Práctica 1: Desarrollo de código para el procesador ARM

Profesores: Javier Resano, Dario Suarez, Enrique Torres, María Villarroya

enrique.torres@unizar.es

Agradecimientos: L. Piñuel, J.I. Gómez, C. Tenllado...

Dep. Arquitectura de Computadores y Automática

Universidad Complutense de Madrid

Objetivos

- Optimizar el rendimiento de un juego acelerando las funciones computacionalmente más costosas.
  - entender funcionalidad del código suministrado
  - realizar versión en ensamblador ARM optimizado
  - medir rendimiento respecto al código optimizado generado por el compilador
  - documentar los resultados.
- Gestionar el tiempo de trabajo del proyecto correctamente en función de la disponibilidad de acceso al sistema específico.

Objetivos

- Conocer la estructura segmentada del procesador ARM 7
- Interactuar con una placa real y ser capaces de ejecutar en ella
  - Gestionar la entrada/salida, desarrollar en C las rutinas de tratamiento de interrupción
- Familiarizarse con la generación cruzada de código ARM y su depuración
  - siguiendo el contenido de los registros internos del procesador y de la memoria
- Desarrollar código en ensamblador y combinarlo con código en C
  - **ARM**: adecuado para optimizar el rendimiento
  - Entender la finalidad y el funcionamiento de las ***Aplication Binary Interface***, ABI (**AATPCS**)
- Optimizar código (a nivel de ensamblador) y Optimizaciones del compilador
  - Medir rendimiento

Bibliografía y Material disponible 1/2

- Básica:
  - David Seal; **ARM Architecture Reference Manual**; 2ª Ed, Addison-Wesley, 2001 (versión antigua disponible en la web de la asignatura)
  - **Cuaderno de prácticas del entorno Embest IDE** (disponible en la web de la asignatura)
- Complementaria:
  - A. Sloss, D. Symes, C. Wright; **ARM system Developer's Guide**; Morgan Kaufman, 2004
  - Steve Furber; **ARM System-on-chip Architecture**; 2ª Ed, Addison-Wesley, 2000
  - William Hohl; **ARM Assembly Language. Fundamentals and Tecniques**; CRC Press, 2009
- Enlaces
  - www.arm.com/documentation
  - www.arm.com/gnu

Bibliografía y Material disponible 2/2

- Cuadernos de prácticas de una asignatura de la Universidad Complutense que usa el mismo entorno de trabajo y las mismas placas
  - **EntradaSalida.pdf**: describe los principales elementos de entrada / salida
  - **P2-ec.pdf**: describe la gestión de excepciones y el mapa de memoria que vamos a usar
- Breve resumen del repertorio de instrucciones ARM
- Documentación original del micro y de la placa
- Códigos fuente iniciales y *Linker script*
- Un proyecto que utiliza el timer0 de la placa:
  - debéis seguir el mismo esquema que este proyecto
- Directrices sobre cómo redactar la memoria

Entorno de trabajo

- Utilizaremos como entorno de trabajo Eclipse con una cadena de herramientas (*toolchain*) para compilación cruzada:
  - Herramientas de GCC para ARM para compilar y depurar
  - Placa de desarrollo **Embest S3CEV40** + **JTAG**
  - El plugin *Zylin Embedded debug* para simular el procesador (sin placa, no E/S)

Estructura de la práctica

- Parte A (1 semana):
  - Objetivos:
    - Familiarizarse con el entorno y los repertorios de instrucciones
    - Crear proyectos, compilar y enlazar
    - Depurar sobre la placa y sobre el simulador.
    - Ejecutar paso a paso en alto nivel y en ensamblador el código suministrado
      - Observar registros y memoria
      - Breakpoints
      - …
  - Preparar de cara a la siguiente sesión:
    - Reprogramar en ensamblador **ARM optimizado** la función más costosa en computo**.**
    - Rutina Servicio Interrupción del Timer2

Estructura de la práctica

- Parte B (2 semanas):
  - Utilizar un temporizador interno (***timer***) para medir tiempos
    - Programar los registros de E/S del timer y la rutina de servicio de interrupción en C.
  - Reprogramar en ensamblador **ARM optimizado** la función más costosa en computo**.**
  - Observar código generado por compilador optimizador en -O0, -O1, -O2, …
  - Comparar las opciones (original, ARM y C optimizado) teniendo en cuenta:
    - Tamaño del código
    - Nº de instrucciones ejecutadas
    - Tiempo de ejecución.
  - Debéis optimizar lo máximo posible los códigos en ensamblador, pero respetando la misma estructura que el código C y la modularidad.
  - **Verificar automáticamente el funcionamiento** (equivalencia funcional)

Evaluación de la práctica

- Fechas estimadas:
  - Entrega de esta parte: **21** de Octubre
  - Entrega de la memoria: **27** de Octubre
  - Las fechas definitivas se publicarán en la página web de la asignatura (moodle)

Realización de la memoria

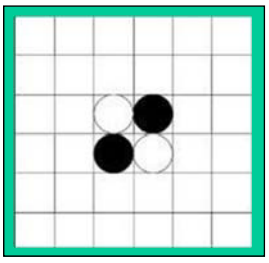
- Resumen ejecutivo (**una cara máximo**).
- Descripción de las optimizaciones realizadas al código ensamblador
- Resultados de la comparación entre distintas versiones de la función
- Descripción de los problemas encontrados en la realización de la práctica y sus soluciones
- Firma y autoría.
- Código fuente del apartado B comentado. Además, la **función ARM** debe incluir una **cabecera** en la que se explique cómo funciona, qué **parámetros** recibe, dónde los recibe y qué **uso** se da a cada **registro** (ej. en el registro 4 se guarda el puntero a la primera matriz)

Esquema

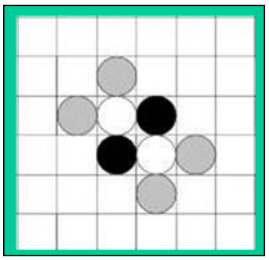
- **Descripción de la práctica**
- Sistema de memoria de la placa S3CEV40
- Sistema de E/S de la placa S3CEV40
- Fuentes y ayudas de C

¿Sabéis jugar al reversi?

