

Práctica 2: Sistema de E/S y dispositivos básicos

Proyecto Hardware

Javier Resano, Grupo GAZ, DIIS Universidad de Zaragoza
Agradecimientos a DACYA, Universidad Complutense de Madrid

Esquema

- **Objetivos y descripción de la práctica**
- **Sistema de memoria de la placa S3CEV40**
- **Sistema de E/S de la placa S3CEV40**

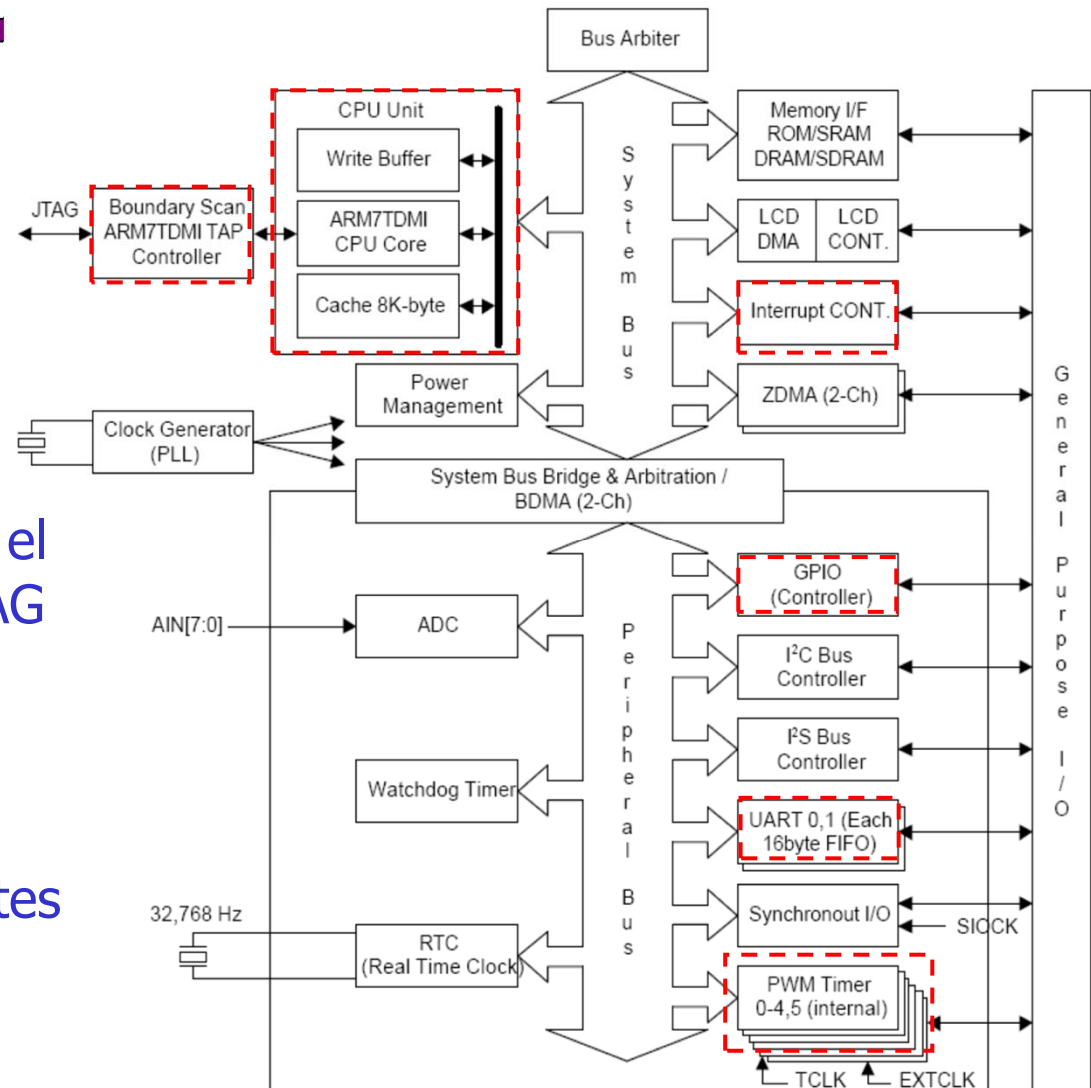
Objetivo: Desarrollar código para un sistema empuotrado real

- Se parte del código desarrollado para un procesador ARM en la práctica 1. Queremos utilizar este código en un entorno real:
 - Placa de desarrollo de ARM
- En esta placa el procesador está acompañado de muchos dispositivos, principalmente de entrada/salida
- Vamos a aprender a interaccionar con ellos trabajando en C, pero siendo capaces de bajar a ensamblador cuando sea necesario

