Resumen

Introducción

Practica 2

Objetivos

Metodología

Ficheros

Implementación

Excepciones

Medidas de tiempo

Pila depuración

Eliminación de rebotes

Juego

Resultados

Mediciones tiempo

Comparación

Problemas encontrados

. . .

Practica 3

Objetivos

Metodología

Comentarios

Calibracion

Pantalla inicio

Tablero

Resultados

Conclusión

Anexo

Código practica2

Código practica3

**Ficheros**

\*\*\*\* poner captura ?

**Implementación**

**Excepciones**

En el fichero excepcion.c se implementa la rutina de servicio *exception\_handler*. Las interrupciones que captura son:

- DABORT: Acceso no alineado a memoria

- SWI: Instrucción Software Interrup de ARM

- UNDEF: Instrucción indefinida

Cuando se declara la función en el fichero se indica que tratara las interrupciones anteriores mediante los atributos siguientes:

void exception\_handler(void) \_\_attribute\_\_((interrupt("DABORT")));

void exception\_handler(void) \_\_attribute\_\_((interrupt("SWI")));

void exception\_handler(void) \_\_attribute\_\_((interrupt("UNDEF")));

Para que dicha función sea la que se ocupe del tratamiento de las interrupciones es necesario indicarlo en la tabla de interrupciones ISR. Para ello en la dirección de la ISR de cada interrupción se asigna la dirección de la función de tratamiento *exception\_handler:*

pISR\_DABORT = (unsigned) exception\_handler;

pISR\_SWI = (unsigned) exception\_handler;

pISR\_UNDEF = (unsigned) exception\_handler;

siendo pISR\_DABORT, pISR\_SWI y pISR\_UNDEF direcciones de la tabla.

La asignación anterior se incluye dentro de la función init\_exception que es invocada solo para inicializar la función de tratamiento. Dentro de esta función también esta la llamada *D8Led\_init* que inicializa el periférico 8-led al que accede la rutina.

En cuanto a la implementación de la rutina *exception\_handler*, en primer lugar llama a la función *CPSR\_read* definida en 44init.asm. Esta función lee el registro CPSR (mediante mrs) y devuelve (en decimal) los 5 bits de menor indice (5 primeros). Con el resultado de esta llamada, la rutina detecta la interrupción causante, pudiendo ser 23 (DABORT), 27(UNDEF) o 31(SWI en este caso).

En función de la interrupción causante, mediante la llamada al periférico *D8Led\_symbol,* se muestran los dígitos D, E o F. Finalmente la ejecución termina con la llamada a un bucle recursivo.

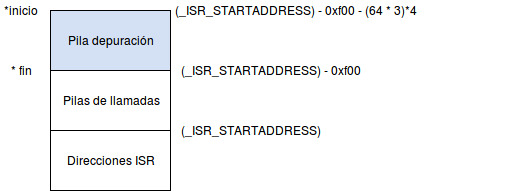
Para la validación del funcionamiento de la rutina se han implementado las siguientes funciones:

- *genera\_dabort*: ejecuta una instrucción ldr a una dirección de memoria impar

- *genera\_undef*: mediante la llamada *asm()* ejecuta una instrucción no definida.

- *genera\_swi*: ejecuta una instrucción swi.

**Pila de depuración**



La pila de depuración se ha implementado como mediante una dirección de inicio, una dirección de final y un puntero que actúa como iterador. La llama a *push\_debug* recibe 2 parámetros (enteros) que almacenara en la pila (ID\_evento y auxData). Ademas de esto la pila almacenara el valor que devuelve la función *timer2\_leer().* Se almacenaran 3 enteros por lo que cada llamada a *push\_debug* almacena en la pila 3 enteros \* 4 palabras/entero por lo que el tamaño de la pila deberá ser múltiplo de dicho valor (para que la pila quede alineada). Se ha decidido dar a la pila tamaño suficiente para 64 llamadas por lo que el tamaño de la pila es 64\*3\*4.

En cuanto a la dirección de la pila, la pila de llamadas comienza en la dirección *ISR\_STARTADRESS – 0xf00*. Dicha dirección marca el final de la pila de depuración (sin llegar a escribir en esa dirección). El comienzo sera por lo tanto la sustracción del tamaño de la pila a dicha dirección.

