PROYECTO HARDWARE

Memoria
Práctica 2 y 3

Curso 2018-2019

Autor:
Rubén Rodríguez Esteban 737215
Adrían Samatán Alastuey 738455

Índice de contenidos

1. Resumen

En este documento se procede a describir la metodología empleada para poder desarrollar el soporte necesario para jugar al juego Reversi, implementado en la primera práctica, desde la placa. Para ello, ha sido necesario gestionar entrada/salida mediante el uso de los botones y del display de 8 segmentos de la placa.

Para realizar el trabajo, se tuvo que analizar la documentación facilitada por el profesorado. Esta fue una de las tareas más difíciles, dado que fue necesario entender la gestión del vector de las interrupciones, el material de la universidad complutense donde figura la documentación correspondiente a la gestión de entrada y salida de la placa, y el manual de la arquitectura arm4v7 para poder comprender los espacios y reservas de direccionamiento de la memoria y la gestión de los cambios de modo del procesador. Seguidamente se estudió un proyecto de prueba en el que se empleaban algunos de los periféricos descritos en esta práctica como son los botones y timers.

Para lograr hacer funcionar el juego del reversi sin necesidad de trabajar físicamente con la placa, la primera tarea fue lograr abstraerse del hardware lo máximo posible. Para ello ha sido necesario simular el comportamiento de los periféricos y encapsular todas las funciones de entrada y salida con las que se ha trabajado de modo que puedan ejecutarse en la placa si se dispone de ella, o en caso contrario, empleando el simulador. Para llevar a cabo la simulación se han usado variables globales en memoria para poder reflejar los distintos estados de los periféricos, junto a programación condicional haciendo uso de directivas del preprocesador (#ifdef/#else/#endif) para poder controlar el paso de una modalidad a otra.

Posteriormente se ha procedido a gestionar el tratamiento de excepciones ya que durante la ejecución del programa pueden darse situaciones de error. Por seguridad ante estos casos se han tenido en cuenta tanto los tipos de excepciones como las instrucciones que las han producido. Además dada la elevada recurrencia de algunas funciones se ha optado por elaborar una pila de depuración donde se han anotado los eventos producidos y el instante en µs en el que han tenido lugar.

Seguidamente se procedió a incorporar al código fuente del reversi implementado en la práctica anterior los distintos periféricos con los que se comunica. Paralelamente, debido a la gran cantidad de fallos que podía experimentar el programa a la hora de ejecutarse, se optó por implementar un latido del programa con la finalidad de poder observar si el programa está vivo o si de lo contrario ha sufrido un fallo en algún momento de la ejecución.

Posteriormente se eliminaron los rebotes en los pulsadores de la placa, ya que al pulsarlos existe la posibilidad de que se produzcan por ser dispositivos reales mecánicos. Para ello fue necesario determinar cuántos accesos a rutina había al presionar el botón, así como los tiempos que debían transcurrir para poder evitar los rebotes, los cuales dependían de la

placa utilizada. Los rebotes fueron gestionados mediante la elaboración de una máquina de estados, explicada en apartados posteriores.

Una vez efectuados todos estos pasos se diseñó una nueva máquina de estados para poder hacer posible la evaluación y control de los movimientos de inserción de las fichas del jugador usando los botones y el dispositivo 8led para poder ver los valores de la posición.

Llegados a este punto, se continuó con la elaboración de una interfaz gráfica del juego para que el usuario pudiera jugar también usando la pantalla táctil de la placa. Para realizar esta tarea fue necesario comprender el funcionamiento básico del LCD a través de un proyecto facilitado por el profesorado que contenía un conjunto de funciones básicas para el manejo del LCD y la pantalla táctil de la placa. Una vez comprendido todo el material proporcionado la interfaz gráfica del juego fue diseñada e incorporada al código principal del juego.

Por último, se aseguró que la ejecución se realizase en modo usuario. También se cargó el programa en la memoria flash de la placa, siendo posible ejecutarlo directamente desde la misma sin necesitar un PC.

2. Introducción

Durante la elaboración de este proyecto se ha procedido a manipular el código fuente del reversi implementado en la primera práctica de la asignatura. Inicialmente el juego del reversi estaba solo desarrollado para poder jugar a través del simulador. De esta forma todo componente del juego como es el tablero, las fichas, y las posiciones sólo podían observarse viendo el mapa de memoria del juego desde el simulador.

En este proyecto se han realizado las manipulaciones necesarias para lograr la ejecución del juego tanto en simulador como en placa, y además se ha incorporado la utilización de la pantalla táctil de la placa de forma que aumenta el grado de jugabilidad ya que el jugador puede ver visualmente.

3. Objetivos

En esta sección se exponen cuales son los objetivos del trabajo realizado en el proyecto

- Tener facilidad y capacidad para depurar un código con varias fuentes de interrupción activas.
- Gestionar la entrada/salida con dispositivos básicos, asignando valores a los registros internos de la placa desde un programa en C.
- Desarrollar en C las rutinas de tratamiento de interrupción.
- Aprender a utilizar periféricos, como los temporizadores internos de la placa, el teclado, la pantalla táctil, etcétera.
- Tener dominio y capacidad para depurar eventos concurrentes asíncronos.
- Entender los modos de ejecución del procesador y el tratamiento de excepciones.
- Gestionar el tiempo de trabajo del proyecto correctamente en función de la disponibilidad de acceso al sistema específico.
- Utilizar la pantalla LCD táctil para visualizar e introducir movimientos en el tablero.
- Cargar código en la memoria Flash de la placa mediante el estándar JTAG, de forma que al encenderla se pueda jugar sin necesidad de conectarse ni descargar el programa.

4. Metodología

En este apartado del documento se procede a explicar de manera más detallada la secuencia de pasos realizados para cumplir con los objetivos del proyecto junto con el esquema del trabajo que se ha seguido, es decir, todos ficheros y funciones que lo componen junto con todas decisiones tomadas de diseño y sus correspondientes justificaciones.

4.1. Estudio de la documentación

Se comenzó por estudiar la documentación proporcionada para poder programar el vector de excepciones, gestionar la configuración de los periféricos empleados, concretamente el timer0, los botones de la placa, el display de 8 segmentos, y se repasaron las instrucciones disponibles en la arquitectura ARMv7 (ARM quick reference card) y el protocolo de llamadas a funciones a seguir (AATPCS).

4.2. Estudio y depuración de las fuentes

Una vez analizada la documentación, se estudiaron los códigos fuente, principalmente los ficheros 8led.c, 8led.h, button.c y button.h con el objetivo de poder entender el funcionamiento de los mismos. También se realizaron diversas pruebas depurando el código. Estas se centraron en entender mejor el código generado por el compilador en los distintos flags de optimización, cómo se efectúan los cambios de contexto para poder detener la ejecución del programa principal en un punto concreto y atender a las rutinas de interrupción así como la gestión de la dirección de retorno para poder reanudar la ejecución del programa en el mismo punto en el que se pausó para atender las interrupciones.

4.3 Abstracción del hardware

Como se ha explicado anteriormente se deseaba que se pudiera trabajar con gran parte de este proyecto sin necesidad de tener físicamente la placa. Para poder conseguirlo era necesario abstraerse del hardware simulando el comportamiento de los periféricos y encapsulando todas las funciones de entrada y salida con las que se trabajaba.

La abstracción del hardware se ha realizado en todos los periféricos con los que se comunica el programa, es decir, en los botones y en la gestión de los rebotes, en el display de 8 segmentos, en el timer0 y en el timer2. Para poder determinar en qué modalidad de funcionamiento se está ejecutando el programa se ha usado un flag del preprocesador denominado SIM_IN_USE, añadido a la configuración del proyecto para distinguir entre la ejecución en placa o en simulador, haciendo uso de las directivas #ifdef e #ifndef. La razón por la que se ha optado por usar este mecanismo es que al ser un parámetro de configuración interno del propio proyecto, éste va a ejecutarse sin problemas independientemente tanto del computador como de la placa. Dicho flag ha sido incorporado

en el panel *"defined symbols -D" en el apartado "prepocessor"* de las opciones del compilador gcc, accesibles desde el menú de propiedades del proyecto Eclipse.
Otro aspecto muy importante de este apartado es la creación de variables estáticas para poder almacenar el estado de los periféricos en caso de usar el simulador. Para poder obtener el estado del botón en el simulador se usa la variable *boton_actual* cuyos posibles valores pueden ser BUTTON_NONE, BUTTON_IZ y BUTTON_DR, definidos en una estructura *enum*, e identificados con los enteros 0, 1 y 2, respectivamente. Esta variable es usada para determinar si se ha pulsado botón o no, y en caso afirmativo, determinar si ha sido el izquierdo o el derecho en la función *button_estado*, y para verificar los posibles rebotes en *botones_antirebotes_gestion* donde se devuelve en µs el tiempo del timer0. Para poder gestionar tanto el timer0 como el timer2 en el simulador se han usado las variables globales *timer0_simulador_us* y *timer2_simulador_us* respectivamente para poder devolver los valores de tiempo en µs cada vez que producen interrupciones en las funciones *timer0_leer* y *timer2_leer*. Por último para poder gestionar el display de 8 segmentos se ha usado la variable *estado_8led* que guarda su valor de 0 a F, concretamente, en la funcion *D8Led_init* para iniciar el estado del led y en la función *D8Led_symbo* para guardar el estado.