Chip principal de la placa: S3C44B0X

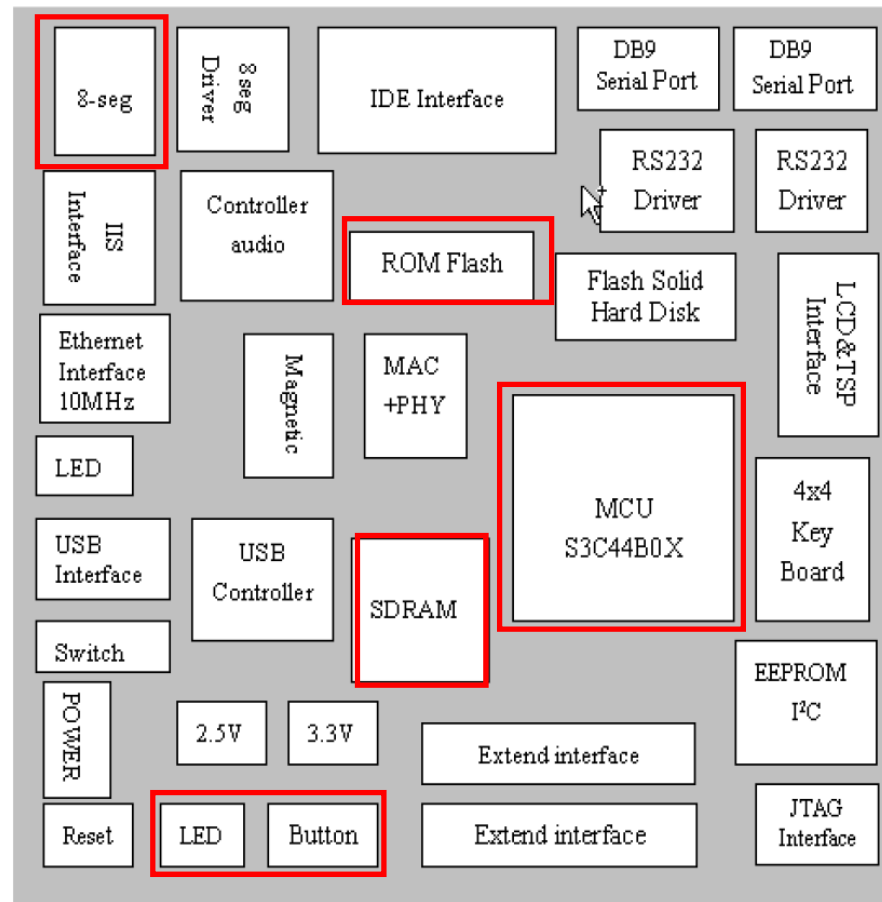
■ System-on-Chip (Soc):

- Procesador ARM7TDMI
- Controladores E/S
- Buses
- Comunicación directa con el PC a través del puerto JTAG
- Temporizadores
- Controlador de interrupciones
- Y muchos más componentes



Elementos en la placa Embest S3CEV40

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



¿Qué tenemos que aprender?

- Cómo configurar la placa para poder utilizarla:
 - Utilización de un script de configuración para inicializar el espacio de memoria
 - 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)

Objetivos: trabajar con una placa real

- Interactuar con una placa real y ejecutar el código desarrollado en la práctica anterior
- Gestionar la entrada/salida con dispositivos básicos, desde un programa en C (utilizando las librerías de la placa)
- Desarrollar en C las rutinas de tratamiento de interrupción
- Utilizar los temporizadores internos de la placa y la entrada/salida de propósito general

Objetivos: aprender a depurar

- Depurar un código con varias fuentes de interrupción activas
- Depurar un código en ejecución (no sólo paso a paso)

Descripción de la práctica

- Estudiar el proyecto que os damos y aprender a usar:
 - Botones
 - 8led (7-segmentos)
 - Temporizadores (timers)

Descripción de la práctica

- Utilizar los timers para medir tiempos
- Gestión de las excepciones
 - realizar una función de tratamiento de excepciones
 - Al producirse una excepción (data abort, undefined, prefetch abort) se invocará a esta función.
 - Identifica qué excepción se ha producido
 - Para la ejecución
 - Avisa al usuario (ejemplo led parpadeando)

Descripción de la práctica

- Desarrollar una pila de depuración
 - *push_debug(int data)* : introduce en la pila dos enteros:
 - Data: se puede usar para identificar los eventos
 - Time: indica cuándo ha sucedido el evento (µs transcurridos desde que empezó la ejecución)
 - Nos dice cuándo se han producido los eventos asíncronos que nos interesan (por ejemplo, los rebotes de los pulsadores)
 - Lista circular: no puede salirse de sus límites

Descripción de la práctica

- Eliminar los rebotes del teclado y pulsadores
- **Meter vuestro código en la placa**
 - Incluir una máquina de estados que gestione la entrada/salida
- **Detectar errores**
 - Si hay algún dato mal puesto lo marcamos en el bit de error de la casilla correspondiente y encendemos un led

Esquema de entrada/salida

- El código comienza al pulsarse un botón
- El botón izquierdo se usa para introducir Fila, Columna y Valor.
- El derecho para confirmar
- El 8led permite visualizar el valor en cada instante

Esquema de entrada/salida

- Al comenzar aparece una F en el 8led
- Al pulsar el botón izquierdo se visualizará un 1
- Si se mantiene el botón pulsado el número se incrementará cada **300 ms** segundo
 - Si llega al 9 y sigue pulsado volverá al 1.
 - Al levantar el dedo se para
 - Al volver a pulsar parte del valor anterior y continua incrementando
- Con el botón derecho se confirma la fila
- Aparecerá una C en el 8led y se repite el proceso para la columna
- Cuando se confirmen la fila y la columna se introducirá el Valor [1..9,0]. Un cero borrara el valor de la celda

Apartados opcionales

- Permitir interrupciones anidadas
- Estudiar el código de la función `init()`, la estructura del linker script y solucionar un problema que puede aparecer al trabajar con chars y esos códigos

Fechas de entrega

- Primera parte: trabajar con el proyecto que os damos
 - Inicio tercera sesión
 - