

PROYECTO HARDWARE

PROYECTO FINAL

Diego Santolaya Martínez (727209)
Abel Chils Trabanco (718997)

ÍNDICE:

1. Introducción.....	pg 1
2. Estructura del proyecto.....	pg 1
3. Descripción de bibliotecas.....	pg 2
4. Descripción de componentes.....	pg 3
4.1 Biblioteca dibujo de pantalla....	pg 3
4.2 Timer 2.....	pg 3
4.3 Tratamiento de excepciones.....	pg 4
4.4 Cargar programa en la RAM.....	pg 4
4.5 TouchPad y calibración.....	pg 4
4.6 Gestión de entrada salida.....	pg 5
4.7 Análisis linker-script (Práctica 2)....	pg 6
5. Diseño de la pantalla.....	pg 6
6. Comparativa rendimiento	pg 9
7. Pruebas.....	pg 10
8. Problemas encontrados.....	pg 10
9. Conclusiones.....	pg 11
Anexo: Partes representativas del código...	pg 11

1. Introducción

El proyecto que se presenta a continuación está basado en las prácticas 1, 2 y 3 de la asignatura, durante las cuales se han desarrollado distintas funcionalidades y mejoras para poder jugar al juego reversi8 con la placa proporcionada en el laboratorio.

Accediendo a la documentación de la placa y sus periféricos se configuran los dispositivos de entrada/salida correctamente y se crean librerías para abstraer el hardware del nivel del juego. De esta forma, desde el juego del reversi se pueden utilizar fácilmente las funciones que interactúan con los periféricos, simplificando enormemente el nivel de complejidad de la máquina de estados.

Al juego final se puede jugar desde la pantalla táctil en la que se visualizará el tablero o con los dos botones y el marcador led da la parte izquierda, permitiendo más de una forma de mover las fichas. También se hace un tratamiento de excepciones y se tienen en cuenta casos extraños que puedan desembocar en confusiones (tocar a la vez los botones y la pantalla, etc.).

2. Estructura del proyecto



El proyecto, está compuesto por 4 niveles, los cuales están delimitados según su dependencia del hardware.

En el primer nivel se encuentran las bibliotecas que interactúan directamente con el hardware, que son tp, lcd, button_rebotes, 8led, led, exception_handler, 44binit.asm, timer2, timer1 y pila.

Luego existe un segundo nivel el cual utiliza solamente las bibliotecas del nivel 1, estando compuesto este nivel por los gestores del lcd (lcdManager) y el touchpad (tpManager). La existencia de este nivel permite crear una separación entre el juego y el hardware, permitiendo esto que las implementaciones de bibliotecas que trabajan con el hardware puedan ser reutilizadas en otros proyectos así como poder desarrollar el juego sin haber desarrollado previamente el uso del hardware.

El tercer nivel es el juego del reversi, el cual utiliza tanto las bibliotecas del nivel 2 (las cuales se desarrollaron en la 3ª práctica) como las bibliotecas del nivel 1 que no están gestionadas por bibliotecas del nivel 2 (las cuales se desarrollaron en la 2ª práctica). El último nivel está compuesto por el fichero main. Este se encarga tanto de inicializar todas las bibliotecas, así como inicializar el juego.

3. Descripción de las distintas bibliotecas

A continuación se describen brevemente las distintas bibliotecas que se usarán para el funcionamiento del proyecto:

- **tp**: Esta biblioteca interactúa con el hardware para poder controlar el touchpad desde ella. Posee una interfaz que será utilizada por todas las demás bibliotecas que deseen acceder al touchpad y desde la que se puede saber si se ha pulsado el touchpad y el punto en el que se ha pulsado según el sistema de coordenadas del lcd((0,0) esquina superior izquierda y (320,240) esquina inferior derecha). Por otro lado esta biblioteca también calibra el touchpad las 8 primeras veces que se pulsa sobre él mediante unos pasos guiados.
- **lcd**: Esta biblioteca interactúa con el hardware para poder controlar el lcd desde ella. Posee una interfaz que será utilizada por todas las demás bibliotecas que deseen acceder al lcd y desde la que se puede dibujar formas básicas (cuadrados, líneas, círculos y letras) sobre él.
- **button rebotes**: Esta biblioteca interactúa con el hardware para poder controlar los botones desde ella. Posee una interfaz que será utilizada por todas las demás bibliotecas que deseen interactuar con los botones y desde la que se puede saber si se ha pulsado un botón, y en caso afirmativo cuál. Por otro lado, esta biblioteca se encarga de gestionar los rebotes producidos durante la pulsación de los botones haciendo uso para ello del timer 0. Además se ha añadido la mejora de autoincremento con la pulsación mantenida.
- **8led**: Esta biblioteca interactúa con el hardware para poder controlar el 8led.
- **led**: Esta biblioteca interactúa con el hardware para poder controlar los 2 leds.
- **exception handler**: Esta biblioteca se encarga de gestionar las excepciones de tipo data abort, prefetch abort, software interrupt y undefined.
- **44binit.asm**: Este fichero se encarga de inicializar la placa así como de copiar el programa de la memoria flash a la memoria ram y ejecutarlo.
- **timer1**: Esta biblioteca se encarga de gestionar el timer 1. Posee una interfaz que será utilizada por todas las demás bibliotecas que deseen acceder al timer 1, desde la que se puede resetear la cuenta y ver el valor de la cuenta en un momento dado. Este timer tiene una baja precisión y es utilizado para controlar el paso de los segundos en el juego(tiempo de juego).

- **timer2**: Esta biblioteca se encarga de gestionar el timer 2. Posee una interfaz que será utilizada por todas las demás bibliotecas que deseen acceder al timer 2, desde la que se puede resetear la cuenta y ver el valor de la cuenta en un momento dado. Su precisión es muy alta y es utilizada por el juego para obtener el tiempo de calculo.
- **pila**: Esta biblioteca se encarga de gestionar la pila de depuración. Posee una interfaz que permite tanto apilar como desapilar eventos de la pila.
- **lcdManager**: Esta biblioteca se encarga de dibujar las pantallas. Para ello hace uso de la biblioteca lcd.
- **tpManager**: Esta biblioteca se encarga de proporcionar una interfaz desde la que se pueda saber en que elemento de la pantalla se ha pulsado (por ejemplo en el tablero, en el botón de pasar, fuera de la pantalla). Su implementación depende del diseño de las pantallas.
- **reversi**: Esta biblioteca contiene el juego del reversi y la maquina de estados principal del programa. Desde esta, se van utilizando el resto de bibliotecas.
- **main**: Este fichero se encarga de inicializar los diferentes componentes, cambia de modo supervisor a modo usuario y proceder a lanzar el juego.