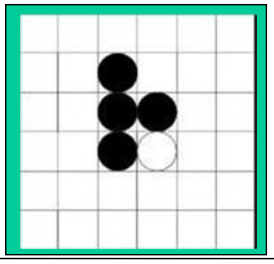
El reversi se juega en un tablero de 8x8



Los jugadores se alternan colocando fichas empezando por las negras. Se puede colocar sólo en las posiciones en las que se rodea al rival en alguna dirección

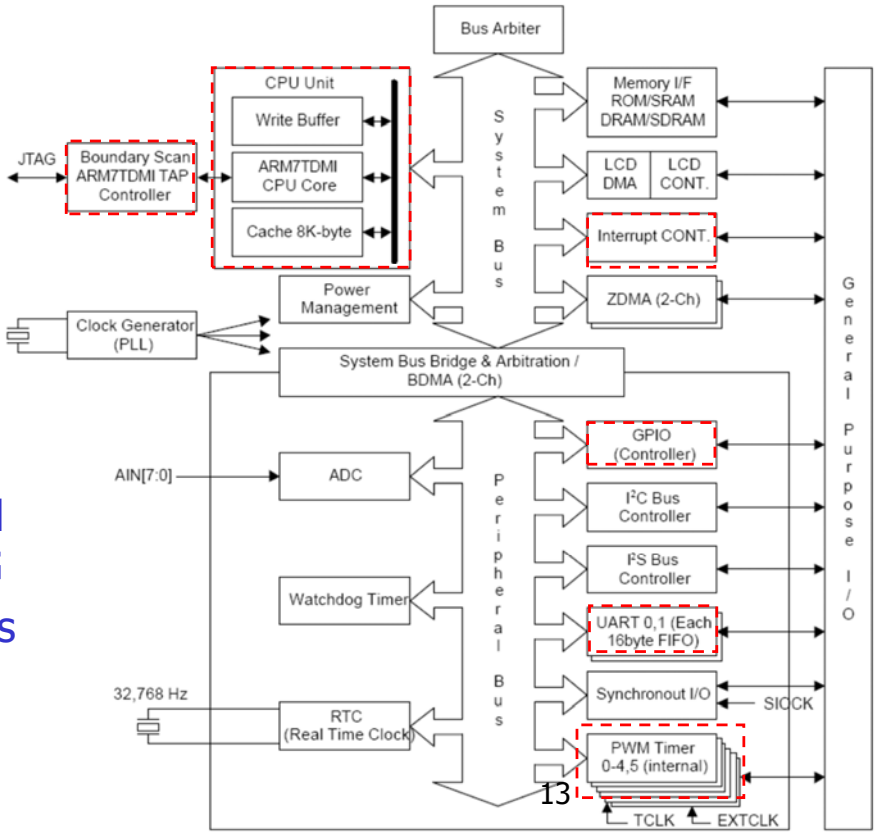


Las fichas rodeadas cambian de color. Cuando no haya movimientos posibles gana el jugador con más fichas de su color



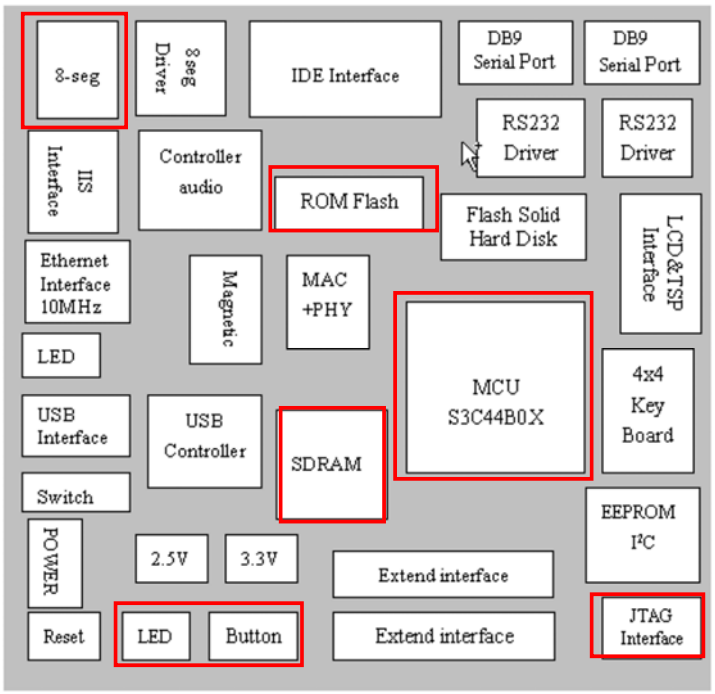
Chip principal de la placa: S3C44B0X

- System-on-Chip (Soc):
  - Procesador ARM7TDMI
  - Controladores E/S
    - Temporizadores
    - Controlador de interrupciones
  - Buses
  - Comunicación directa con el PC a través del puerto JTAG
  - Y muchos más componentes



Elementos en la placa Embest S3CEV40

- ¿Qué vamos a usar?
  - Chip S3C44B0X
  - Memoria
  - Leds
  - Pulsadores
  - 8-segmentos
  - JTAG
  - LCD



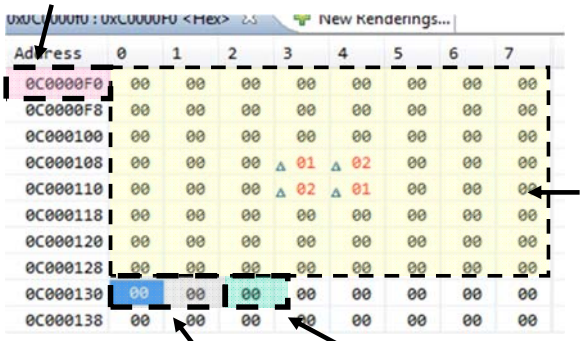
¿Qué tenemos que aprender?

- Cómo configurar la placa para poder utilizarla:
  - Utilización de un script de configuración para inicializar el espacio de memoria [**3**]
  - Registros de configuración de los elementos utilizados
- Gestión del hardware del sistema utilizando C:
  - Utilización de las bibliotecas de la placa
  - Gestión de las interrupciones en C
  - Entender las estructuras que genera el compilador a partir del código fuente (especialmente la pila de programa) [**2**]

¿Cómo se juega en el simulador?