En caso de que se esté empleando directamente la placa, no se trabajan con estas variables sino con los propios registros de cada periférico, descritos en el código del proyecto.

4.4 Tratamiento de las excepciones

Como se explicó en apartados anteriores, durante la ejecución del código del programa pueden ocurrir situaciones en las que el procesador detecta un error dando lugar a una excepción. En este proyecto se ha procedido a capturar las excepciones *Data Abort, Undefined y SWI (Instruction Software Interrupt).* El tratamiento de las excepciones se encuentra en el módulo exceptions. Para poder representar este tipo de interrupciones en el display de 8 segmentos. se ha asignado para cada una de ellas un valor específico, de esta manera, el código de la excepción *SWI* es el 1, el de la excepción *Data Abort* es un 2 y el de la excepción *Undefined* es un 3. La asignación a cada tipo de excepción tratada de un valor de codificación es porque son más fáciles de identificar por el programador en caso de que ocurran, por lo que la tarea de depurar y corregir el programa se hace más amena.

Para poder gestionar las interrupciones se ha empleado la variable *excepcion_tratada,* que almacena la última excepción capturada por el programa, y la variable *dir_instr_excepcion* para almacenar la dirección de memoria de la instrucción que la ha generado. La función exception_init permite activar el tratamiento de las excepciones Data Abort, SWI y Undefined. Para gestionar el tratamiento de las excepciones se emplea la función *exception_tratamiento* encargada de detectar si se ha producido o no una excepción tras la ejecución de la instrucción actual. La posible excepción capturada se almacena en la variable *excepcion_tratada,* tal y como se ha detallado anteriormente. Una vez detectada la excepción se ejecuta la función exception_bucle. Dicha función detiene la ejecución del

programa entrando en un bucle infinito donde se muestra por medio de un parpadeo constante el código de la excepción en el display de 8 segmentos.

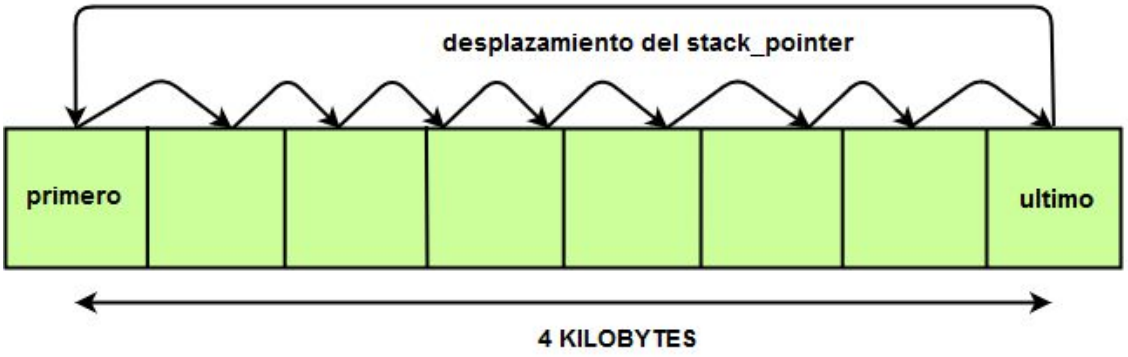
Para poder probar el correcto funcionamiento de las excepciones se han implementado unos ficheros de pruebas de pruebas se ha implementado una función llamada *exceptions_test,* localizada en el módulo tests*,* donde se han forzado los tres tipos de excepciones a capturar. Dichas pruebas son explicadas posteriormente en el apartado de pruebas.

4.5 Pila de depuración

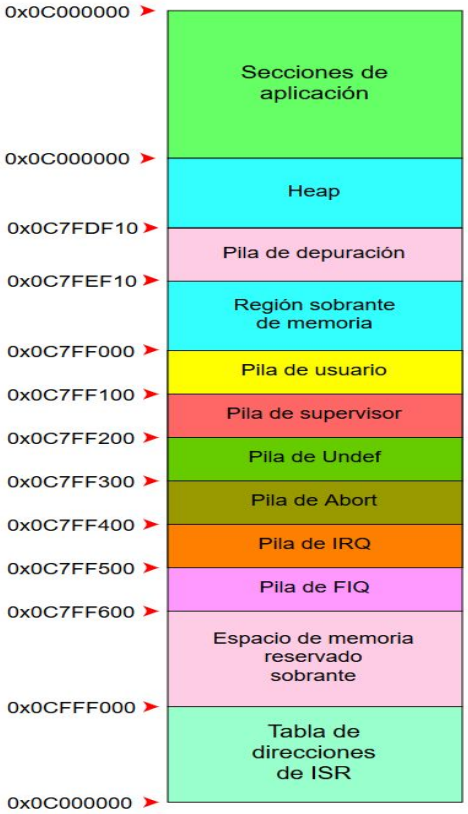
Dada la existencia de diversas funciones concurrentes en el sistema se ha optado por implementar una pila de depuración. En dicha estructura de datos se han procedido a anotar todos los hechos que resultan relevantes durante la ejecución del programa, concretamente el estado de los botones, y el instante de tiempo en el que se ha producido su interrupción. La pila se ha ubicado al final del espacio de memoria y antes del espacio reservado para las pilas del resto de modos de usuario. La pila comienza en la dirección 0xC7FDF10 y termina en la dirección 0xC7FEF10, de tal forma que la pila tiene exactamente un tamaño de 4 Kilobytes, es decir 1024 palabras de 4 bytes cada una, lo cual es un tamaño más que suficiente para la cantidad de de información que se va a almacenar. La pila se ha implementado como una lista circular controlando los límites de manera que la información de la pila no se desborde.

La pila de depuración está implementada por medio de un vector estático de longitud *TAMANYO_PILA * 4,* cuyo resultado es 4 Kilobytes. Para permitir el desplazamiento a través de la pila se usa un puntero a la cima denominado *stack_pointer*. El espacio de la pila se inicia todo a ceros antes de poder emplearla con objeto de desechar la posible información residente en esa región de memoria. La inicialización de la pila se lleva a cabo por medio de la función *pila_debug_init*, la inserción de datos en la pila se efectúa por medio de la función *push_debug.* Se pasan dos parámetros, *id_evento* y *aux_data.* El valor *id_evento* representa el tipo de evento a apilar, cuyo formato es un entero sin signo de 8 bits, dado que sólo se almacena el byte más significativo, y *aux_data,* una variable que almacena un entero sin signo de 32 bits, de los cuales se guardan en la pila 24 bits. La inserción de los datos se hace modulo TAMANYO_PILA para asegurar que nunca se va a desbordar la pila, de forma que la información residente en direcciones de memoria siguientes se mantiene totalmente a salvo de ser eliminada. El conjunto de funciones que permiten gestionar la pila se hallan en el módulo pila_depuracion.

Seguidamente se muestra una imagen que representa el mecanismo de almacenamiento de eventos en la pila de depuración.



En la siguiente captura se muestra un esquema con la región de memoria asignada a cada una de las pilas.



Si la pila de usuario crece demasiado, podría llegar a causar conflictos con la pila de depuración, puesto que se está ejecutando el proyecto sobre hardware sin sistema operativo ni software para controlar las distintas regiones de memoria. Para probar la pila se ha realizado una función llamada *pila_depuracion_test* que realiza diversas invocaciones a la función *push_debug* para verificar que los datos se insertan correctamente. Dicha función se encuentra en el módulo tests.

4.6 Integración del juego

A lo largo del desarrollo del proyecto se han ido incorporando diversos periféricos como son el timer2, los botones, y el timer0 (todavía no explicado). Para poder estructurar mejor el código del programa se ha creado la función *reversi_main* llamada desde el fichero main.c una vez los periféricos y la placa están inicializados. Esta función se encarga de gestionar toda la actividad del juego y de los periféricos en un bucle infinito, de modo que el juego permanece siempre en ejecución.

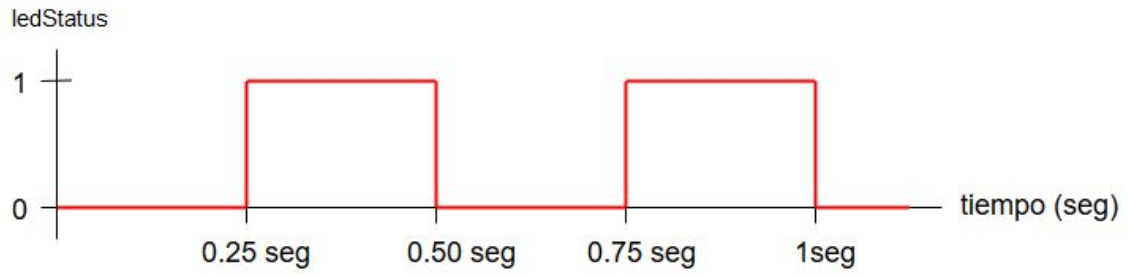
El bucle infinito residente en esta función es bastante complejo dada la gran magnitud de tarea que se gestiona en él. Como paso previo a la entrada en el bucle se llama a la función *init_table*, encargada de inicializar el tablero con las fichas. Una vez se entra en el bucle, se procede a llevar a cabo la lectura de tiempo del timer0 gracias a la función timer0_leer, se gestiona el latido (explicado en el apartado siguiente) por medio de la función *gestion_latido* y finalmente se procesa toda la lógica de movimientos del juego. Si la partida finaliza se comienza otra nueva, en caso contrario, se prosigue con la que está en vigor. La lógica del juego está gestionada en la función *reversi_procesar*.