4. Descripción de los diferentes componentes del proyecto

4.1 Biblioteca de dibujo de pantallas

Debido a que el dibujo y diseño de las pantallas es algo que consume mucho tiempo, para poder desarrollar la biblioteca lcdManager sin acceso a la placa, se desarrollo una biblioteca que sustituye a la biblioteca lcd proporcionando la misma interfaz, que permite crear una imagen con lo que se mostraría en el lcd.

Para hacer uso de esta biblioteca, se ha de estar sin placa y sin el emulador de arm, compilando de forma nativa para el ordenador desde el que se va a ejecutar. La imagen resultante está en formato ppm. Esta puede ser visualizada de forma nativa desde algunas distribuciones linux como Ubuntu, en cambio en otros sistemas operativos como Windows se necesita de programas externos para visualizarla.

4.2 Biblioteca timer 2

La biblioteca timer 2, se encarga de controlar un timer que posea la máxima precisión posible. Para lograrlo, se utilizó una configuración en la cual se estableció un nivel de preescalado

de 0 y un valor de divisor de $\frac{1}{2}$. Esto produce que el contador se decremente con una frecuencia de 32MHz.

Para evitar un gran número de interrupciones, se estableció el valor inicial de la cuenta en 65535(máximo valor permitido), y el valor final de la cuenta en 0. De esta forma llega una interrupción cada 2.05 milisegundos.

Cada vez que llega una interrupción, se aumenta un contador y en el momento que se pide leer el tiempo transcurrido, se devuelve el valor de ese contador multiplicado por 65535 más el valor de la cuenta del timer en ese momento, todo ello dividido entre 32. Esta división se produce debido a que como la frecuencia con la que se decrementa el contador es de 32 Mhz, esto implica que cada $\frac{1}{32}$ de microsegundo el contador es decremnetado. Por ello, el número total de decrementos que ha sufrido el contador, dividido por 32 es el número total de microsegundos que ha pasado.

Para tener la máxima precisión posible, antes de devolver el valor en cuestión se comprueba si el contador ha aumentado. Si es así, se devuelve el aumento del contador.

4.3 Biblioteca de tratamiento de excepciones

Para realizar la identificación y gestión de excepciones, se utilizan 3 funciones. Dos de ellas son utilizadas para realizar la identificación y la tercera es usada para la gestión de estas.

La primera de ellas será invocada una vez que salta un prefetch abort y la segunda lo es con el resto de excepciones. El uso de dos funciones para la identificación de excepciones es debido a que un prefetch abort y un data abort producen el mismo modo de ejecución, por lo que son indistinguibles una vez se han producido. Por ello es necesario que la rutina de interrupción de una de ellas sea diferente. En este caso se decido que el prefetch abort sea diferente ya que en el entorno que se está ejecutando, esta no se va a producir.

En estas funciones de identificación también se identifica la instrucción causante mediante el link register y se llama a otra función que es la que se encarga de gestionar la excepción.

Al gestionar la excepción, lo que se realiza es el guardado de esta en la pila de depuración y se entra en un bucle infinito en el cual se hace que parpadeen los leds y el 8 led, para de esta forma parar la ejecución del programa y avisar al usuario.

4.4 Carga del código en la memoria RAM

Para conseguir que el código se cargue en la RAM, desde el fichero de inicialización de la placa, primero se copia todo el programa a la región que el compilador trabaja como si fuera inicio de código en la RAM (Image_RO_Base).

El compilador piensa que el programa se encuentran ubicado entre Image_RO_Base y Image_ZI_Base, haciendo todas las referencias en base a estas direcciones. Por ello se copia toda la memoria RAM desde la dirección 0x0, (dirección en la que se encuera el programa al ser cargado desde la flash), a la dirección Image_RO_Base. Como el programa tiene un tamaño de Image_ZI_Base - Image_RO_Base, la memoria se irá copiando hasta el momento en el que se vaya a escribir sobre la dirección Image_ZI_Base, que se parará de copia. Más tarde se inicializará a cero la región de memoria ubicada entre Image_ZI_Base y Image_ZI_Limit. Por último se salta a la dirección de RAM donde se ha copiado el código para seguir la ejecución de este pero desde ahí.

4.5 Biblioteca tp y calibración de la pantalla táctil.

Para calibrar la pantalla táctil se exige al usuario nada más empezar el juego que pulse sobre los 4 bordes del tablero. De esta forma se puede obtener el sistema de coordenadas en el

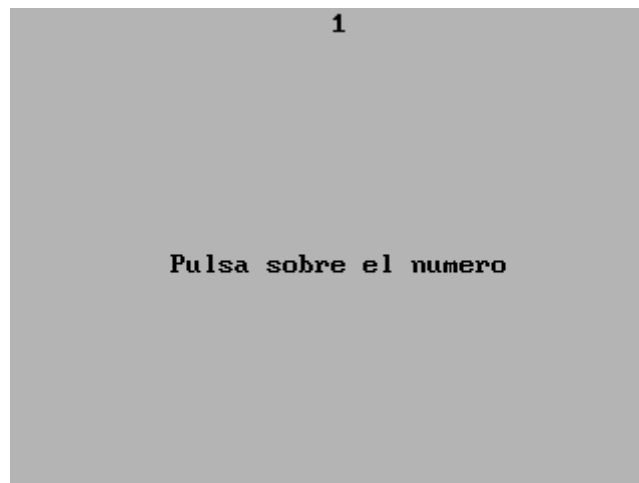
que actúa el touchpad. Esto es almacenado y cada vez que se realice una pulsación, se realiza un cambio de coordenadas del sistema que utiliza el touchpad al sistema que utiliza el lcd.

4.6 Gestión entrada salida

En el proyecto, el control de la entrada salida se produce en 3 módulos diferentes, los cuales son `button_rebotes`, `lcd` y `tp`. Aunque la concurrencia entre el uso del touchpad y el uso de los botones se gestiona en una máquina que se encuentra en el módulo principal del juego.