El tablero está almacenado en memoria (hay que adaptar el tamaño de la ventana para que se visualice bien)
Los movimientos se indican también escribiendo en tres posiciones determinadas de memoria

Posición de comienzo del tablero

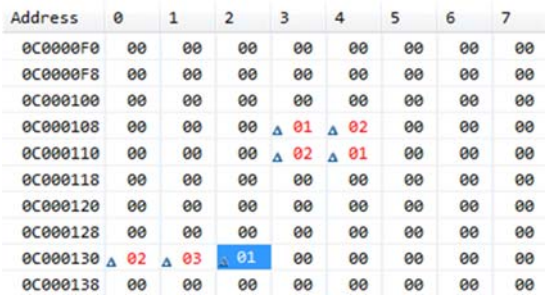


Tablero 01: ficha blanca
02: ficha negra
00: posición vacía

fila-columna

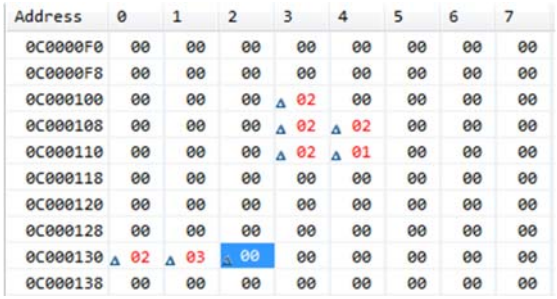
ready

Ejemplo de movimiento



Address	0	1	2	3	4	5	6	7
0C0000F0	00	00	00	00	00	00	00	00
0C0000F8	00	00	00	00	00	00	00	00
0C000100	00	00	00	00	00	00	00	00
0C000108	00	00	00	01	02	00	00	00
0C000110	00	00	00	02	01	00	00	00
0C000118	00	00	00	00	00	00	00	00
0C000120	00	00	00	00	00	00	00	00
0C000128	00	00	00	00	00	00	00	00
0C000130	02	03	01	00	00	00	00	00
0C000138	00	00	00	00	00	00	00	00

Orden de colocar ficha negra en la fila 2 columna 3



Address	0	1	2	3	4	5	6	7
0C0000F0	00	00	00	00	00	00	00	00
0C0000F8	00	00	00	00	00	00	00	00
0C000100	00	00	00	02	00	00	00	00
0C000108	00	00	00	02	02	00	00	00
0C000110	00	00	00	02	01	00	00	00
0C000118	00	00	00	00	00	00	00	00
0C000120	00	00	00	00	00	00	00	00
0C000128	00	00	00	00	00	00	00	00
0C000130	02	03	00	00	00	00	00	00
0C000138	00	00	00	00	00	00	00	00

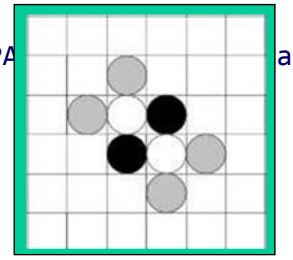
El tablero se actualiza con el movimiento

reversi8_2019.c

```
// Tamaño del tablero
#define DIM   8

…

// Estados de las casillas del tablero
const char CASILLA_VACIA = 0; //deberia ser enum
const char FICHA_BLANCA  = 1;
const char FICHA_NEGRA   = 2;

char __attribute__ ((aligned (8))) tablero[DIM][DIM] = {
    {CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASIL
    {CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASIL
    …
    {CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASILLA_VACIA,CASIL
 };

////////////////////////////////////////////////////////////////////
// VARIABLES PARA INTERACCIONAR CON LA ENTRADA SALIDA
// Pregunta: ¿hay que hacer algo con ellas para que esto funcione bien?
// (por ejemplo añadir alguna palabra clave para garantizar que la sincronización a través de

char fila=0, columna=0, ready = 0;
…
```

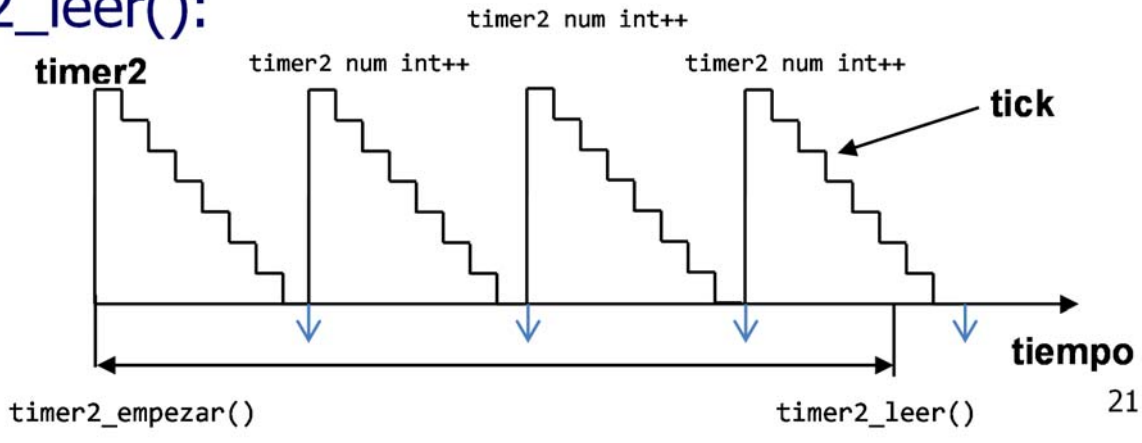
reversi8_2019.c

```
char ficha_valida(char tablero[][DIM], char f, char c, int *posicion_valida)
{
    …
    return ficha;
}

int patron_volteo(char tablero[][DIM],int *longitud,char FA,char CA,char SF,char SC,char color)
{
    …
    casilla = ficha_valida(tablero, FA, CA, &posicion_valida);
    …
    while ((posicion_valida == 1) && (casilla != color))
        *longitud = *longitud + 1;
        casilla = ficha_valida(FA, CA, tablero, &posicion_valida);
    if ((posicion_valida == 1) && (casilla == color))
        return PATRON_ENCONTRADO; // si hay que voltear una ficha o más hemos encontrado el pat
    else
        return NO_HAY_PA                    ay que voltear no hay patrón
    …
}
```



Medir Tiempos

- Utilizar *timer* para medir tiempos
- monótono
  - timer2_inicializar()
  - timer2_empezar()
  - timer2_leer():



