__((interrupt("IRQ")));

#define preescalado_clear 0xFFFFFF00
```

```
#define preescalado 0x00000009 // bits 7-0 (=0) preescalado de timer0 (cuanto menor, mayor frecuencia)
#define entrada_mux_clear 0xFFFFFFF0
#define entrada_mux 0x00000000 // bits 3-0 (=0) entrada_mux de timer0 (la 00 es el divisor 2)
#define max_int 64000 // máximo valor para el contador
#define comp_int 0 // valor de comparación
#define bit_timer0_intmod_irq 0xFFFFDFFF //bit 13 (=0) perteneciente a timer0 en rINTMOD que corresponde a IRQ
#define bit_timer0_autoreload 0x0008 // auto-reload on
#define bit_timer0_manual 0x0002 // manual update on
#define bit_timer0_start 0x0001 // start
#define bit_timer0_clean 0xFFFFFFF0 // bits 3-0 (=0) pertenecientes a timer0 en rTCON

static volatile int timer0_num_int = 0;

/* Función de tratamiento de interrupción */
void timer0_ISR(void)
{
	timer0_num_int = timer0_num_int+1;

	/* borrar bit en I_ISPC para desactivar la solicitud de interrupción */
	rI_ISPC |= BIT_TIMER0; // BIT_TIMER0 está definido en 44b.h y pone un uno en el bit 13 que corresponde al Timer0
}

/* Inicializa el módulo */
void timer0_inicializar()
{
	/* Configuración controlador de interrupciones aplicando máscaras */
	//rINTMOD = 0x0; // Configura las líneas como de tipo IRQ
	rINTMOD &= bit_timer0_intmod_irq; // Configura timer0 como de tipo IRQ
	rINTCON = 0x1; // Habilita int. vectorizadas y la línea IRQ (FIQ no)
	rINTMSK &= ~(BIT_TIMER0); // habilitamos en vector de máscaras de interrupción el Timer0

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

	/* Configura el Timer0 aplicando máscaras */
	rTCFG0 &= preescalado_clear;
	rTCFG0 |= preescalado; // ajusta el preescalado, cuanto más pequeño más rápido
	rTCFG1 &= entrada_mux_clear;
	rTCFG1 |= entrada_mux; // selecciona la entrada del mux que proporciona el reloj. La 00 corresponde a un divisor de 1/2.
	// No hacen falta máscaras, exclusivos del timer0
	rTCNTB0 = max_int; // valor inicial de cuenta (la cuenta es descendente)
	rTCMPB0 = comp_int; // valor de comparación
}


/* Empieza a contar */
void timer0_empezar()
{
```

```
	rTCON &= bit_timer0_clean; // Se reestablece el control de timer0, por si se llama a esta función con el bit start a on del timer0

	rTCON |= bit_timer0_autoreload; // autoreload on
	rTCON |= bit_timer0_manual; // manual update on
	timer0_num_int = 0;
	rTCON ^= bit_timer0_manual; // manual update off
	rTCON |= bit_timer0_start; // start
}

/* Para el contador y devuelve el tiempo en milisegundos */
uint32_t timer0_parar()
{
	rTCON &= bit_timer0_clean; // bits timer0 a 0
	return timer0_leer();
}

/* Devuelve el tiempo en milisegundos */
uint32_t timer0_leer()
{
	uint32_t ticks1 = (max_int * timer0_num_int) + (max_int - rTCNTO0);
	uint32_t ticks2 = (max_int * timer0_num_int) + (max_int - rTCNTO0);

	// Al consultar primero la variable timer0_num_int y después el contador, puede haber una medición que sea inferior a la real (subestimación), pero nunca superior (sobreestimación)
	// La frecuencia real es 32, ya que al usar el divisor 2 (entrada_mux) se divide entre 2 la frecuencia
	if (ticks2 - ticks1 >= max_int) return ticks2 / 3200; // Devuelve en milisegundos (Frecuencia real 3.2MHz)
	else return ticks1 / 3200;
}

/* Devuelve el número de interrupciones */
uint32_t interr()
{
	return timer0_num_int;
}
```

==========

simular.h

```
/* Para realizar las simulaciones, se han redefinido todos los elementos de más bajo nivel */
#ifndef _SIMULAR_H_
#define _SIMULAR_H_

#define simulacion // comentar si no es una simulación
#ifdef simulacion

#include "stdint.h"
```

```
/*--- declaración de funciones visibles del módulo simular.c/simular.h ---*/

// common
#define _ISR_STARTADDRESS 0xc7fff00

#ifndef ULONG
#define ULONG unsigned long
#endif

#define UINT unsigned int
#define USHORT unsigned short
#define UCHAR unsigned char
#define U32  unsigned int
#define INT32U  unsigned int
#define INT32int
#define U16  unsigned short
#define INT16U  unsigned short
#define INT16short int
#define S32  int
#define S16  short int
#define U8   unsigned char
#define INT8U       unsigned char
#define byte unsigned char
#define INT8 char
#define      S8     char

#ifndef FALSE
#define FALSE       0
#endif
#ifndef TRUE
#define TRUE        1
#endif

#define OK          1
#define FAIL 0
#define FileEnd     1
#define      NotEnd 0

void sys_init();

// 8led
void D8Led_init();
void D8Led_symbol(int value);

// button
typedef enum estado_button {button_none, button_iz, button_dr} estado_button;
typedef void(function_ptr)(estado_button boton);

void button_iniciar(void);
estado_button button_estado();
void button_empezar(function_ptr *callback);
void button_habilitar_interr();