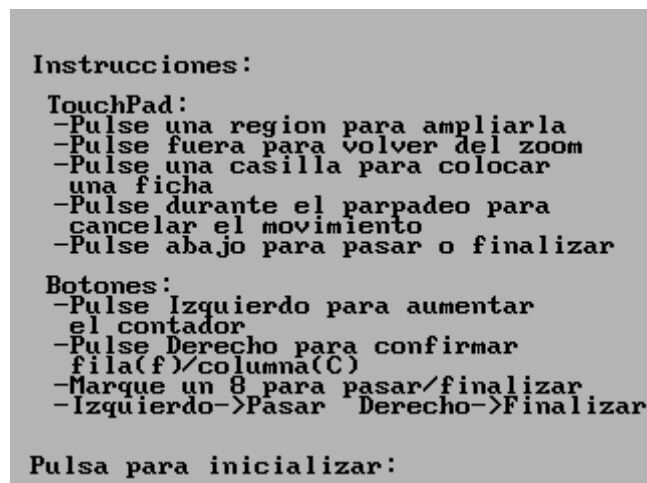
El módulo `button_rebotes` es el módulo que controla la gestión de los botones incluyendo los rebotes de los mismos, así como también es el que sirve de interfaz para saber si se ha pulsado algún botón y cual. Para gestionar los rebotes, se utiliza una máquina de estados la cual tiene 4 estados (esperando, gestionando rebotes iniciales, mantenido y gestionando rebotes finales). Se está en el estado esperando mientras no hay ninguna pulsación. En el momento que existe una pulsación se pasa por un estado transitorio llamado botón pulsado, en el cual se inicializa el timer que controla los rebotes y se deshabilitan las interrupciones de botones y se pasa directamente al estado gestionando rebotes iniciales. En este estado se mantiene durante 200 milisegundos, tras los cuales se vuelven a habilitar las interrupciones de botón y se realiza una encuesta periódica a este para comprobar si se ha soltado. Cada cierto periodo de tiempo que el usuario mantenga el botón pulsado, se registra en la interfaz una nueva pulsación, para de esta forma poder cumplir el requisito de que una pulsación larga equivaliese a varias pulsaciones. Una vez que se detecta que se ha soltado el botón se procede a desactivar de nuevo las interrupciones de botón, y tras 200 milisegundos, en los cuales se producirán los rebotes finales, vuelve al estado de espera. Para realizar la interacción con el lcd se utiliza el módulo `lcd`, aunque para el dibujado de las pantallas se utiliza el módulo `lcdManager` que hace uso de `lcd`. Este, está creado de tal forma que solamente se redibujarán las zonas de pantalla que se actualizan. Con esto se evita en situaciones como mostrar la cuenta del tiempo de juego o mostrar una ficha parpadeando, redibujar la pantalla completa.

La interacción con el touchpad se realiza en el módulo `tp`, el cual se encarga tanto de calibrar como de transformar las coordenadas que se producen en el `tp` a un formato que sea más fácil de integrar con el proyecto (formato de coordenadas que utiliza el `lcd`).



Esta es la primera pantalla que se muestra en el juego, y se utiliza para calibrar el touchpad.

- **Instrucciones:**





Después de calibrar el touchpad y antes de cada partida, se muestra esta pantalla, en la cual se muestran las instrucciones del juego.

- **Tablero completo:**



En esta pantalla se muestra el tablero sin estado de zoom.

- **Tablero Zoom:**

	0	1	2	3	
0					<div>TIEMPO JUEGO</div> <div>segundos</div>
1					
2					<div>TIEMPO CALCULO</div> <div>ns</div>
3					

Pulse o introduzca (8, 8) para pasar

En esta pantalla se muestra el tablero con estado de zoom.

- **Pasar o finalizar:**

PULSA PARA PASAR	PULSA PARA FINALIZAR
PULSA PARA VOLVER	

Esta pantalla se muestra cuando el usuario introduce algún 8 mediante los botones o cuando pulsa sobre la región inferior de la pantalla, en los estados en los que se muestra el tablero.

- **Puntuación:**

Resultado:
 Fichas negras: 12
 Fichas blancas: 17
 Han ganado las blancas
 Pulsa para volver a jugar

Esta es la pantalla que aparece cuando se finaliza una partida

6. Comparativa en el rendimiento entre las diferentes versiones desarrolladas en la práctica 1

Tiempos obtenidos en la ejecución de las diferentes combinaciones sobre la placa de desarrollo.

	O0	O1	O2	O3	Os
C-C	29	17	15	15	17
C-ARM	22	17	16	16	17
C-THUMB	23	18	17	17	19
ARM-C	25	14	13	13	14
ARM-ARM	18	14	13	13	14
ARM-THUMB	19	15	14	14	15

Como se puede comprobar, al ejecutar las diferentes versiones sobre la placa de desarrollo, con nivel de optimización O3, la combinación que mejor rendimiento ofrece es ARM-ARM. Por otro lado se puede ver cómo no existe diferencia en el rendimiento entre las versiones compiladas con nivel de optimización 2 y las compiladas con nivel de optimización 3 (Aunque se comprobó que el código ejecutado era diferente).

Aunque la combinación que ofrece mejor rendimiento es ARM-ARM, para la versión final del proyecto se decidió utilizar la versión C-C, debido a que de esta forma se limitaba menos al compilador a la hora de realizar optimizaciones sobre el proyecto en su conjunto, que con la combinación ARM-ARM si que se hubiese producido.

Tiempos obtenidos en la ejecución de las diferentes combinaciones sobre la simulador.

	O0	O1	O2	O3	Os
C-C	7.1	4.8	2.7	0	4.87
C-ARM	5.5	4.8	4.6	2.6	4.9
C-THUMB	6.1				
ARM-C	6.3				
ARM-ARM	4.7				
ARM-THUMB	5.3				

Se puede comprobar como al ejecutar las diferentes versiones sobre el simulador la diferencia de rendimiento entre las versiones compiladas con nivel de optimización 2 y las compiladas con nivel de optimización 3 si que varía a diferencia de cuando se ejecuta sobre la placa.

Número de instrucciones que se ejecutan según las diferentes combinaciones.

	O0	O1	O2	O3	Os
C-C	68	44	20	2	40
C-ARM	48	48	27	33	46
C-THUMB	53				
ARM-C	55				

ARM-ARM	33				
ARM-THUMB	39				

7.Pruebas

Para comprobar el correcto funcionamiento del sistema, se ha probado individualmente cada función involucrada en el juego, incluyendo las de bajo nivel que interactúan con los periféricos. Esto se consigue inicializando el sistema y, con la ayuda de breakpoints y JTag, se llama aisladamente a la función a probar, a la cual se le somete a distintos tipos de pruebas (comportamiento normal, comportamientos no usuales, etc.) para poder comprobar la consistencia del programa incluso en casos en los que el usuario haga uso incorrecto de los periféricos.