Esquema

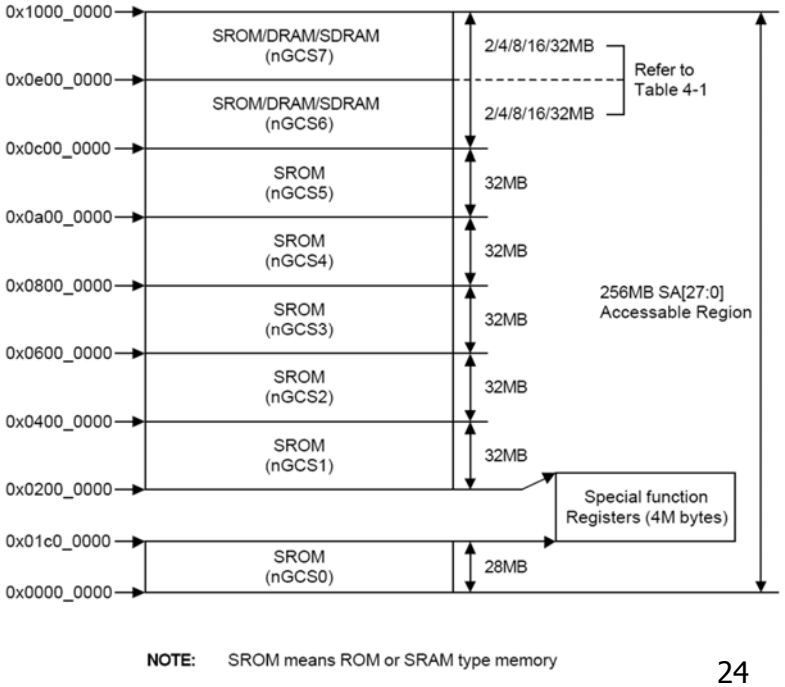
- Descripción de la práctica
- **Sistema de memoria de la placa S3CEV40**
- Sistema de E/S de la placa S3CEV40
- Fuentes y ayudas de C

Espacio de direcciones ARM7TDMI

- Principales características:
  - Direcciones de 32 bits
  - 4GB de tamaño
  - Espacio compartido por E/S y memoria
    - E/S localizada en memoria: los puertos de E/S forman parte del espacio de direcciones
    - La distinción es externa al procesador

Espacio de direcciones del S3C44B0X

- Espacio dividido en 8 fragmentos de 32MB
  - Gestionado por el controlador de memoria
  - En función de los bits 28-24 de la dirección se genera la correspondiente señal CS (nGCS0-7)
  - El comportamiento de cada banco es configurable
  - Es necesario configurar los bancos antes de poder usarlos
  - Usaremos un script



NOTE: SROM means ROM or SRAM type memory

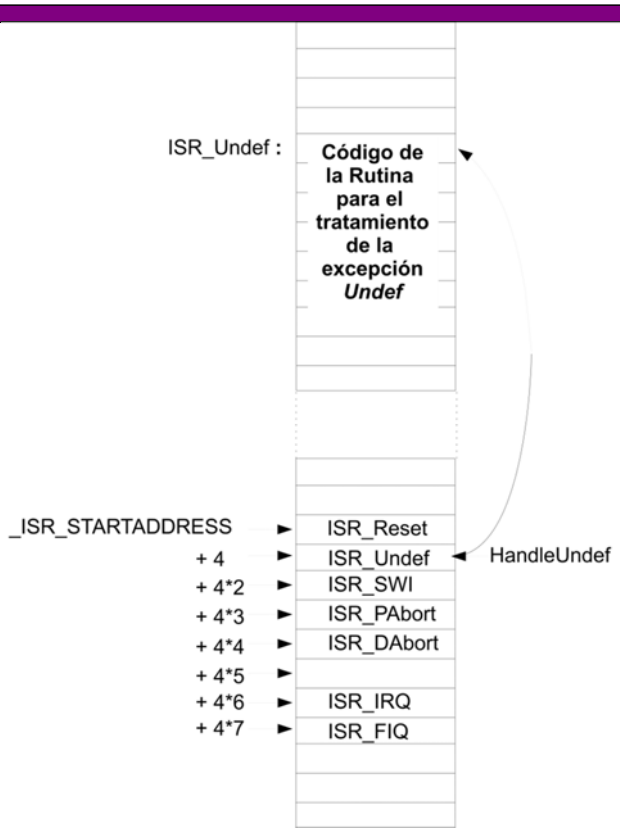
Espacio de direcciones del S3C44B0X

- Zona de registros especiales
  - Ocupa el último tramo del banco-0 [0x01C0_0000-0x01F_FFFF]
  - Se corresponde con los registros internos del S3C44B0X
  - No gestionada por el controlador de memoria
  - La entrada/salida se configura escribiendo/leyendo en estos registros

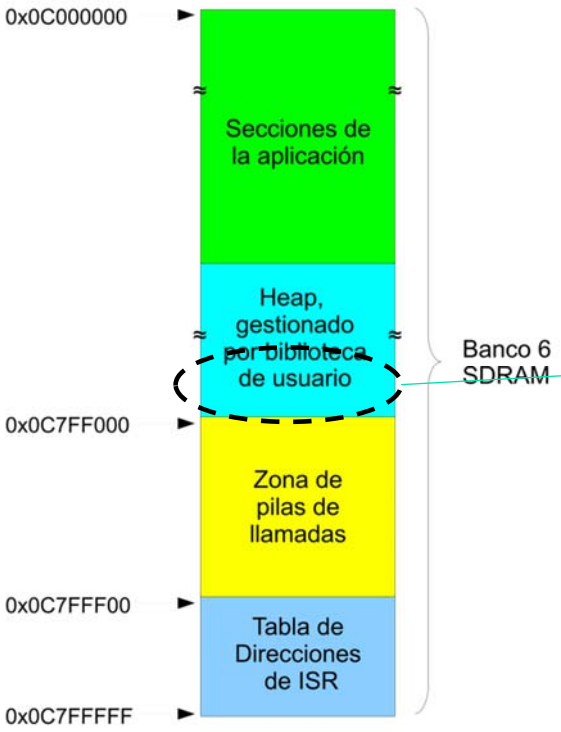
Configuración de la tabla de vectores de interrupción

- La tabla almacena instrucciones de salto a las ISRs
- Se utiliza la etiqueta `ISR_STARTADDRESS` para señalar el comienzo de la tabla de vectores
- A partir de esa etiqueta y en posiciones consecutivas se escriben las direcciones de las ISRs