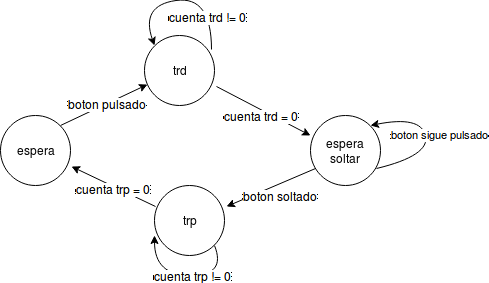
La función *debug\_init* se encarga de situar el iterador en la dirección de inicio:

pos = (unsigned int\*)inicio

La función debug\_push compara que el iterador no haya llegado a la dirección final de la pila y en ese caso sitúa el iterador de nuevo al comienzo de esta. Escribe los 3 enteros seguidos en memoria a partir de la posición del iterador y aumenta la dirección del iterador en 3 unidades (porque ha escrito 3 enteros).

**Eliminacion de rebotes**

El diseño de la maquina de estados para la eliminacion de rebotes del boton es la siguiente:



Estados e implementacion:

El estado de la maquina se almacena mediante una variable compartida (definida en button.h) de tipo enum con los diferentes estados:

enum estados\_boton{

espera = 0,

trp = 1,

espera\_soltar = 2,

trd = 3,

aumenta = 4

} volatile estado;

- espera: El estado inicial de la maquina comienza en la rutina de interrupcion del boton *Eint4567\_ISR.* En dicha rutina se comprueba la condicion de salida “boton pulsado”. Para la deteccion de la pulsacion de los botones 6 o 7 se comprueba el el bit 6 o 7 del registro Rpdtag esta a 0 (0 = pulsado, 1 = no pulsado). En caso de estar pulsado, se almacena cual de los botones es el pulsado y se avanza el estado al estado trd.

- trd:

[FALTA EXPLICACION CONFIGURACION TIMER 0]

[CUANTO SON 25 INTERRUPCIONES CON EL TIMER 0]

- estado soltar: para controlar que se compruebe si el boton a sido soltado cada cierto intervalo de tiempo se define la variable *cuenta\_medio.* Como hay que controlar cada cierto intervalo de tiempo si el boton esta soltado, esta condicion se controla desde la rutina de interrupcion del timer0. En esta se comprueba si la variable de cuenta a llegado a 0. Si es el caso se evalua si el boton a sido soltado (leyendo el bit 6 o 7 de Rpdtag según el boton almacenado anteriormente). En caso de no haber sido soltado, se reinicia la cuenta y se repite el procedimiento. Cuando finalmente se suelta el boton se avanza al siguiente estado y se inizializa la cuenta de la fase trp.

- trp

**Incorporacion de botones en el juego**

[CAMBIAR DETECCION BOTONES]

**Practica3**

**Objetivos**

Se va a incorporar a la practica anterior la posibilidad de jugar a traves del periferico touchscreen. Primero se muestra una pantalla de calibracion para el touchpad. Se pedira al usuario que pulsa distintas posiciones hasta obtener unos valores de calibracion coherentes. Seguidamente se mostrara una pantalla de inicio y se esperara a que el usuario pulse la pantalla o el boton. Se mostrara el tablero con las casillas. El usuario pulsara un cuadrante y se mostrara un zoom a esa zona. El usuario seleccionara una casilla o volvera atrás. Una vez seleccionada podra cancelar su eleccion durante unos segundos. Esta eleccion tambien se podra realizar con los botones. El usuario tambien dispone de la opcion de pasar de turno o finalizar la partida. Finalmente se muestra el resultado de la partido. Durante el juego se muestra el tiempo total y el tiempo de calculo.

Ademas de lo anterior, se haran las interrupciones del timer FIQ para darles mayor prioridad y se editaran los fucheros para volcar el juego en la memoria flash de la placa.

**Ficheros**

**Desarrollo**

Todas las implementaciones de Lcd y Touchscreen se incluyen en el codigo del juego en el fichero reversi8.c A continuacion se explicaran las diferentes vistas de pantallas y su funcionalidad:

**Comentarios**

En varios casos es necesario mostrar por pantalla cadenas de texto o numeros. Para esta finaliadad de utiliza la funcion de la librería del *LCD Lcd\_DspAscII8x16.*

En la pantalla del tablero los diferentes elementos como el tablero, las fichas y los tiempos se implementan en funciones separadas. Cada una de estas funciones por separado carga en el buffer las instrucciones y finalmente en la funcion *esperar\_mov* se muestra todo lo cargado en el buffer con *Lcd\_Dma\_Trans().*

**Calibracion**

**Pantalla de inicio**

La pantalla de inicio muestra el mensaje “Toque la pantalla para jugar”.