Éste método permite ir verificando las distintas funciones para luego posteriormente al añadirlas al proyecto principal, tener la certeza de que lo que falla no son las funciones que se usan y el error proviene de otras instrucciones sin comprobar.

La depuración con la ayuda de JTag nos proporciona la interfaz adecuada para comprobar los valores de los registros, variables, etc. en cualquier punto de la ejecución de nuestro programa, pudiéndose tomar las decisiones oportunas para solucionar los fallos.

8.Problemas encontrados y soluciones

Durante la práctica se encontraron diversos problemas, la mayoría fruto de la ejecución asíncrona de las rutinas de interrupción, aunque también se tuvieron problemas con el compilador y con las placas.

Cabe destacar un error que producía que al definir un vector de caracteres inicializado dentro de una función y compilar con un nivel de optimización o2, el compilador sustituía este por un vector de 0s. Para solucionar este error del compilador, se definieron los vectores de caracteres o fuera de las funciones de manera global con los atributos *static const* los que fueran constantes, o en el caso de que no lo fuesen se debieron de definir como *volatile*.

Por otro lado, también se detectó otro error en la precisión de los *timers*. Se estuvo realizando pruebas de precisión con el *timer 1*, colocando el preescalado al máximo (256), el mayor divisor posible (32), la cuenta en su máximo valor(65535) y el valor de comparación al mínimo(0). Con esta configuración, deberían de llegar interrupciones cada 8.13 segundos según la documentación de la placa y según la fórmula que permite calcular cada cuanto se decrementa el contador. En cambio, midiendo desde un dispositivo externo el tiempo entre dos interrupciones, este era de aproximadamente dos segundos y medio. Para asegurarnos de que esto no era producido por la configuración de otro periférico las pruebas se realizaron solamente inicializando la placa y el *timer* en cuestión.

A este problema no se le encontró solución, por lo que se debió lidiar con el durante la programación de *timers*, estableciendo las cuentas de los contadores unas 4 veces superior a las que el resultado teórico indicaba que necesitábamos.

9. Conclusiones

Tras la realización del proyecto queda clara la importancia de comprender el funcionamiento de los dispositivos de entrada/salida para poder utilizarlos adecuadamente en un proyecto, y acudir a la documentación del componente cuando sea necesaria información adicional.

Es un punto fundamental para los desarrolladores avanzados poder tener varios niveles de abstracción, haciendo más cómoda e intuitiva la interacción con los periféricos. Esto además nos permite la escalabilidad y la reutilización del código, ya que estas funciones serán reutilizadas por otros proyectos o programas gracias a que no se han implementado específicamente para un proyecto en concreto.

La organización del programa en una máquina de estados también es imprescindible para poder entender correctamente el funcionamiento del sistema, y para que este funcione de una forma ordenada y lógica, sin repetir de forma innecesaria fragmentos de código.

Siguiendo estas pautas se puede implementar un programa complejo que posteriormente se pueda reutilizar para otros con distinto fin sin necesidad de cambiar el código, haciendo más fácil la vida del programador en un futuro.

Anexo: Partes representativas del código

Configuración y uso del timer 2

```
/**
 * Pre: ---
 * Post: Inicializa el timer 2
 */
void timer2_inicializar(void) {
    /* Establece la rutina de servicio para TIMER2 */
    pISR_FIQ = (unsigned) timer2_ISR;
    /* Configura el Timer2 */
    rTCFG0 = rTCFG0 & ~(0xff00); // factor de preescalado mínimo, para aumentar la precisión
    rTCFG1 = rTCFG1 &
        ~(0xf00); // selecciona la entrada del mux que proporciona el reloj. La 00 corresponde a
    un divisor de 1/2.
    rTCMPB2 = 0; // valor de comparación
    rTCNTB2 = 65535; // valor inicial de cuenta (la cuenta es descendente)
    rINTMSK = rINTMSK & ~(BIT_GLOBAL | BIT_TIMER2); // Enmascara todas las líneas excepto
    Timer2 y el bit global
}

/**
 * Pre: El timer 2 ha de estar inicializado
 * Post: Comienza la cuenta del timer 2
 */
void timer2_empezar(void) {
    rTCON = (rTCON & ~(0xf000)) | 0x2000;
    /* iniciar timer (bit 0) con auto-reload (bit 3)*/
    rTCON = (rTCON & ~(0xf000)) | 0x9000;
    timer2_num_int = 0;
}

/**
 * Pre: Se ha de haber comenzado una cuenta con el timer 2
 * Post: Devuelve la cuenta del timer 2
 */
int timer2_leer(void) {
    int timer_actual = (timer2_num_int * 65535) + (65535 - rTCNT02);
}
```

```

/* Si durante la operaci?n ha cambiado timer2_num_int, implica que se ha terminado
la cuenta y que se ha activado la subrutina */
if (timer2_num_int * 65535 > timer_actual) return (timer2_num_int * 65535) / 32;
else return timer_actual / 32;
}

```

Copia del código a la memoria RAM

```

*****
/* Copy memory */
*****
LDR r0,=0x0
LDR r1,=Image_RO_Base
LDR r3,=Image_ZI_Base
LoopRw:
    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:
    cmp r0, r1
    strcc r3, [r0], #4
    bcc LoopZI
    ldr pc,=Init_Codigo
Init_Codigo:
    nop
    nop
    nop

```

Máquina de estados principal del juego

// Máquina de estados principal del juego

*// La función reversi8 es la máquina de estados y las diferentes funciones son los estados de esta máquina de estados
// y sus transiciones*

```
static int tiempo_juego = 0;
```

```
static int tiempo_procesamiento = 0;
```

```

#define ESTADO_INICIO_DEL_JUEGO 1
#define ESTADO_MUESTRA_TABLERO 2
#define ESTADO_MUESTRA_ZOOM 3
#define ESTADO_MUESTRA_PARPADEO 4
#define ESTADO_MUESTRA_SCORE 5
#define ESTADO_PASAR_O_FINALIZAR 6
#define ESTADO_PROCESA_MOVIMIENTO 7
#define ESTADO_PASAR 8
static int estado_juego = ESTADO_INICIO_DEL_JUEGO;

```

```

int fn_estado_inicio_del_juego(char candidatas[DIM][DIM]) {
    init_table(tablero, candidatas);
    LcdM_Show_Instructions();
    tiempo_procesamiento = 0;
    tiempo_juego = 0;
    // Espera pulsaci?n