Configuración de la tabla de vectores de interrupción



Mapa de memoria de un programa de prácticas



La dirección de comienzo de la aplicación la define el linker script

Pila de depuración

La dirección de comienzo de la tabla de interrupciones y de las pilas de cada modo del procesador se definen en el la función ResetHandler dentro del 44binit.s

Esquema

- Descripción de la práctica
- Sistema de memoria de la placa S3CEV40
- **Sistema de E/S de la placa S3CEV40**
- Fuentes y ayudas de C

Sistema de E/S del ARM7TDMI

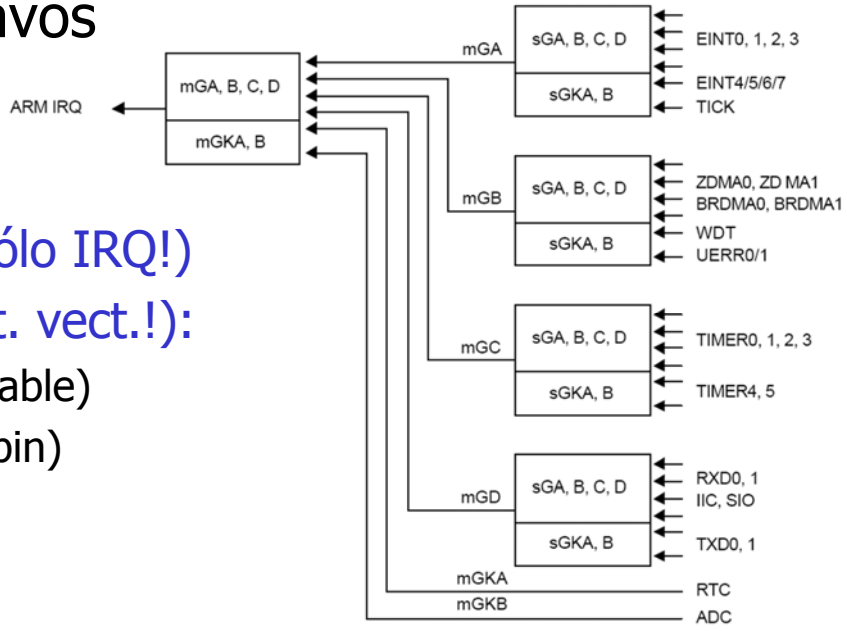
- Principales características:
  - E/S localizada en memoria (los puertos de E/S forman parte del espacio de direcciones)
  - 1 línea externa de RESET
  - 2 líneas de interrupción externas: IRQ y FIQ
  - Interrupciones autovectorizadas
  - Identificación de dispositivos mediante encuesta

Sistema de E/S del S3C44B0X

- Controlador de interrupciones
  - Permite ampliar el nº de líneas de petición de interrupción del ARM7TDMI
  - 30 posibles fuentes de interrupción usando 26 líneas (algunas fuentes comparten línea)
  - Añade soporte de **interrupciones vectorizadas** (¡Sólo para aquellas fuentes que usan IRQ!)
- En esta práctica usaremos esta opción
  - Implementado mediante 5 controladores encadenados (1 maestro + 4 esclavos)

Controlador de interrupciones

- 26 líneas de interrupción
- 1 maestro + 4 esclavos
- Configurable:
  - Modo IRQ/FIQ
  - Int. vectorizadas (isólo IRQ!)
  - Prioridades (isólo int. vect.!):
    - Orden (fijo/programable)
    - Modo (fijo/round-robin)



Controlador de interrupciones

- Registros de configuración
  - INTCON (Interrupt Control Register), 3 bits
    - V (bit [2]) = 0, habilita las interrupciones vectorizadas
    - I (bit [1]) = 0, habilita la línea IRQ
    - F (bit [1]) = 0, habilita la línea FIQ
  - INTMOD (Interrupt Mode Register), 1 bit por línea
    - 0 = modo IRQ; 1 = modo FIQ

Controlador de interrupciones

- Registros de gestión
  - INTPND (Interrupt Pending Register), 1 bit/línea
    - 0 = no hay solicitud; 1 = hay una solicitud
  - INTMSK (Interrupt Mask Register), 1 bit/línea
    - 0 = int. disponible; 1 = int. enmascarada
  - I_ISPC (IRQ Int. Service Pending Clear Reg.), 1 bit/línea
    - 1 = borra el bit correspondiente del INTPND e indica al controlador el final de la rutina de servicio **(¡FIN RUTINA SERVICIO!)**
  - F_ISPC (FIQ Int. Service pending Clear Reg.), 1 bit/línea
    - Ídem

Controlador de interrupciones

- Registros de gestión para int. vectorizadas:
  - I_ISPR (IRQ Int. Service Pending Reg.), 1 bit/línea
    - Indica la interrupción que se está sirviendo actualmente
    - Sólo el bit de la línea más prioritaria puede estar a 1 (≠INTPND)

Controlador de interrupciones

- Funcionamiento interrupciones vectorizadas:
  - Cuando el ARM intenta leer la instrucción en la dir. 0x18 (vector IRQ)
  - El controlador actúa sobre el bus de datos e inserta una instrucción de salto al vector correspondiente a la línea más prioritaria activa
    - EINT0 (0x20)
    - EINT1 (0x24)
    - …
    - EINT4/5/6/7 (0x30)
    - …
    - INT_ADC (0xc0)

Controlador de interrupciones

- Código de ejemplo para int. vectorizadas:

```
…
b HandlerIRQ                ; 0x18
b HandlerFIQ                ; 0x1c
ldr pc,=HandlerEINT0        ; 0x20
ldr pc,=HandlerEINT1        ; 0x24
ldr pc,=HandlerEINT2    ; 0x28
ldr pc,=HandlerEINT3    ; 0x2c
ldr pc,=HandlerEINT4567    ; 0x30
…
```

Declaración de funciones para gestionar interrupciones en C

- `gcc` dispone de un atributo de función para especificar su uso como gestor de interrupciones
  - `void f () __attribute__ ((interrupt ("IRQ")));`
- ¿Cuál sera la diferencia del bloque de activación en estas funciones?
- La documenación completa se encuentra en `https://gcc.gnu.org/onlinedocs/gcc-7.1.0/gcc/ARM-Function-Attributes.html`

S3C44B0X: Controlador de interrupciones

- Desde lenguaje C:

```
#include "44b.h"
...
void
  mi_handler_EINT1(void) __attribute__ ((interrupt ("IRQ")));
...
pISR_EINT1 = (unsigned) mi_handler_EINT1;
```