4.7 Latido

En esta sección del documento se procede a explicar la gestión del latido, este mecanismo sirve para poder comprobar de manera visual por parte del programador que el programa desarrollado sigue vivo por medio de un parpadeo constante (latido) de un led de la placa.
El latido es una técnica altamente eficaz de detección de fallos de ejecución ya que permite determinar visualmente si el programa funciona bien en todo momento o no.

Para la implementación del latido se ha empleado el timer0. Dicho periférico sigue el mismo esquema de implementación que el timer2, aunque con algunas variaciones. El timer0 se ha configurado para que genere 50 interrupciones por segundo y para que el led de la izquierda de la placa parpadee cada 2 Hz, es decir, dos veces por segundo. Para poder realizar esta configuración se han tenido que reajustar los intervalos de tiempo a medir y el preescalado. El parpadeo del led se gestiona en la función *gestion_latido,* localizada en el módulo gestion_latido. *E*sta constante marca el tiempo que el led debe estar encendido o apagado, cuyo valor es 250000 µs, o sea, un cuarto de segundo La razón por la que toma este valor es porque el led tiene que parpadear a 2 Hz, en otras palabras, el led debe de estar dos cuartos de segundo encendido y los restantes apagado, de forma alternada.

En el siguiente cronograma se puede ver como cambia la señal del estado del led de encendido a apagado y viceversa en función del tiempo medido en segundos.



La señal *ledStatus* representa el estado de led, encendido o apagado. Cuando la señal *ledStatus* está activada el led de la placa está encendido y cuando está desactivada el led está apagado. En el gráfico puede observarse que el tiempo que tarda la señal en completar un ciclo es de medio segundo. La frecuencia al ser el inverso del periodo son 2 Hz, y por tanto el tiempo que tiene que estar el led encendido o apagado es 0.25 seg, o lo que es equivalente, 250000 µs de tiempo.

4.8 Gestión de los rebotes de los botones

Para poder jugar al reversi se van a usar los pulsadores de la placa, tal y como se comentó al principio del documento. Un problema muy importante es que los botones, por ser dispositivos mecánicos reales, al ser pulsados producen señales oscilantes llamadas rebotes. Para poder determinar si se producen o no los rebotes se ha recurrido a la pila de depuración cuyos detalles de implementación ya han sido explicados anteriormente. Adicionalmente se ha tenido que implementar por medio de una colección de métodos el dispositivo del botón a bajo nivel junto con la correspondiente rutina de servicio-interrupción. Es importante hacer énfasis en que el dispositivo button depende del hardware por completo.

El dispositivo button cuenta en su interfaz con un *enum estado_button {BUTTON_NONE, BUTTON_IZ, BUTTON_DR}* para representar los estados de los botones. También se encuentra la función *button_iniciar* que inicializa el dispositivo dejándolo listo para ser usado, la función *button_empezar* que re-activa interrupciones y reprograma la función de callback a llamar desde la RSI, *button_estado* que retorna el estado actual del botón, la función *button_deshabilitar* para deshabilitar interrupciones por botón, y por último, la función *button_ISR*, donde se ha especificado el código de la rutina al producirse la interrupción.

Se ha implementado también un módulo de más alto nivel que gestiona los posibles rebotes denominado *botones_antirebotes,* cuyas funciones son invocadas desde la función *reversi_main,* el cual utiliza los servicios proporcionados por los dispositivos button y timer0. El autómata del módulo es gestionado desde la función *botones_antirebotes_gestión* y tiene el siguiente esquema:

1. El autómata comienza en ESTADO_INICIAL*,* insertándose en la pila el instante de tiempo cero y que no hay botón pulsado. Se permanece en ese estado hasta que el usuario presiona un botón determinado, el izquierdo o el derecho.

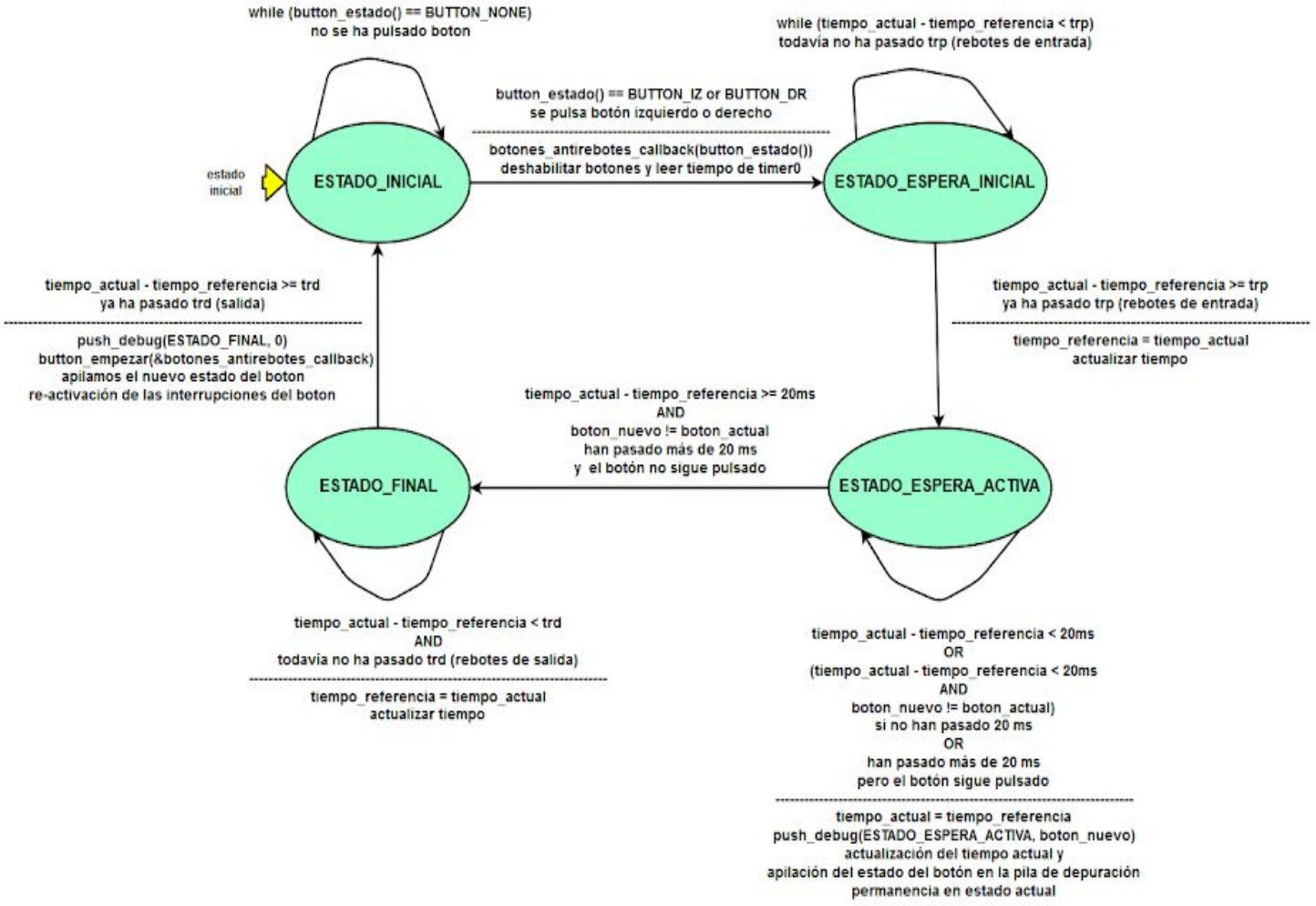
2. En el momento en el que se presiona el botón se llama a la función *botones_antirebotes_callback*. En dicha función se deshabilita la llegada de interrupciones de los botones invocando a la función *button_deshabilitar*, se almacena el botón pulsado, izquierdo o derecho en la variable *botton_actual,* se obtiene el tiempo en el que se ha producido el evento invocando a la función timer0_leer y se pasa al estado ESTADO_ESPERA_INICIAL.

3. Una vez en este estado se comprueba que ha transcurrido un tiempo de retardo inicial para eliminar los posibles rebotes de entrada llamado TIEMPO_ESPERA_INICIAL_US (trp) cuyo valor depende de las placas, y que posteriormente se explicará el procedimiento seguido para obtenerlo. Transcurrido ese retardo inicial se cambia al estado ESTADO_ESPERA_ACTIVA, donde se comprueba cada 20 ms si el usuario sigue manteniendo pulsado el botón o ha levantado el dedo. Tras el transcurso de ese retardo se evalúa si el usuario ha levantado el dedo. Para ello, se invoca a la función *button_estado*() para ver el estado actual del botón y se compara con el anterior. Se inserta en la pila el nuevo evento y se comprueba si el estado del botón anterior coincide con el actual. Si coincide significa que el usuario sigue presionando el botón, por lo que se permanece en ese estado, pero si es distinto se cambia de estado a ESTADO_FINAL.