```

```

    while (estado_botones() == BOTON_NO_PULSADO && estado_tp() == TP_NO_PULSADO) {
    }
    return ESTADO_MUESTRA_TABLERO;
}

#define ESTADO_TABLERO_NO_ZOOM_ESPERA_FILA 0
#define ESTADO_TABLERO_NO_ZOOM_ESPERA_COLUMNNA 1
#define ESTADO_TABLERO_NO_ZOOM_INTRODUCIENDO_FILA 2
#define ESTADO_TABLERO_NO_ZOOM_INTRODUCIENDO_COLUMNNA 3
//Devuelve el siguiente estado
int fn_estado_muestra_tablero(int *codigo_pulsacion, int *estado_filas, int *estado_columnas) {
    timer1_start();
    LcdM_Show_Board_No_Zoom(tablero);
    D8Led_symbol(15);
    int estado_tp = 0;
    int estado_bt = 0;
    *estado_filas = 0;
    *estado_columnas = 0;
    int estado_ms_tp = ESTADO_TABLERO_NO_ZOOM_ESPERA_FILA;
    LcdM_Show_Processing_Time(tiempo_procesamiento);
    LcdM_Show_Game_Time(tiempo_juego);
    while (1) {
        if(timer1_leer() > 1){
            timer1_start();
            ++tiempo_juego;
            LcdM_Show_Game_Time(tiempo_juego);
        }
        estado_tp = estado_tp_no_zoom();
        estado_bt = estado_botones();
        if (estado_tp != TP_NO_ZOOM_NO_PULSADO && estado_tp !=
TP_NO_ZOOM_FUERA_PANTALLA) {
            // Se ha pulsado el tp
            switch (estado_tp) {
                case TP_NO_ZOOM_PASAR:
                    return ESTADO_PASAR_O_FINALIZAR;
                default:
                    // Pulsado en regi?n
                    *codigo_pulsacion = estado_tp;
                    return ESTADO_MUESTRA_ZOOM;
            }
        } else if (estado_bt == BOTON_PULSADO_IZQUIERDA) {
            switch (estado_ms_tp) {
                case ESTADO_TABLERO_NO_ZOOM_ESPERA_FILA:
                    D8Led_symbol(*estado_filas);
                    estado_ms_tp = ESTADO_TABLERO_NO_ZOOM_INTRODUCIENDO_FILA;
                    break;
                case ESTADO_TABLERO_NO_ZOOM_ESPERA_COLUMNNA:
                    D8Led_symbol(*estado_columnas);
                    estado_ms_tp =
ESTADO_TABLERO_NO_ZOOM_INTRODUCIENDO_COLUMNNA;
                    break;
                case ESTADO_TABLERO_NO_ZOOM_INTRODUCIENDO_FILA:
                    *estado_filas = aumentarContador(*estado_filas);
                    D8Led_symbol(*estado_filas);
                    break;
                case ESTADO_TABLERO_NO_ZOOM_INTRODUCIENDO_COLUMNNA:
                    *estado_columnas = aumentarContador(*estado_columnas);
                    D8Led_symbol(*estado_columnas);
                    break;
            }
        } else if (estado_bt == BOTON_PULSADO_DERECHA) {
            switch (estado_ms_tp) {
                case ESTADO_TABLERO_NO_ZOOM_INTRODUCIENDO_FILA:
                    estado_ms_tp = ESTADO_TABLERO_NO_ZOOM_ESPERA_COLUMNNA;
                    D8Led_symbol(12);
                    break;

```

```

        case ESTADO_TABLERO_NO_ZOOM_INTRODUCIENDO_COLUMNNA:
            if (*estado_filas == 8 ||
                *estado_columnas == 8) {// Se ha introducido algun 8, asi que ir al estado

                return ESTADO_PASAR_O_FINALIZAR;
            } else {
                return ESTADO_MUESTRA_PARPADEO;
            }
        }
    }
}

```

```

#define MUESTRA_ZOOM_8_LED_SYMBOL 10
int fn_estado_muestra_zoom(int region_tablero, int *estado_filas, int *estado_columnas) {
    timer1_start();
    LcdM_Show_Board_Zoom(tablero, region_tablero);
    D8Led_symbol(MUESTRA_ZOOM_8_LED_SYMBOL);
    int estado_tp = 0;
    int estado_bt = 0;
    LcdM_Show_Processing_Time(tiempo_procesamiento);
    LcdM_Show_Game_Time(tiempo_juego);
    while (1) {
        if(timer1_leer() > 1){
            timer1_start();
            ++tiempo_juego;
            LcdM_Show_Game_Time(tiempo_juego);
        }
        estado_tp = estado_tp_zoom();
        estado_bt = estado_botones();
        if (estado_tp != TP_ZOOM_NO_PULSADO ) {
            //Se ha pulsado una casilla o pasar
            switch (estado_tp) {
                case TP_ZOOM_PASAR:
                    return ESTADO_PASAR_O_FINALIZAR;
                case TP_ZOOM_FUERA_PANTALLA:
                    return ESTADO_MUESTRA_TABLERO;
                default:
                    // Pulsado en regi?n
                    estado_tp = estado_tp - 1;
                    *estado_columnas = estado_tp % 4;
                    *estado_filas = estado_tp / 4;
                    switch(region_tablero){
                        case 1:
                            break;
                        case 2:
                            *estado_columnas = *estado_columnas + 4;
                            break;
                        case 3:
                            *estado_filas = *estado_filas + 4;
                            break;
                        case 4:
                            *estado_filas = *estado_filas + 4;
                            *estado_columnas = *estado_columnas + 4;
                            break;
                    }
                    //Sacar fila y columna
                    return ESTADO_MUESTRA_PARPADEO;
                }
            } else if (estado_bt != BOTON_NO_PULSADO) {
                return ESTADO_MUESTRA_TABLERO;
            }
        }
    }
}