Cabecera de macros que permite manipular las direcciones usuales de manera simbólica

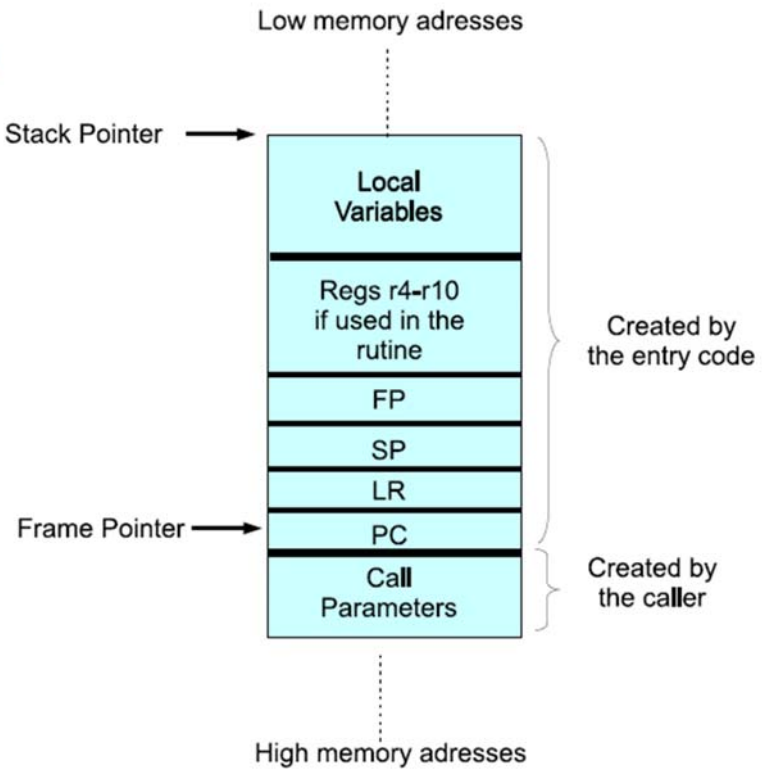
Recordatorio: Marco de pila

- Estructura de una rutina:
  - Prólogo

```
MOV     IP, SP
STMDB   SP!, {r4-r10,FP,IP,LR,PC}
SUB     FP, IP, #4
SUB     SP, #SpaceForLocalVaribles
```

  - Epílogo:

