Resumen

Introducción

Practica 2

Objetivos

Metodología

Ficheros

Implementación

Excepciones

Medidas de tiempo

Pila depuración

Eliminación de rebotes

Juego

Resultados

Interrupciones

Mediciones tiempo

Comparación

. . .

Practica 3

Objetivos

Metodología

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.

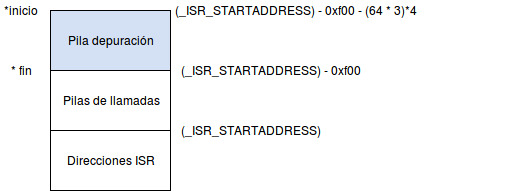
Para la validación del funcionamiento de la rutina se han implementado las siguientes funciones:

- *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).