```

```

#define MUESTRA_PARPADEO_TIEMPO_PARPADEO 5

```



```

#define MUESTRA_PARPADEO_8_LED_SYMBOL 10
int fn_estado_muestra_parpadeo(int estado_filas, int estado_columnas) {
    timer1_start();
    LcdM_Show_Board_No_Zoom(tablero);
    D8Led_symbol(MUESTRA_PARPADEO_8_LED_SYMBOL);
    int cuenta_timer1 = 0;
    LcdM_Show_Processing_Time(tiempo_procesamiento);
    LcdM_Show_Game_Time(tiempo_juego);
    int ultimo_timer_sumado = 0;
    while (estado_botones() == BOTON_NO_PULSADO && estado_tp() == TP_NO_PULSADO &&
timer1_leer() < MUESTRA_PARPADEO_TIEMPO_PARPADEO) {
        int t1l= timer1_leer();
        if (t1l > cuenta_timer1) {
            cuenta_timer1 = timer1_leer();
            LcdM_Show_Board_Position_Selected(estado_filas * 8 + estado_columnas + 1);
        }
        if((t1l - ultimo_timer_sumado) > 1 ){
            ++tiempo_juego;
            LcdM_Show_Game_Time(tiempo_juego);
            ultimo_timer_sumado = t1l;
        }
    }
    cuenta_timer1 = timer1_leer();
    if (cuenta_timer1 < MUESTRA_PARPADEO_TIEMPO_PARPADEO) {
        // Implica que o bien se ha pulsado sobre el boton o la pantalla
        return ESTADO_MUESTRA_TABLERO;
    }else{
        return ESTADO_PROCESA_MOVIMIENTO;
    }
}

int fn_estado_pasar_o_finalizar() {
    LcdM_Show_Skip();
    int estado_tp, estado_bt;
    while (1) {
        estado_tp = estado_tp_pasar_o_finalizar();
        estado_bt = estado_botones();
        if(estado_bt != BOTON_NO_PULSADO){
            switch (estado_bt){
                case BOTON_PULSADO_IZQUIERDA:
                    return ESTADO_PASAR;
                case BOTON_PULSADO_DERECHA:
                    return ESTADO_MUESTRA_SCORE;
            }
        }else if(estado_tp != TP_PASAR_O_FINALIZAR_NO_PULSADO){
            switch (estado_tp){
                case TP_PASAR_O_FINALIZAR_PASAR:
                    return ESTADO_PASAR;
                case TP_PASAR_O_FINALIZAR_FINALIZAR:
                    return ESTADO_MUESTRA_SCORE;
                case TP_PASAR_O_FINALIZAR_VOLVER:
                    return ESTADO_MUESTRA_TABLERO;
            }
        }
    }
}

int fn_estado_muestra_score() {
    int puntos_negras, puntos_blancas;
    contar(tablero,
        &puntos_blancas, &puntos_negras);
    LcdM_Show_Score(puntos_negras, puntos_blancas);
    // Espera pulsaci?n
    while (estado_botones() == BOTON_NO_PULSADO && estado_tp() == TP_NO_PULSADO) {
    }
    return ESTADO_INICIO_DEL_JUEGO;
}

```

```
}
```

```
int fn_estado_pasar(char candidatas[DIM][DIM]) {
    timer2_empezar();
    char f, c; // fila y columna elegidas por la m?quina para su movimiento
    // escribe el movimiento en las variables globales fila columna
    int done = elegir_mov(candidatas, tablero, &f, &c);
    int resultado;
    if(done == -1){
        resultado = ESTADO_MUESTRA_SCORE;
    }else{
        tablero[f][c] = FICHA_BLANCA;
        actualizar_tablero(tablero, f, c, FICHA_BLANCA);
        actualizar_candidatas(candidatas, f, c);
        resultado = ESTADO_MUESTRA_TABLERO;
    }
    tiempo_procesamiento += timer2_leer();
    return resultado;
}
```

```
int fn_estado_procesa_movimiento(int estado_filas, int estado_columnas, char
candidatas[DIM][DIM]){
    timer2_empezar();
    tablero[estado_filas][estado_columnas] = FICHA_NEGRA;
    actualizar_tablero(tablero, estado_filas, estado_columnas, FICHA_NEGRA);
    actualizar_candidatas(candidatas, estado_filas, estado_columnas);
    char f, c;
    // escribe el movimiento en las variables globales fila columna
    int done = elegir_mov(candidatas, tablero, &f, &c);
    if (done != -1) {
        tablero[f][c] = FICHA_BLANCA;
        actualizar_tablero(tablero, f, c, FICHA_BLANCA);
        actualizar_candidatas(candidatas, f, c);
    }
    tiempo_procesamiento += timer2_leer();
    int puntos_negras, puntos_blancas;
    contar(tablero,
        &puntos_blancas, &puntos_negras);
    if((puntos_negras + puntos_blancas) == 64)
        return ESTADO_MUESTRA_SCORE;
    else
        return ESTADO_MUESTRA_TABLERO;
}
```

```
void reversi8() {
    char __attribute__((aligned (8))) candidatas[DIM][DIM] =
    {
        {NO, NO, NO, NO, NO, NO, NO, NO},
        {NO, NO, NO, NO, NO, NO, NO, NO},
        {NO, NO, NO, NO, NO, NO, NO, NO},
        {NO, NO, NO, NO, NO, NO, NO, NO},
        {NO, NO, NO, NO, NO, NO, NO, NO},
        {NO, NO, NO, NO, NO, NO, NO, NO},
        {NO, NO, NO, NO, NO, NO, NO, NO},
        {NO, NO, NO, NO, NO, NO, NO, NO}
    };
    int filas_, columnas_, estado_pulsacion_; // fila y columna elegidas por la m?quina para su
movimiento
    estado_juego = ESTADO_INICIO_DEL_JUEGO;
    timer1_start();
    while (1) {
        switch (estado_juego) {
            case ESTADO_INICIO_DEL_JUEGO:
                estado_juego = fn_estado_inicio_del_juego(candidatas);
                break;
            case ESTADO_MUESTRA_TABLERO:
```

```

        estado_juego = fn_estado_muestra_tablero(&estado_pulsacion_, &filas_,
&columnas_);
        break;
    case ESTADO_MUESTRA_ZOOM:
        estado_juego = fn_estado_muestra_zoom(estado_pulsacion_ , &filas_, &columnas_);
        break;
    case ESTADO_PASAR_O_FINALIZAR:
        estado_juego = fn_estado_pasar_o_finalizar();
        break;
    case ESTADO_MUESTRA_PARPADO:
        estado_juego = fn_estado_muestra_parpado(filas_,columnas_);
        break;
    case ESTADO_MUESTRA_SCORE:
        estado_juego = fn_estado_muestra_score();
        break;
    case ESTADO_PROCESA_MOVIMIENTO:
        estado_juego = fn_estado_procesa_movimiento(filas_,columnas_,candidatas);
        break;
    case ESTADO_PASAR:
        estado_juego = fn_estado_pasar(candidatas);
    }
}
}
}