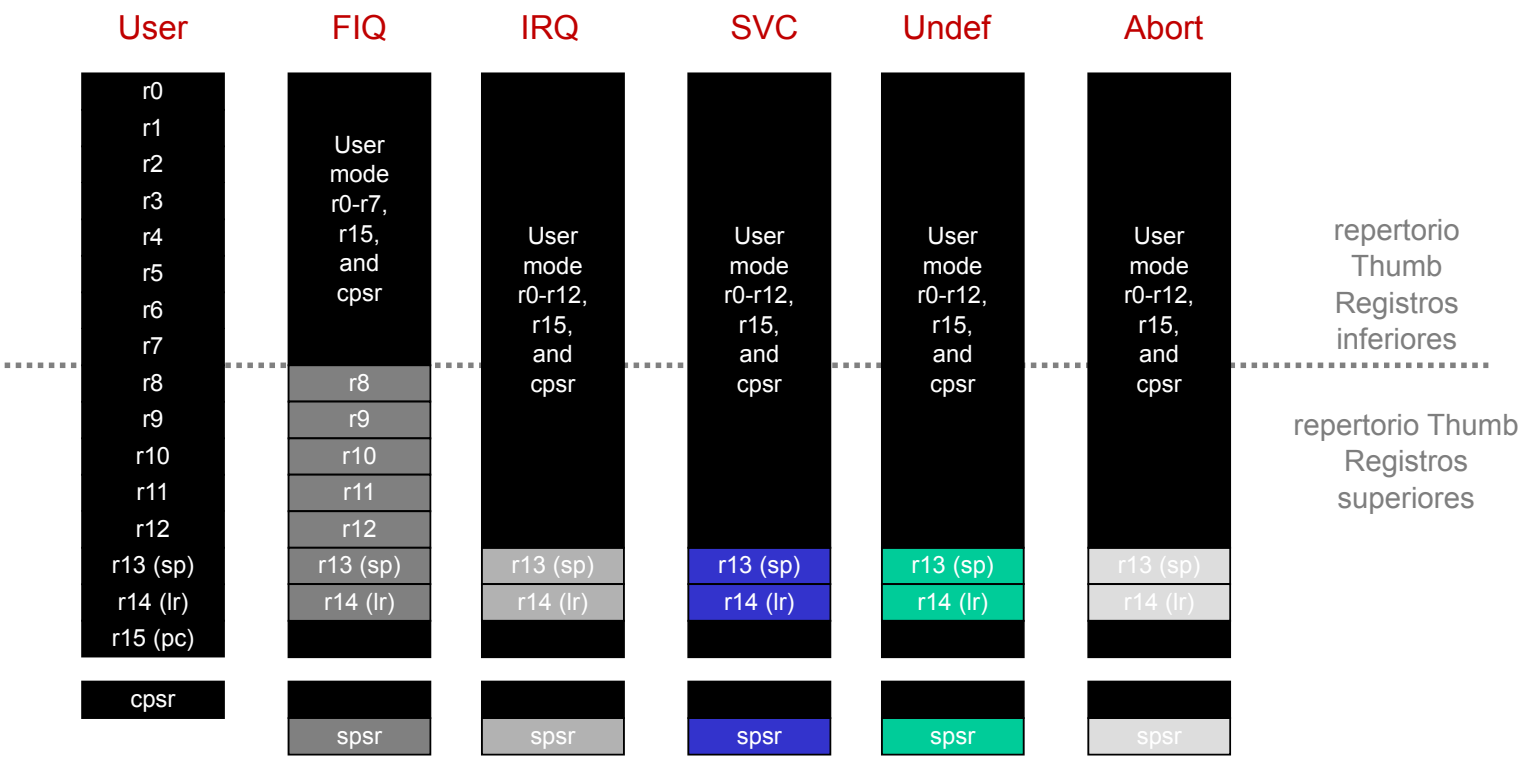
```
LDMDB   FP, {r4-r10,FP,SP,PC}
```



Marco de pila para excepciones

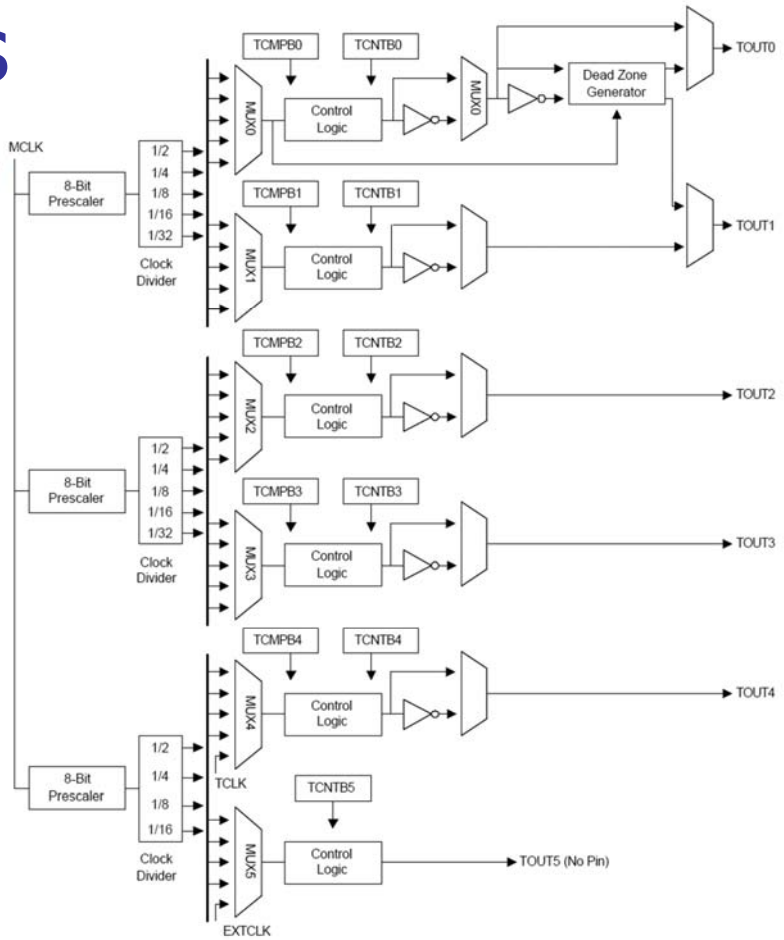
- La gestión es muy parecida:
- Prólogos y epílogos casi idénticos al anterior
- Diferencias:
  - Antes de guardar LR hay que restarle 4 (8 para las excepciones abort) **¿Por qué?**
  - Hay que guardar todos los registros: **también IP y r0-r3!**
  - Al entrar y al salir el procesador cambia de modo:
    - En la entrada el procesador cambia de forma automática en función del tipo de excepción
    - Al terminar para volver al modo anterior hay que añadir "^" a la instrucción de retorno

Registros y modos



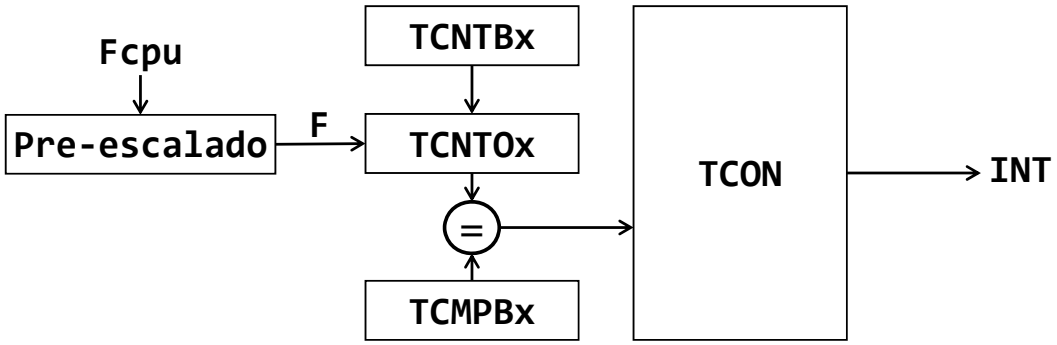
Temporizadores

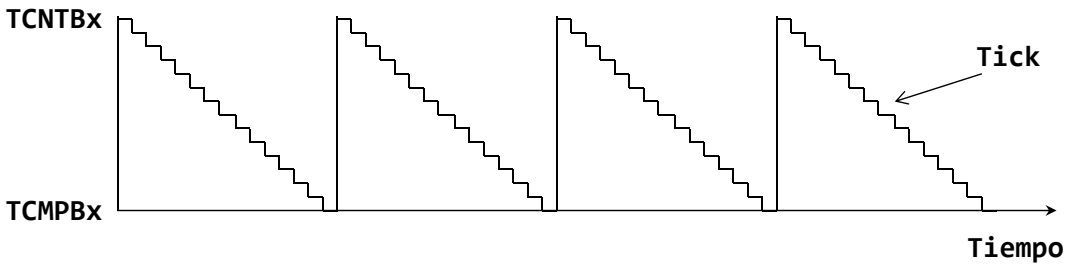
- Esquema Pulse Width Modulation (PWM) Timers
  - 5 temporizadores con PWM y conexión externa
  - 1 temporizador sin PWM y sin conexión externa