4. En el estado ESTADO_FINAL se ha insertado, para poder eliminar los posibles de salida, un retardo final TIEMPO_ESPERA_FINAL_US (trd), cuyo valor también depende de la placa. Tras ese retardo se guarda en la pila que ya no hay botón pulsado, y se llama a la función *button_empezar* para re-activar los botones. Este autómata refleja la gestión de los rebotes tal y como se ha explicado

Un aspecto importante es cómo se ha llevado a cabo el cálculo de los tiempos de retardo para poder eliminar los posibles rebotes de entrada y salida, TIEMPO_ESPERA_INICIAL_US y TIEMPO_ESPERA_FINAL_US respectivamente. Estos valores no son constantes dado que dependen de la placa en la que se está ejecutando el programa. Por consiguiente ha sido necesario seguir un procedimiento para poder interpolar cuáles son los valores de esos tiempos. Este procedimiento se ejecuta en la función *botones_antirebotes_calibrar* localizado en el módulo botones_antirebotes,

El procedimiento emplea la pila de depuración colocando la cima en el primer elemento y comienza asignando tiempos por defecto a TIEMPO_ESPERA_INICIAL_US y a TIEMPO_ESPERA_FINAL_US. Dichos valores son siempre los mismos ya que son asignados a partir de una estructura enum. Seguidamente se procede a realizar tantas calibraciones como indica el valor NUM_CALIBRACIONES. El valor de NUM_CALIBRACIONES es 5 pero puede ser ajustado si se cree conveniente.



Para cada una de estas calibraciones se deja inicialmente un tiempo de retardo llamado TIEMPO_MARGEN_ANTIREBOTE_US. Si se pulsa el botón se guarda en una variable llamada *tiempo_espera_inicial_calibrado_us* el resultado de restar al instante de tiempo en el que se pulsa el botón el tiempo que se registró cuando fue soltado en la anterior pulsación. Si se suelta el botón se guarda en la variable *tiempo_espera_final_calibrado_us* el resultado de restar al instante de tiempo en el que se suelta el botón el tiempo de la anterior vez que fue pulsado. De esta forma la resta de los tiempos es el rebote.

Para poder obtener el tiempo del último evento insertado en la pila se usa la función pila_debug_ultimo_evento. Una vez efectuados los cálculos anteriores se procede a colocar la cima en la primera posición. Nótese que no se está usando la pila de una forma correcta ya que se está accediendo al segundo elemento de la pila en vez de a último y que además se está borrando, hecho que tampoco es del todo correcto. Sin embargo, se ha optado por hacerlo así porque es más fácil de gestionar los calibrados, y además, sólo se hace una única vez y al principio de la ejecución del programa, por lo que no supone ningún peligro para el funcionamiento correcto de la pila.

Una vez efectuadas las NUM_CALIBRACIONES calibraciones, se procede a efectuar una media de los tiempos de retardo de entrada y de salida dividiendo estos valores entre el total de calibraciones realizadas. Con las medias de calibración calculadas, se procede a compararlas para comprobar que efectivamente se ha calibrado bien y finalmente si los resultados son correctos se asignan los cálculos obtenidos a TIEMPO_ESPERA_INICIAL_US y TIEMPO_ESPERA_FINAL_US.

4.9 Implementación de la jugada con botones

Para jugar al juego se debe saber en qué fila y columna quiere colocar ficha el usuario. Anteriormente se había hecho mediante las variables fila, columna y ready. Ahora se han efectuado los cambios pertinentes para hacerlo mediante los botones (dispositivo botones_antirebotes) y el dispositivo 8leds para la visualización de los valores. Para ello se ha añadido al bucle principal la gestión de jugada_por_botones con una nueva máquina de estados que permite al usuario introducir los movimientos (fila, columna y ready). El autómata se controla desde la función *jugada_lista_botones*.

1. El autómata comienza en el estado GESTION_FILA, en dicho estado se pone el valor del display de 8 segementos con el valor 0xF de modo que el usuario puede ver que va a proceder a seleccionar la fila. Una vez el led muestra el valor correspondiente se pasa al estado SELECCION_FILA.

2. En el estado SELECCION_FILA, el usuario procede a elegir en qué fila quiere insertar su ficha usando el botón izquierdo, en este estado y en el de SELECCION_COLUMNA es donde se ha incorporado la opcionalidad del autoincremento. La razón por la que se ha decidido añadir el autoincremento es porque introducir los datos pulsación a pulsación resultaba tedioso.

   En este estado se mantiene el display con el valor 0xF hasta que el usuario presiona el botón izquierdo. Si después de pulsar el botón y de haber transcurrido el tiempo necesario para eliminar los rebotes, el usuario sigue pulsando ese mismo botón, por medio del autoincremento se modifica la coordenada de la fila módulo VALOR_FILA_COLUMNA_MAX, para impedir la inserción de fichas en filas situadas fuera del tablero por medio de la función *incrementar_valor_actual* . La modificación de la fila puede observarse en el valor que aparece en el display de 8 segmentos. Para confirmar la fila en cuestión el usuario debe pulsar el botón derecho, cambiando así al estado GESTIÓN_COLUMNA.

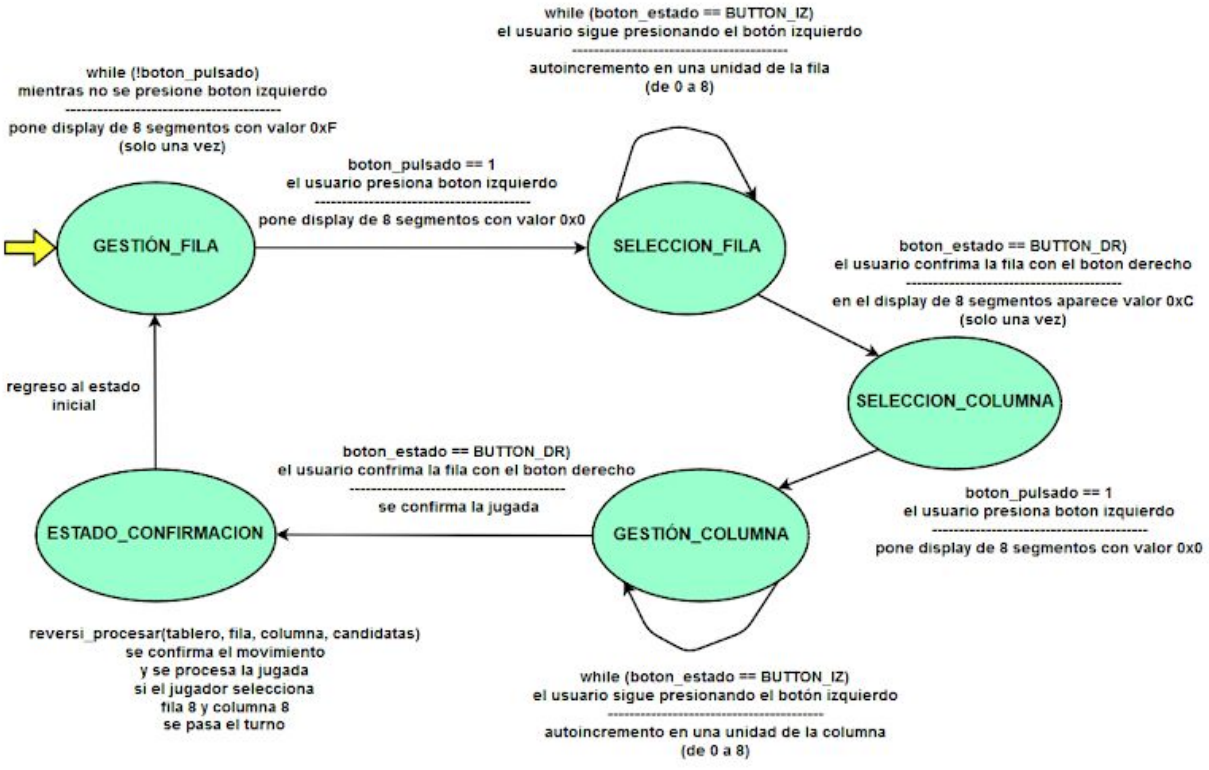
3. En este estado se realizan las mismas operaciones que en el estado GESTION_FILA pero en el display de 8 segmentos se muestra un 0xC para representar la columna. Seguidamente se pasa al estado SELECCION_COLUMNA. La selección de la columna también se gestiona con el botón izquierdo.

4. La secuencia de operaciones efectuadas en este estado son exactamente las mismas que en el estado SELECCION_COLUMNA, por lo que no se ha considerado necesario hacer más hincapié en este apartado.

Cuando se confirman la fila y la columna el movimiento se procesa en la lógica del juego, implementada en la función reversi_procesar.

Es importante tener en cuenta que la máquina de estados diseñada no está completa porque aún faltan algunos elementos que se han añadido en apartados posteriores. En vista a que el autómata debía ser modificado, se optó por hacer un diseño de la máquina claro y ordenado para poder trabajar con ella más adelante sin problemas.