```

Gestión de excepciones

```

#define TYPE_UNDEF 0;

#define TYPE_PREFETCH_ABORT 1;
#define TYPE_DATA_ABORT 2;
#define TYPE_SWI 3;
void exception_manager(void) {
    unsigned int cpsr, type, lr;
    asm("mov %0,lr\n":"=r" (lr));
    asm("mrs %0,CPSR\n":"=r" (cpsr));
    int mask = cpsr & 0x1f;
    switch (mask) {
        case 0x17:
            type = TYPE_DATA_ABORT;
            break;
        case 0x13:
            type = TYPE_SWI;
            break;
        default:
            type = TYPE_UNDEF;
    }
    exception_manager_save(type, lr);
}
void exception_manager_pabort(void) {
    unsigned int lr, type;
    asm("mov %0,lr\n":"=r" (lr));
    type = TYPE_PREFETCH_ABORT;
    exception_manager_save(type, lr);
}
void exception_manager_save(unsigned int type, unsigned int lr) {
    push_debug(type, lr);
    D8Led_symbol(type);
    int i = 0;
    while (1) {
        if (i == 0) {
            D8Led_symbol(type);
            led1_on();
            led2_off();
        }
    }
}

```

```

        i = 1;
    } else {
        D8Led_symbol(8);
        led2_on();
        led1_off();
        i = 0;
    }
    DelayTime(100);
}
}
void init_exceptions() {
    pISR_UNDEF = (unsigned) exception_manager;
    pISR_PABORT = (unsigned) exception_manager_pabort;
    pISR_SWI = (unsigned) exception_manager;
    pISR_DABORT = (unsigned) exception_manager;
}

```

Inicialización y máquina de estados de la gestión de pulsaciones del botón

```

#define ESTADO_BOTON_ESPERANDO 0
#define ESTADO_BOTON_PULSADO 1
#define ESTADO_BOTON_MANTENIDO 2
#define ESTADO_BOTON_REBOTES_INICIALES 3
#define ESTADO_BOTON_REBOTES_FINALES 4

```

```

volatile static int estado_actual;
volatile static int boton_en_gestion;
volatile static int boton_pulsado;
volatile static int interrupciones_mantenido;

```

```

void Botones_ISR(void) {
    /* Identificar la interrupcion (hay dos pulsadores)*/
    int which_int = rEXTINTPND;
    switch (which_int) {
        case 4:
            boton_pulsado = BOTON_PULSADO_IZQUIERDA;
            boton_en_gestion = BOTON_PULSADO_IZQUIERDA;
            break;
        case 8:
            boton_pulsado = BOTON_PULSADO_DERECHA;
            boton_en_gestion = BOTON_PULSADO_DERECHA;
    }
    estado_actual = ESTADO_BOTON_PULSADO;
    gestor_anti_rebotes();
    /* Finalizar ISR */
    rEXTINTPND = 0xf; // borra los bits en EXTINTPND
    rI_ISPC |= BIT_EINT4567; // borra el bit pendiente en INTPND
}
void Botones_anti_inicializar(void) {
    /* Configuración del controlador de interrupciones. Estos registros están definidos en 44b.h */
    pISR_EINT4567 = (int) Botones_ISR;
    rPCONG = 0xffff; // Establece la función de los pines (EINT0-7)
    rPUPG = 0x0; // Habilita el "pull up" del puerto
    rEXTINT = rEXTINT | 0x22222222; // Configura las líneas de int. como de flanco de bajada
    /* Por precaución, se vuelven a borrar los bits de INTPND y EXTINTPND */
    rI_ISPC |= (BIT_EINT4567);
    rEXTINTPND = 0xf;
    rINTMSK = rINTMSK & ~(BIT_GLOBAL | BIT_EINT4567);
    // inicializa el timer que se encarga de gestionar los rebotes
}

```

```

    estado_actual = ESTADO_BOTON_ESPERANDO;
    boton_pulsado = BOTON_NO_PULSADO;
    boton_en_gestion = BOTON_NO_PULSADO;
    timer0_inicializar();
}

void timer0_ISR(void) {
    gestor_anti_rebotes();
    /* borrar bit en I_ISPC para desactivar la solicitud de interrupci?n*/
    rI_ISPC |= BIT_TIMER0;
}

void timer0_inicializar(void) {
    /* Establece la rutina de servicio para TIMER0 */
    pISR_TIMER0 = (unsigned) timer0_ISR;
    /* Configura el Timer0 */
    rTCFG0 = rTCFG0 | 0x255; // factor de preescalado maximo, para aumentar el retardo de los
pulsos
    rTCFG1 = (rTCFG1 & ~(0xf)) |
0x4; // selecciona la entrada del mux que proporciona el reloj.
    rTCNTB0 = TIEMPO_RETARDO_REBOTES;
    rTCMPB0 = 0; // valor de comparaci?n
}

void gestor_anti_rebotes(void) {
    switch (estado_actual) {
        case ESTADO_BOTON_PULSADO:
            // Se inicializa el timer
            rINTMSK = rINTMSK & ~(BIT_GLOBAL |
bit global
BIT_TIMER0); // Emascara todas las lineas excepto Timer0 y el

            rTCON = (rTCON & ~(0xf)) | 0x2;
            rTCON = (rTCON & ~(0xf)) | 0x09;
            rINTMSK = rINTMSK | (BIT_EINT4567);
            estado_actual = ESTADO_BOTON_REBOTES_INICIALES;
            interrupciones_mantenido = 0;
            break;
        case ESTADO_BOTON_REBOTES_INICIALES:
            rTCNTB0 = TIEMPO_ENCUESTA_PERIODICA;
            rTCON = (rTCON & ~(0xf)) | 0x2;
            rTCON = (rTCON & ~(0xf)) | 0x09;
            estado_actual = ESTADO_BOTON_MANTENIDO;
            break;
        case ESTADO_BOTON_MANTENIDO:
            if (((boton_en_gestion == BOTON_PULSADO_DERECHA) && ((rPDATG & (0x1 <<
7)) != 0))
|| ((boton_en_gestion == BOTON_PULSADO_IZQUIERDA) && ((rPDATG & (0x1
<< 6)) !=
0))) { // Si alguna de
las posiciones es igual a 0 implica que esta pulsado
            rTCNTB0 = TIEMPO_RETARDO_REBOTES;
            rTCON = (rTCON & ~(0xf)) | 0x2;
            rTCON = (rTCON & ~(0xf)) | 0x09;
            estado_actual = ESTADO_BOTON_REBOTES_FINALES;
        } else if (interrupciones_mantenido == TIEMPO_MANTENIDO_PULSACION) {
            interrupciones_mantenido = 0;
            boton_pulsado = boton_en_gestion;
        } else {
            ++interrupciones_mantenido;
        }
        break;
        case ESTADO_BOTON_REBOTES_FINALES:
            rEXTINTPND = 0xf; // borra los bits en EXTINTPND
            rI_ISPC |= BIT_EINT4567; // borra el bit pendiente en INTPND
            rINTMSK = rINTMSK & ~(BIT_EINT4567);
            rINTMSK = rINTMSK | (BIT_TIMER0);
            //Elimino interrupciones timer
            rTCON = rTCON & ~(0xf);
    }
}