Temporizadores

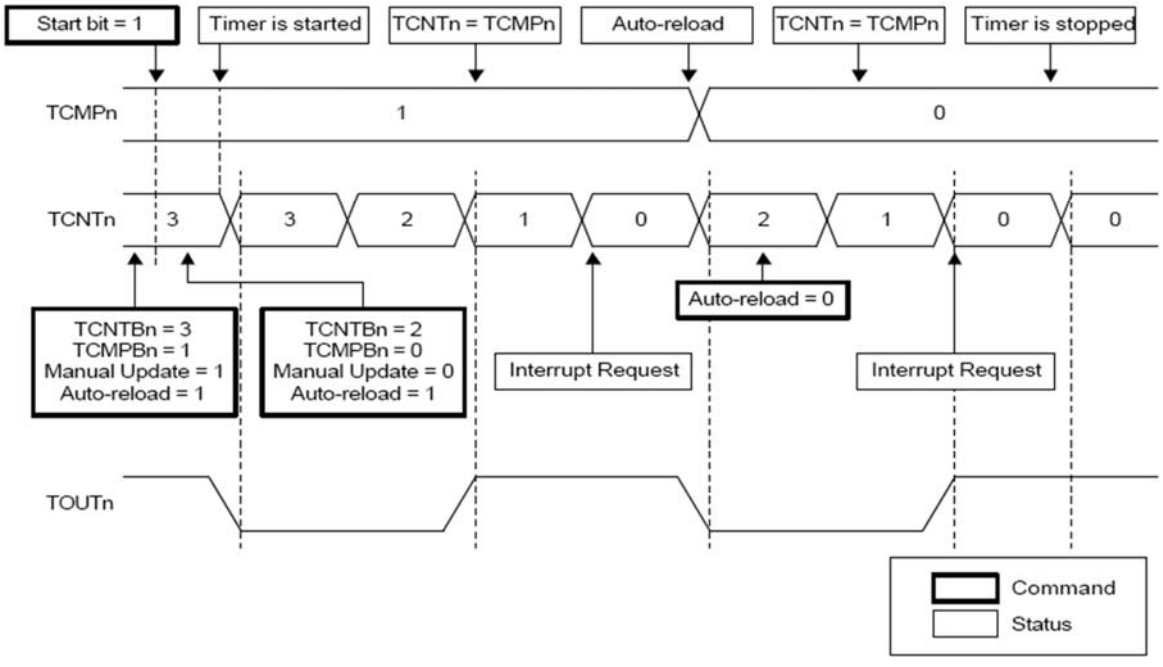
- Comportamiento básico





Temporizadores

- Comportamiento básico



Temporizadores

- Registros
  - TCFG0 (Timer configuration register 0)
    - Configura los módulos de pre-escalado
  - TCFG1 (Timer configuration register 1)
    - Configura cuál es el temporizador que usará la DMA, y para cada temporizador selecciona la salida del divisor de frecuencia
  - TCON (Timer control register)
    - Controla el comportamiento de los temporizadores (start/stop, auto-reload, etc.)

Temporizadores

- Registros
  - TCNTB0-5 (Count buffer register 0-5)
    - Registro de buffer del valor de inicialización
  - TCMPB0-5 (Compare buffer register 0-5)
    - Registro de buffer del valor de comparación
  - TCNTO0-5 (Count observation register 0-5)
    - Registro que permite consultar el valor actual del temporizador

Esquema

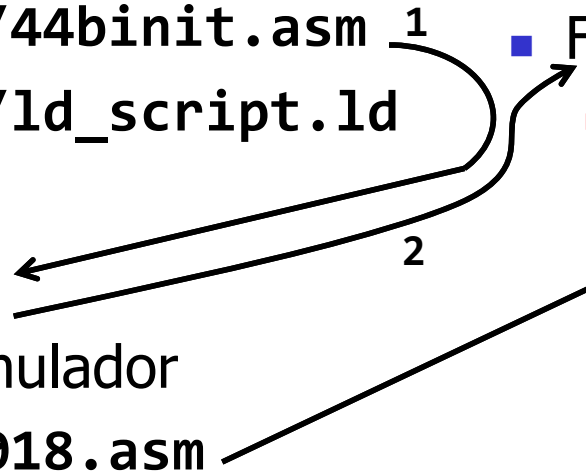
- Descripción de la práctica
- Sistema de memoria de la placa S3CEV40
- Sistema de E/S de la placa S3CEV40
- **Fuentes y ayudas de C**

Ficheros de partida

- **`ev40boot.cs`:** realiza la inicialización necesaria antes de poder cargar el programa (bancos de memoria, etc.)
- **`44binit.s`:** inicialización y tabla de vectores (inst. de salto)
- **`44b.h`:** macros para usar nombres simbólicos
- **`44blib{.h,.c}`**: códigos de utilidad para el desarrollo en C

Fuentes reversi

- Fuentes Placa
  - **./common/44binit.asm**
  - **./common/ld_script.ld**
  - **main.c**
    - Main()
- Fuentes Emulador
  - **init_b_2018.asm**
  - **ld_script.ld**
- Fuentes Juego
  - **reversi8_2019.c**
    - no hay main()
    - reversi8()



Fuentes reversi emulador

- init_b_2018.asm
  - inicio
- ld_script.ld
  - loader script
- reversi8_2019.c
  - funciones principales
  - no hay main()

```
/*init_b_2018.asm*/
.text
# ENTRY /*mark the first instruction to call */
.global start
start:
.arm /*indicates that we are using the ARM isa*/
#------standard initial code
# --- Setup interrupt / exception vectors
  B    Reset_Handler

…

Reset_Handler:
#
  MOV  sp, #0x4000 /*set up stack pointer(r13)*/

.extern     reversi8
  ldr  r5, = reversi8
  mov  lr, pc
  bx   r5
```

Variables en C

- visibilidad, alcance y cualificadores
  - global, local
  - `const (#define, enum)`
  - `volatile`
  - `extern`
  - `static`
    - en globales
    - en locales
    - en funciones

Volatile

- `volatile` es una palabra reservada que indica que una variable puede ser modificada **concurrentemente** por un operador externo aunque parezca que desde el código no se modifique
- Ejemplo de variables que necesiten ser definidas como volátiles son todos los registros de E/S de periféricos o las variables compartidas con *rsi.*

Volatile: Ejemplo de uso

- Temporizador que espera 10 ticks de reloj para imprimir por pantalla

```
volatile uint32_t ticks = 0;
while( ticks < 10);
printf(”10 ticks\n”);
```

- Sin `volatile` un compilador podría asumir que es un bucle infinito y optimizarlo como while(1);

Static

- `static` es una palabra reservada en C que indica que la duración de la variable es la duración del programa y que la variable o función sólo es visible en su unidad de compilación (fichero) (internal linkage).
- Restringir el alcance y visibilidad de variables y funciones reduce errores y toda función que sólo deba emplearse en el fichero actual deberá estar definida como static.

Static: Ejemplo de uso

reloj.c

```
static unsigned long eventos = 0;
void reloj_eventos() { eventos++; … }
```

teclado.c

```
static unsigned long eventos = 0;
void teclado_eventos() { eventos++; }
```

- El enlazador no tendrá problemas con ambos ficheros objetos y sabrá que tiene que definir 2 variables globales distintas en la sección de datos del ejecutable