A continuación se muestra una captura de la máquina de estados que representa la jugada por medio de los botones



4.10 Linker script

Llegados a este punto se ha procedido a estudiar la estructura del linker script y el código de la función init. Analizando su código en profundidad se ha detectado un problema que puede aparecer con alguna de las direcciones de los segmentos al declarar datos de tamaños inferiores al tamaño de palabra.

En el linker script aparecen tres regiones importantes, cada una de ellas comienza en una dirección de memoria específica. Dichas regiones son *Image_RO*, donde se almacenan las instrucciones del código y constantes, la región *Image_RW* donde se encuentran las variables globales no inicializadas a cero, y la región *Image_ZI,* donde se encuentran todas las variables globales a inicializar a cero.

De acuerdo a la configuración inicial del linker script, se encontró un fallo en la región de las variables inicializadas a cero, es decir, la región Image_ZI. Esta región no contiene la instrucción *ALIGN(4)*. La ausencia de esta instrucción significa que los datos no se almacenan en direcciones múltiplos de cuatro, es decir, no se almacenan en palabras.

El almacenamiento de los datos en direcciones que no tienen porque ser múltiplos de cuatro supone un problema dado que tras finalizar esta región se encuentran los espacios de memoria reservados para las pilas, ya que la inicialización de la región Image_ZI se realiza de palabra en palabra.

El problema es que como el comienzo de la región de las pilas y la región *Image_ZI* están consecutivas, puede ocurrir que una variable iniciada a cero se guarde en la intersección de ambas secciones, es decir, bytes en *Image_ZI* y en la pila. Por lo que se podrían borrar parte de los datos guardados en las pilas, desestructurandolas, y quedando desalineadas de cara al almacenamientos posteriores, pudiendo afectar con el mismo problema a las pilas posteriores. De este modo la inserción de la instrucción ALIGN(4) evita que se pisen parcialmente los datos, es decir, la posible sobreescritura reemplazaría las palabras enteras.

Con la instrucción *ALIGN(4)* se soluciona una parte del problema ya que el posible desbordamiento de los datos sigue vigente, puesto que si la cantidad de variables iniciadas a cero es considerable se introducirán en espacios de memoria pertenecientes a las pilas, pero este problema no surge dado que el espacio dedicado a la región Img_ZI de cara a las variables empleadas es más que suficiente.

4.11 Juego completo y pantalla LCD

En este apartado se ha llevado a cabo la configuración de la pantalla táctil LCD, de tal forma que ahora al reversi se puede jugar usando tanto la pantalla táctil como los botones. Para poder llevar a cabo la configuración del LCD ha sido necesario estudiar el material de documentación proporcionado por el profesorado referente al LCD touch. Una vez entendido, se ha procedido a desarrollar un conjunto de pantallas que ayudan al usuario a poder ver mejor el desarrollo del juego a nivel visual. Seguidamente se proporciona un autómata del juego en su versión final.

Al encender la placa, se muestra una pantalla donde se informa al usuario de que se va a proceder a calibrar la pantalla. Dicho estado se PANTALLA_CALIBRADO, donde se muestra una pantalla destinada a informar al usuario de que va a proceder a calibrar la

pantalla y los botones. Para llevar a cabo la calibración de la pantalla se realizan N_CALIBRACIONES de pantalla, el número de calibraciones está representado en un enum de esa forma si se desean realizar más el programador sólo tiene que cambiar el valor de N_CALIBRACIONES, cuyo valor por defecto es cuatro dado que se ha optado por realizar una calibración para cada esquina.

En cada una de las calibraciones se le muestra por pantalla al usuario en qué región de la pantalla debe pulsar. Concretamente debe pulsar en la esquina superior izquierda, en la esquina inferior izquierda, en la esquina inferior derecha y por último en la esquina superior derecha. Se ha optado por escoger este orden porque es muy fácil de seguir por el usuario dada su sencillez y mecanicidad, además el programa va informando de dónde tiene que pulsar, de ese modo se minimiza que el usuario pueda tener problemas. Una vez efectuada la calibración de la pantalla se pasa al estado BOTONES_CALIBRADO, donde se efectúa la calibración de los botones, es aquí donde se invoca al método que permite calcular los tiempos para eliminar los rebotes de entrada y salida, dependientes de la placa, cuyo nombre es *botones_antirebotes_calibrar,* y cuyo funcionamiento ya ha sido explicado anteriormente. Para que la calibración sea más amena, el programa muestra un mensaje informativo al usuario para avisarle de que los botones van a calibrarse, y adicionalmente, avisa al usuario de cuándo debe pulsar y soltar el botón. En dicho estado se gestiona toda la actividad del autómata de los botones antirebotes, véase apartado 4.8 del documento.

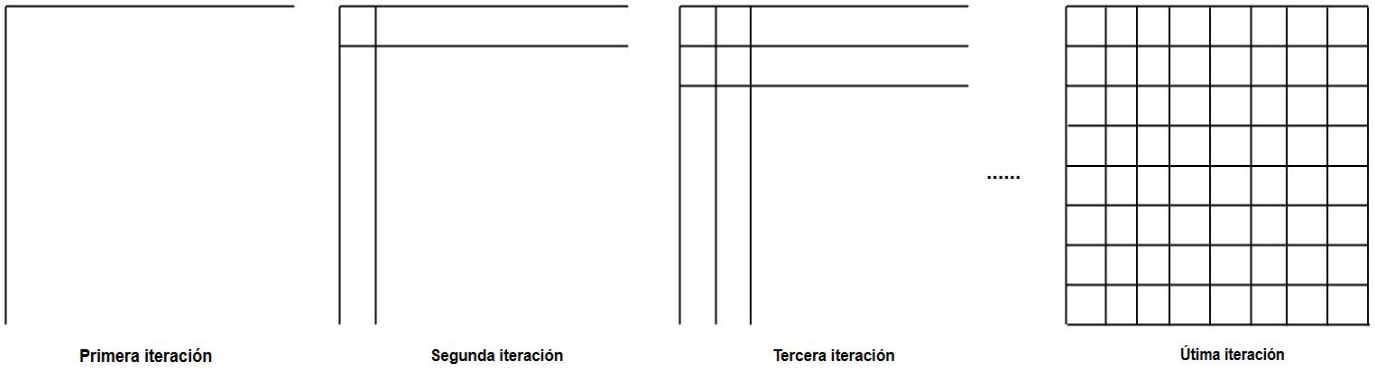
Una vez finalizadas las calibraciones, se muestra una pantalla introductoria del juego que cuenta con el título, un conjunto de instrucciones básicas sobre las reglas para jugar y una leyenda en la que se informa al usuario de que debe tocar la pantalla para poder jugar. Dicha pantalla ha sido elaborada por medio de la implementación de la función *dibujar_pantalla_inicial*, que es invocada desde la función *mostrar_pantalla_inicial*, localizada en el fichero reversi_2018.c. El juego comienza en el estado PANTALLA_INICIAL, donde se gestiona todos los elementos anteriores.

Para poder permanecer en la pantalla hasta que la pantalla es presionada se ha implementado la función *comprobar_pantalla_tactil_presionada*, que retorna si se ha presionado o no la pantalla. Dicha función es invocada también desde *mostrar_pantalla_inicial*.

En el momento en el que el usuario presiona la pantalla, se origina una interrupción por parte de ésta, y se invoca a la función coordenadas_pantalla_tactil_presionada. Una vez presionada la pantalla, se pasa al estado INICIO_PARTIDA donde se presenta el tablero inicial con las fichas blancas y negras en sus posiciones iniciales invocando a la función *dibujar_pantalla_juego*. En dicha función se gestiona el dibujado tanto del tablero como de las fichas iniciales, pero por separado, invocando a dos funciones diferentes. Para dibujar el tablero se llama a la función en la que se realiza un bucle desde cero hasta DIM_TABLERO, cuyo valor es 7, donde se escriben las coordenadas de las casillas tanto en vertical como en horizontal. Para construir las casillas, en cada iteración del bucle se dibuja una línea vertical y otra horizontal cuyas posiciones se calculan a partir de una posición de referencia desplazada por medio de un offset lo suficientemente grande como para que entre las líneas quepan las fichas. De esta forma las intersecciones resultantes entre las líneas

verticales y horizontales generan las casillas. La razón por la que se escogió este método es porque es

muy intuitivo, cómodo y fácil de implementar. A continuación se muestra como se ha procedido a dibujar el tablero del juego de un modo más gráfico.



Para dibujar las fichas cuatro fichas iniciales en el centro del tablero se ha usado la función *dibujar_ficha_virtual.* Todas estas funciones pueden encontrarse en el módulo pantalla. Una vez dibujado el tablero con las fichas se pasa al estado PARTIDA_EN_CURSO para gestionar los movimientos de los jugadores.

En dicho estado se ha procedido a dibujar una leyenda en la parte derecha de la pantalla donde se muestra el tiempo total transcurrido en la partida en segundos. También se mostrará información de “profiling”, en este caso se ha implementado para poder ver en µs el tiempo acumulado invertido en los cálculos del programa, el tiempo acumulado en la ejecución de la función *patron_volteo*, y el número de veces total que ha sido invocada. Se debe enfatizar que en la primera parte del proyecto se desarrollaron tres versiones de *patron_volteo*, concretamente fueron *patron_volteo*, *patron_volteo_arm_c* y *patron_volteo_arm_arm*, y para cada una de ellas se calculó el tiempo de ejecución para los diferentes flags de optimización del compilador. De acuerdo a los cálculos obtenidos la más eficiente fue *patron_volteo*, es decir, la original, ya que con el flag -03 su tiempo es de 6 µs. Por consiguiente los cálculos mostrados en la leyenda han sido calculados con esta versión.