// excepcion
```

```
void init_excepcion();

// led
void leds_off();                                    // todos los leds apagados (*)
void led1_on();                                     // led 1 encendido (*)
void led1_off();                                    // led 1 apagado

// lcd
/* tamano pantalla */
#define TLCD_160_240 (0)
#define VLCD_240_160 (1)
#define CLCD_240_320 (2)
#define MLCD_320_240 (3)
#define ELCD_640_480 (4)
#define SLCD_160_160 (5)
#define LCD_TYPE MLCD_320_240

#if (LCD_TYPE == TLCD_160_240)
#define SCR_XSIZE (160)
#define SCR_YSIZE (240)
#define LCD_XSIZE (160)
#define LCD_YSIZE (240)
#elif (LCD_TYPE == VLCD_240_160)
#define SCR_XSIZE (240)
#define SCR_YSIZE (160)
#define LCD_XSIZE (240)
#define LCD_YSIZE (160)
#elif (LCD_TYPE == CLCD_240_320)
#define SCR_XSIZE (240)
#define SCR_YSIZE (320)
#define LCD_XSIZE (240)
#define LCD_YSIZE (320)
#elif (LCD_TYPE == MLCD_320_240)
#define SCR_XSIZE (320)
#define SCR_YSIZE (240)
#define LCD_XSIZE (320)
#define LCD_YSIZE (240)
#elif (LCD_TYPE == ELCD_640_480)
#define SCR_XSIZE (640)
#define SCR_YSIZE (480)
#define LCD_XSIZE (640)
#define LCD_YSIZE (480)
#elif (LCD_TYPE == SLCD_160_160)
#define SCR_XSIZE (160)
#define SCR_YSIZE (160)
#define LCD_XSIZE (160)
#define LCD_YSIZE (160)
#endif

/* screen color */
#define MODE_MONO (1)
#define MODE_GREY4 (4)
#define MODE_GREY16 (16)
#define MODE_COLOR (256)
```

```
#define Ascii_W 8
#define XWIDTH 6
#define BLACK 0xf
#define WHITE 0x0
#define LIGHTGRAY 0x5
#define DARKGRAY 0xa
#define TRANSPARENCY 0xff

#define HOZVAL (LCD_XSIZE / 4 - 1)
#define HOZVAL_COLOR (LCD_XSIZE * 3 / 8 - 1)
#define LINEVAL (LCD_YSIZE - 1)
#define MVAL (13)
#define M5D(n) ((n)&0x1fffff)
#define MVAL_USED 0

/* tamano array */
#define ARRAY_SIZE_MONO (SCR_XSIZE / 8 * SCR_YSIZE)
#define ARRAY_SIZE_GREY4 (SCR_XSIZE / 4 * SCR_YSIZE)
#define ARRAY_SIZE_GREY16 (SCR_XSIZE / 2 * SCR_YSIZE)
#define ARRAY_SIZE_COLOR (SCR_XSIZE / 1 * SCR_YSIZE)

/* clkval */
#define CLKVAL_MONO (12)
#define CLKVAL_GREY4 (12)
#define CLKVAL_GREY16 (12)
#define CLKVAL_COLOR (10)

#define LCD_BUF_SIZE (SCR_XSIZE * SCR_YSIZE / 2)
#define LCD_ACTIVE_BUFFER (0xc300000)
#define LCD_VIRTUAL_BUFFER (0xc300000 + LCD_BUF_SIZE)

#define LCD_PutPixel(x, y, c)                                                \
  (*(INT32U *)(LCD_VIRTUAL_BUFFER + (y)*SCR_XSIZE / 2 + ((x)) / 8 * 4)) =      \
      (*(INT32U *)(LCD_VIRTUAL_BUFFER + (y)*SCR_XSIZE / 2 + ((x)) / 8 * 4)) &  \
          (~(0xf0000000 >> ((((x)) % 8) * 4))) |                               \
      ((c) << (7 - ((x)) % 8) * 4)
#define LCD_Active_PutPixel(x, y, c)                                         \
  (*(INT32U *)(LCD_ACTIVE_BUFFER + (y)*SCR_XSIZE / 2 + (319 - (x)) / 8 * 4)) = \
      (*(INT32U *)(LCD_ACTIVE_BUFFER + (y)*SCR_XSIZE / 2 +                   \
                   (319 - (x)) / 8 * 4)) &                                   \
          (~(0xf0000000 >> (((319 - (x)) % 8) * 4))) |                       \
      ((c) << (7 - (319 - (x)) % 8) * 4)

#define GUISWAP(a, b)                                                        \
  {                                                                          \
    a ^= b;                                                                  \
    b ^= a;                                                                  \
    a ^= b;                                                                  \
  }

//Funciones
void Lcd_Init(void);
void Lcd_Active_Clr(void);
```

```
void Lcd_Clr(void);
void Lcd_DspAscII8x16(INT16U x0, INT16U y0, INT8U ForeColor, INT8U *s);
void Lcd_Draw_Box(INT16 usLeft, INT16 usTop, INT16 usRight, INT16 usBottom,
                  INT8U ucColor);
void Lcd_Dma_Trans(void);

// TP
void TS_init(void);
int TS_puls();

// timer0
void timer0_inicializar();
void timer0_empezar();
uint32_t timer0_parar();
uint32_t timer0_leer();

// timer2
void timer2_inicializar();
void timer2_empezar();
uint32_t timer2_parar();
uint32_t timer2_leer();


#endif

#endif /* _SIMULAR_H_ */
```