```

```

        estado_actual = ESTADO_BOTON_ESPERANDO;
        boton_en_gestion = BOTON_NO_PULSADO;
        break;
    case ESTADO_BOTON_ESPERANDO:
        //Vacio
        break;
    }
}

// Devuelve el botón pulsado (BOTON_PULSADO_IZQUIERDA o BOTON_PULSADO_DERECHA) en el caso
de que lo esté alguno o BOTON_NO_PULSADO, en caso de que no lo esté ninguno

int estado_botones(void) {
    int devolver = boton_pulsado;
    boton_pulsado = BOTON_NO_PULSADO;
    return devolver;
}

```

Inicialización y transformación de coordenadas en el touch screen

```

volatile static unsigned int Xmax = 0;

volatile static unsigned int Ymax = 0;
volatile static unsigned int Xmin = 2000;
volatile static unsigned int Ymin = 2000;
volatile static unsigned int x = 0;
volatile static unsigned int y = 0;
volatile static int pulsa = 0;
volatile static int calibraciones = 0;
/**
 * Función auxiliar que modifica Xmax e Ymax si se introduce una x o una y superior a la
Xmax e Ymax actual
 * */
void ajustar_x_y() {
    if (x > Xmax) {
        Xmax = x;
    } else if (x < Xmin) {
        Xmin = x;
    }
    if (y > Ymax) {
        Ymax = y;
    } else if (y < Ymin) {
        Ymin = y;
    }
}
/**
 * Función auxiliar que calibra la pantalla las 8 primeras veces que se le invoca
 * */
int calibrar_tp() {
    if (calibraciones < 8) {
        ajustar_x_y();
        calibraciones += 1;
        return 1;
    } else {
        return 0;
    }
}

```

```

/*****
*****
* name:      TSInt
* func:      TouchScreen interrupt handler function
* para:      none
* ret:       none
* modify:
* comment:
*****
*****/
void TSInt(void) __attribute__((interrupt("IRQ")));
void TSInt(void) {
    int i;
    // X position Read
    rPDATE = 0x68;
    rADCCON = 0x1 << 2;          // AIN1
    //DelayTime(5000);           // O0 delay to set up the next channel
    DelayTime(25000);           // O2 delay to set up the next channel
    x = 0;
    for (i = 0; i < 5; i++) {
        rADCCON |= 0x1;          // Start X-position A/D conversion
        while (rADCCON & 0x1);   // Check if Enable_start is low
        while (!(rADCCON & 0x40)); // Check ECFLG
        x += (0x3ff & rADCSTAT);
    }
    // read X-position average value
    x = x / 5;
    // Y position Read
    rPDATE = 0x98;
    rADCCON = 0x0 << 2;          // AINO
    //DelayTime(5000);           // O0 delay to set up the next channel
    DelayTime(25000);           // O2 delay to set up the next channel
    y = 0;
    for (i = 0; i < 5; i++) {
        rADCCON |= 0x1;          // Start Y-position conversion
        while (rADCCON & 0x1);   // Check if Enable_start is low
        while (!(rADCCON & 0x40)); // Check ECFLG
        y += (0x3ff & rADCSTAT);
    }
    // read Y-position average value
    y = y / 5;
    if ((calibrar_tp() == 1) || ((x >= Xmin) && (x <= Xmax) && (y >= Ymin) && (y <=
Ymax))) {
        // Ya esta la pantalla calibrada y la posicion es valida o no esta calibrada
        pulsa = 1;
    }
    rPDATE = 0xb8;               // should be enabled
    // DelayTime(100000); //ParaO0 // delay to set up the next channel
    DelayTime(500000); //ParaO2 // delay to set up the next channel
    rI_ISPC = BIT_EINT2;        // clear pending_bit
}
/**
* Pre: Se ha inicializado el touch screen
* Post: Obtiene la coordenada x de la ?ltima pulsaci?n (el sistema de referencia es el que
utiliza el lcd
* para dibujar los p?xeles)
*/
int getX(void) {
    int temp = x - Xmin;

```

```

    int dif = Xmax - Xmin;
    return (320 * temp) / dif;
}
/**
 * Pre: Se ha inicializado el touch screen
 * Post: Obtiene la coordenada y de la ?ltima pulsaci?n (el sistema de referencia es el que
utiliza el lcd
 * para dibujar los p?xeles)
 */
int getY(void) {
    int temp = Ymax - y;
    int dif = Ymax - Ymin;
    return ((240 * temp) / dif);
}
/**
 * Pre: Se ha inicializado el touch screen
 * Post: Devuelve 1 si se ha pulsado el touch screen desde la ?ltima vez que se invoc? a esta
funci?n
 */
int haPulsado(void) {
    if (pulsar == 0) {
        return 0;
    } else {
        pulsar = 0;
        return 1;
    }
}
}
/**
 * Pre: ---
 * Post: Inicializa el touch screen
 */
void TS_init(void) {
    rI_ISPC |= BIT_EINT2;           // clear pending_bit
    rPUPE = 0x0;                   // Pull up
    rPDATE = 0xb8;                  // should be enabled
    DelayTime(500);
    rEXTINT |= 0x200;               // falling edge trigger
    rCLKCON = 0x7ff8;              // enable clock
    rADCPSR = 0x1; //0x4;           // A/D prescaler
    pISR_EINT2 = (int) TSInt;      // set interrupt handler
    rI_ISPC |= BIT_EINT2;          // clear pending_bit
    rINTMSK = rINTMSK & ~(BIT_GLOBAL | BIT_EINT2); // rINTMSK &
}

```