La gestión del tiempo total que dura la partida se gestiona por medio de una función denominada *actualizar_tiempo_partida*, encargada de actualizar el tiempo por cada segundo transcurrido. Los tiempos de cálculos son gestionados en la función *actualizar_tiempo_calculos*, encargada de ir actualizando el tiempo que tarda en ejecutarse *patron_volteo*, y para contabilizar las invocaciones a *patrón_volteo* se emplea la función *actualizar_tiempo_invocaciones_patron_volteo.* Todas estas funciones pueden encontrarse en el módulo *gestion_profiling*. Las nuevas modificaciones, es decir, las fichas y los tiempos, se actualizan al refrescar la pantalla de juego del tablero con una tasa de 60 fps con las nuevas modificaciones. Se ha seleccionado esta frecuencia dado que es la estándar en el

desarrollo gráfico de videojuegos. Una vez refrescado el contexto de la partida se pasa al estado REALIZAR_JUGADA, es aquí donde el usuario hace el movimiento.

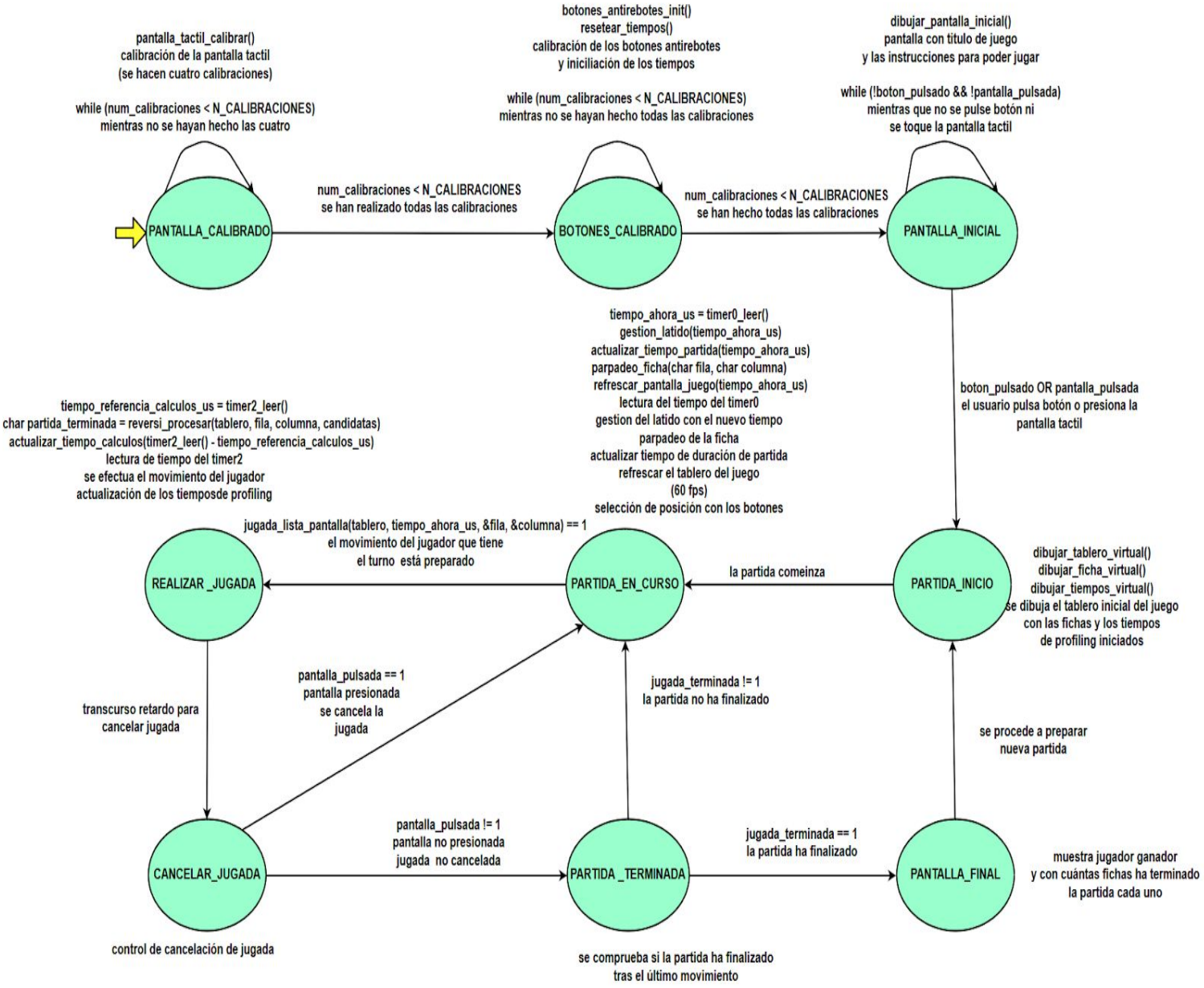
A la hora de seleccionar la jugada el usuario emplea los botones de la placa, tal y como se explicó en apartados anteriores. No obstante se ha añadido una funcionalidad adicional que consiste en que mientras el usuario se desplaza por las casillas del tablero para seleccionar el lugar donde quiere poner la ficha, ésta se encuentra parpadeando en gris y salte a las fichas ya colocadas. Para poder realizar dicha tarea ha habido que modificar el autómata de la jugada por botones para poder permitir esta nueva funcionalidad. Adicionalmente se ha modificado en el autómata la forma con la que se confirman los movimientos, y además se permite la opción de cancelar. En la primera versión del autómata se confirmaba presionando con el botón derecho, sin embargo, se ha cambiado para ahora confirmarse tocando la pantalla táctil. Cuando el usuario hace la jugada se pasa al estado CANCELAR_JUGADA.

Para poder programar la cancelación del movimiento se ha cambiado la estructura del autómata de manera que al confirmar el movimiento se produce un retardo de 2 segundos, de modo que si se presiona la pantalla antes de transcurrir ese tiempo se cancela el movimiento, en caso contrario, se confirma el movimiento en cuestión y se procede con la lógica del juego. Para pasar de turno, sólo basta con presionar la pantalla táctil, ya no es necesario validar fila 8 y columna 8, a diferencia de el autómata anterior. La cancelación del movimiento se gestiona en el estado CANCELAR_JUGADA. Si el usuario cancela, se pasa al estado PARTIDA_EN_CURSO para volver a introducir el movimiento. En caso contrario, se confirma el movimiento y se pasa al estado PARTIDA_TERMINADA para verficar si con ese nuevo movimiento la partida concluye o no. Si no concluye se retorna al estado PARTIDA_EN_CURSO para recibir nuevo movimiento, sino se pasa al estado PANTALLA_FINAL.

Para terminar, en el estado PANTALLA_FINAL, cuando la partida finaliza con la victoria de uno de los jugadores se ha decidido insertar una pantalla final que muestra que jugador es el ganador, y con cuántas fichas se ha quedado cada uno en el tablero dado que inicialmente estaba implementado sólo que para tras acabar se resetearan los tiempos del profiling, y se dibujara de nuevo el tablero con las fichas, lo cual era muy poco estético. A continuación se muestran una imagen del autómata con todo el desarrollo del juego.

4.12 Control del modo usuario

La ejecución de todo el programa hasta ahora se ha efectuado mayoritariamente con el procesador en modo supervisor. Se han realizado los cambios pertinentes para que el reversi se ejecute en modo usuario. Para ello se ha cambiado el modo del procesador a usuario antes de llamar a la función *reversi_main*. Ciertas funciones de E/S se ejecutan en modo IRQ o supervisor, allí donde sea necesario pero retornarán la ejecución a modo usuario tras la ejecución de la parte privilegiada.



La realización del cambio de modo se ha gestionado por medio de la implementación de la función *cambiar_a_modo_usuario* invocada desde main.c. Esta función obtiene la dirección

de memoria en la que empieza la pila de modo usuario, mueve el estado de ejecución del programa localizado en el registro cpsr a un registro de propósito general (r0), carga el contenido del registro cpsr en el registro spsr, así se salva el contenido del registro cpsr y se puede restaurar después si fuera necesario, se modifican los bits de control del procesador en el registro cpsr para cambiar a modo usuario, cuyo código en hexadecimal es 0x10, y finalmente se cambia la pila a la del modo usuario, ya que sino se estaría usando la pila del modo supervisor.

4.13 Plataforma automática

Para lograr que el programa se pueda ejecutar sin la necesidad de disponer PC se ha procedido a desarrollar una plataforma automática. Para ello se ha cargado el programa en la memoria Flash de la placa utilizando openOCD y JTAG. Para ello se han seguido los siguientes pasos:

1. En primer lugar se ha generado el binario a escribir en la memoria Flash a partir del fichero .elf. Esto se hace con la utilidad de gcc objcopy. Se ha realizado desde el intérprete de comandos ejecutando la orden *arm-none-eabi-objcopy -O binary prueba.elf fichero.bin*

2. A continuación, se ha volcado el binario a la Flash con el siguiente comando *openocd-0.7.0.exe -f test/arm-fdi-ucm.cfg -c "program prueba.bin 0x00000000".* Para ejecutarlo es necesario estar en la ruta del fichero del openocd.

Al encender la placa de nuevo, el código está almacenado en la Flash, pero allí no puede ser ejecutado correctamente porque no funcionan las escrituras y porque las direcciones reales no cuadran con las usadas por el linker. Así que se ha procedido a copiarlo a la RAM y a partir de ahí se puede jugar al reversi como antes. Para ello se ha añadido el código que inicializa el controlador de memoria, el código que copia el contenido de la ROM a la memoria RAM al comienzo de la ejecución, y, por último, el código que salte al Main recién copiado en la RAM. Esta nueva funcionalidad estará dentro de la rutina de tratamiento del reset. Dichas secciones aparecen documentadas en el fichero *44binit.asm*, localizado en el directorio *common* de los fuentes entregados.

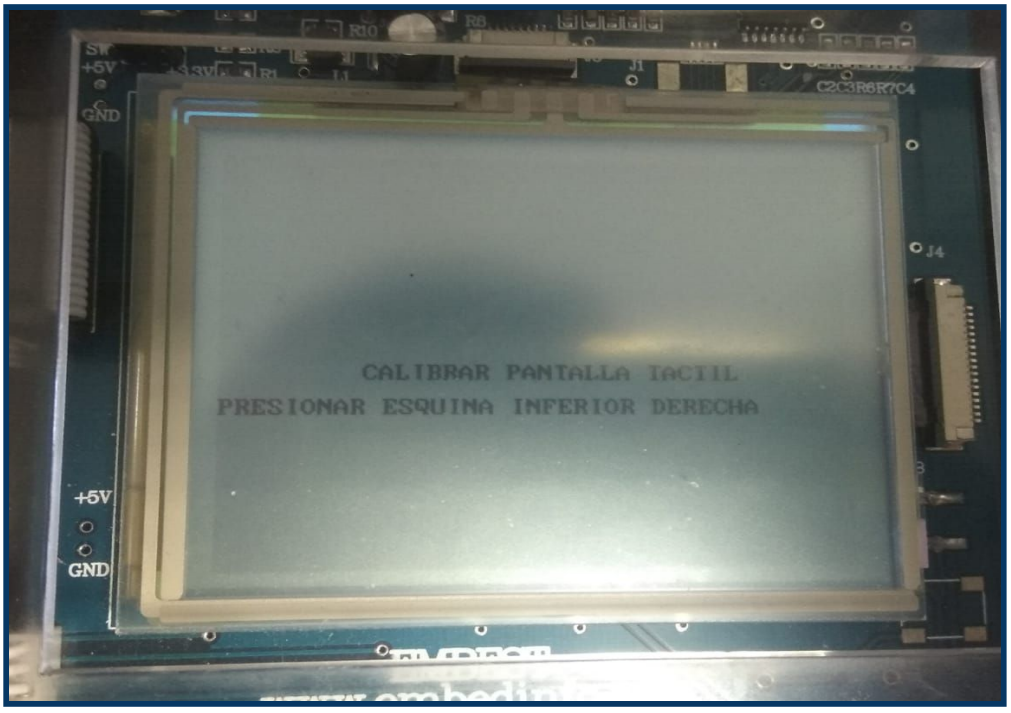
4.14 Capturas de la ejecución del juego

A continuación se muestran un conjunto de capturas del juego en ejecución una vez cargado en la flash de la placa. Dichas capturas fueron tomadas con el objetivo de mostrar el aspecto que tiene la interfaz gráfica que se la ha dado al juego.

Esta captura sirve para informar al usuario en qué regiones de la pantalla debe pulsar para poder realizar la calibración. De las cuatro calibraciones sólo se han incluído dos ya que solo cambian las posiciones.

A continuación se muestra una imagen del proceso de las calibraciones de pantalla, concretamente esta es la calibración de la esquina superior izquierda de la pantalla.

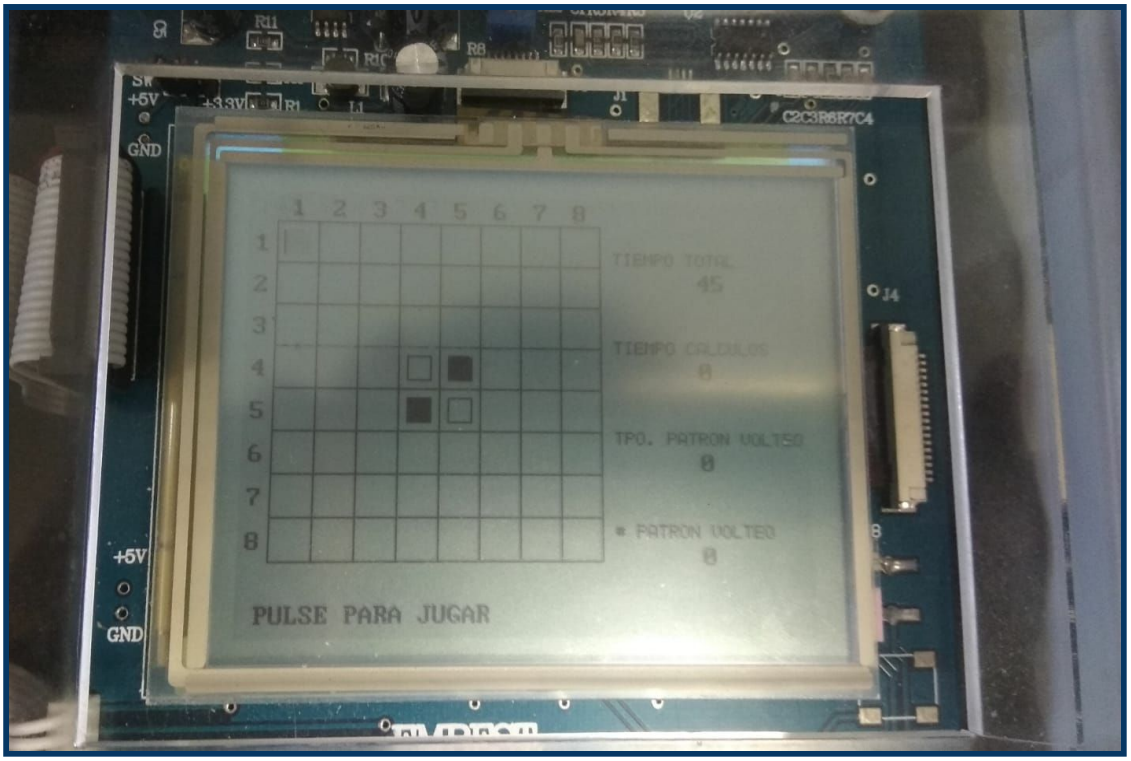
Esta captura muestra la calibración de la esquina inferior derecha de la pantalla.



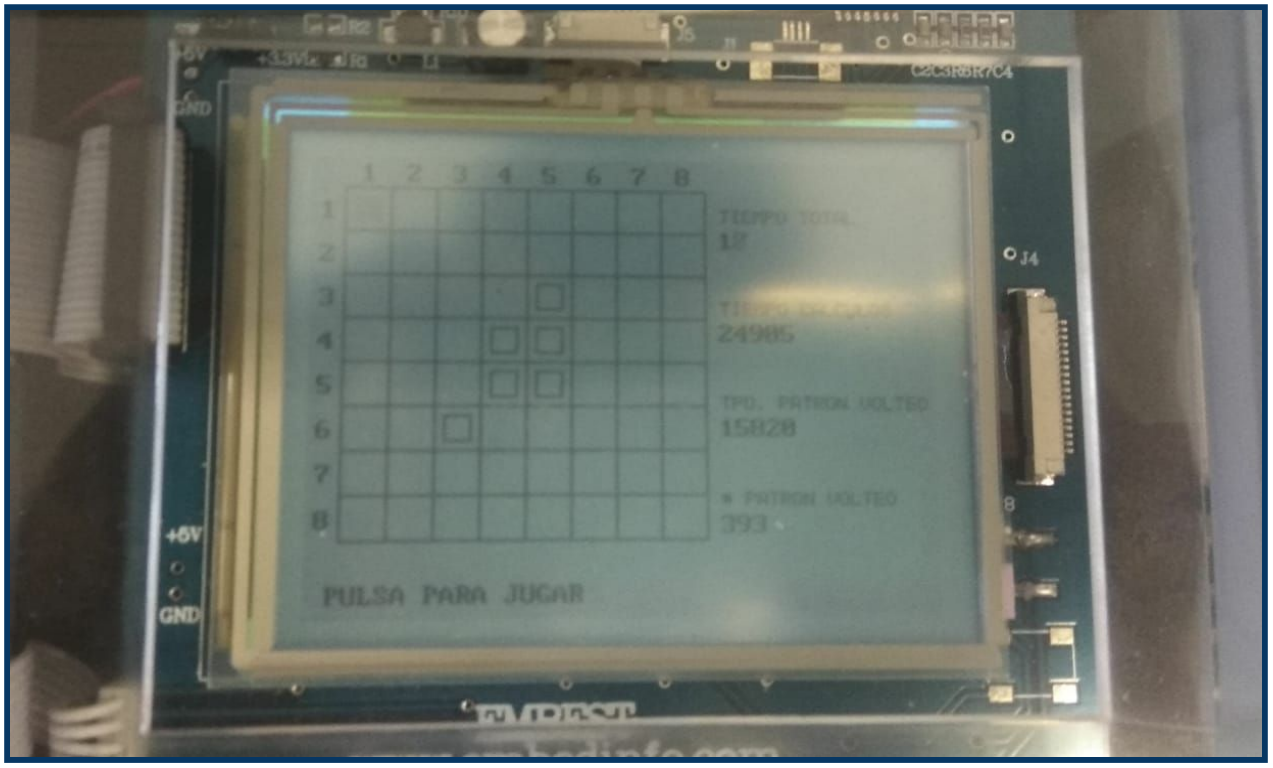
Seguidamente se muestra una captura en la que se le informa al usuario que se va a proceder a realizar la calibración de los botones de la placa, para eliminar los rebotes.

La captura siguiente muestra el momento de la calibración del botón, concretamente cuando le pide soltar el botón que ha pulsado.

La siguiente captura muestra la apariencia del juego en el momento en el que se inicia la partida con el tablero, las fichas en las correspondientes posiciones y los tiempos de profiling a la derecha



La siguiente captura muestra el tablero final una vez que ha concluído la partida, en dicha imagen se puede observar que el jugador ganador es el blanco.



Finalmente se ha proporcionado una captura final del juego con el resultado del ganador y con cuántas fichas ha quedado cada uno en el tablero.

4.15 Pruebas de funcionamiento

Llegados a este punto se procedió con la parte de realización de las pruebas del código para ver que el programa en cuestión funcionaba correctamente y que cumplía todos los requisitos especificados.

En primer lugar se han desarrollado dos pequeñas funciones para poner a prueba el tratamiento de la pila de depuración y la captura de las excepciones. La pruebas de la pila de depuración se ha realizado en una función llamada *pila_debug_test*, donde se realizan una serie de llamadas a la función *push_debug* pasando parámetros aleatorios ya que el único objetivo de este test era comprobar que la inserción de los datos se hacía correctamente, que la pila tenía espacio más que suficiente para almacenar los datos, y que el control del desbordamiento era gestionado correctamente.

Para probar las excepciones se ha implementado una función llamada *exceptions_test* que se encarga de forzar excepciones de tipo SWI, Undefined y Data Abort. Para poder forzar la interrupción *SWI* se ha empleado la instrucción *__asm__ volatile("swi 1")*. El argumento 1 es ignorado por el procesador, aunque el controlador de excepciones lo puede emplear para determinar qué servicio se solicita.

Para forzar la excepción *Undefined* se ha usado la instrucción *__asm__ volatile(".word 0xe7f000f0\n")*. Esta instrucción genera una excepción de tipo undefined porque de acuerdo con el manual de la arquitectura de arm es un tipo de instrucción con un formato indefinido.

Por último para generar la excepción de tipo *Data Abort* se han usado las siguientes tres instrucciones:

```
intptr_t invalid_ptr = 0xFFFFFFFF;
int invalid_var = *(int *) invalid_ptr;
return invalid_var;
```

Para poder comprender por qué estas tres instrucciones generan la excepción Data Abort se va a explicar desde la perspectiva del lenguaje ensamblador la secuencia de pasos que se realizan. Al ejecutar la primera instrucción se están reservando 4 bytes de memoria para la variable *invalid_ptr*, que es de tipo *intptr_t*, es decir, un puntero a un entero con signo. A dicho puntero se le asigna la constante *0xFFFFFFFF,* que en términos de ensamblador es una instrucción *mov registro #constante.* Posteriormente se reservan otros 4 bytes de memoria para la variable *invalid_var* de tipo *int,* y a dicha variable se le asigna el valor de lo que haya en el puntero, pero en esa dirección no hay nada, saltando así la excepción Data Abort.

Una vez probadas las excepciones y la pila de depuración se continuó elaborando pruebas para poder verificar que la lógica del reversi iba adecuadamente con las máquinas de estados y los rebotes. Para ello se procedieron a jugar los siguientes casos:

1. Al comenzar la partida, después de haber iniciado el tablero con las fichas blancas y negras, se procedió a que el jugador negro, es decir, la persona, pasará turno continuamente, introduciendo fila 8 y columna 8, hasta que el blanco, la CPU, volteara todas las fichas negras.

   Esta prueba ha servido para determinar que el jugador pasa turno correctamente, que las fichas son insertadas en las posiciones correctas, que la partida termina correctamente y que después de haber concluído la partida comienza una nueva.

2. Se ha procedido a jugar una partida completa para determinar que el juego funciona de manera adecuada. Mediante esta prueba se ha logrado afianzar las conclusiones a las que se llegó en la prueba anterior. Esta prueba ha servido para comprobar que no se experimentan fallos a lo largo del transcurso de la partida.

   A continuación se describen a razón de uno por línea los turnos con las jugadas por parte de ambos bandos, blancas y negras, N indica que ha sido un movimiento de las negras y B para indicar que el movimiento lo han realizado las blancas. La partida concluyó con la victoria del jugador blanco

N fila 4, columna 2	B fila 3 ,columna 2
N fila 2, columna 3	B fila 4, columna 2
N fila 2, columna 1	B fila 2, columna 4
N fila 5, columna 4	B fila 2, columna 0
N fila 1, columna 4	B fila 2, columna 5
N fila 4, columna 1	B fila 5, columna 2
N fila 1, columna 0	B fila 0, columna 0
N fila 2, columna 6	B fila 3, columna 7
N fila 6, columna 2	B fila 7, columna 2
N fila 1, columna 3	B fila 0, columna 2
N fila 3, columna 5	B fila 5, columna 2
N fila 3, columna 1	B fila 0, columna 3
N fila 5, columna 1	B fila 3, columna 0
N fila 7, columna 4	B fila 7, columna 5
N fila 4, columna 5	B fila 4, columna 0
N fila 1, columna 1	B fila 0, columna 1
N fila 1, columna 6	B fila 5, columna 5
N fila 1, columna 5	B fila 0, columna 7
N fila 4, columna 6	B fila 0, columna 6
N fila 1, columna 2	B fila 3, columna 7
N fila 5, columna 6	B fila 6, columna 7

5 Problemas encontrados

Durante la elaboración de este proyecto se han tenido que solventar problemas de distinta índole. El principal problema de la realización de este proyecto ha sido la gestión de los rebotes de los botones, sobre todo qué procedimiento ingenieril emplear para poder solucionarlo ya que inicialmente los tiempos de retardo para los rebotes de entrada y salida se calculaban mediante ensayo-error cada vez que se cambiaba de placa, lo cual no era para nada eficiente.

Otro problema a destacar ha sido la gestión de las excepciones, particularmente en el tratamiento del vector de interrupciones, puesto que hubo que entender cómo había que capturarlas y cómo poder forzarlas manualmente, haciendo especial énfasis en el Data Abort debido a que había diversas formas de provocarla.

También hubo problemas a la hora de concretar algunos aspectos de diseño de la interfaz gráfica como por ejemplo cómo dibujar el tablero dado que se disponían de diferentes planteamientos para dibujarlo, tanto recursivos como iterativos, pero costó decidir cuál de todos ellos era el más sencillo y eficiente. También hubo problemas a la hora de seleccionar la apariencia de las fichas ya que inicialmente se había optado por hacerlas redondas en vez de cuadradas por medio de un bitmap. Sin embargo, por falta de tiempo se optó por hacerlas cuadradas aprovechando así parte del código suministrado en la documentación.

6 Correcciones

Se reestructuró el código del proyecto para que resultase más fácil de entender el funcionamiento del mismo. No hubo más correcciones.

7. Conclusiones

A lo largo de estas dos prácticas, y en definitiva de todo el cuatrimestre, se ha adquirido un alto conocimiento de la importancia de esta asignatura y del papel que desempeña en la formación de cualquier ingeniero informático. Es importante enfatizar la inmensa cantidad de conocimientos que se han adquirido y su elevada utilidad de cara a un futuro laboral. Los miembros del equipo están muy satisfechos tanto con la labor del profesorado en la coordinación de la asignatura y en el trato con el alumnado como con el trabajo personal realizado, ya que cumple todos los requisitos obligatorios y, además algunos apartados opcionales.

8. Bibliografía

Como documentación de ayuda se ha empleado todo el material de apoyo proporcionado por el profesorado de la asignatura. Dicho material está compuesta por el manual del lenguaje ensamblador ARM, el estándar ATPCS, la documentación referente al chip S3C44B0X y la placa S3CEV40, los archivos de la universidad complutense, el manual de arquitectura ARM y el tratamiento de excepciones. Toda esta información se puede encontrar en la sección de material de prácticas de la asignatura Proyecto Hardware en Moodle.

9. Anexos

Dada la gran cantidad de ficheros de código fuente que componen el proyecto, han sido presentados aparte bien comentados para facilitar su comprensión.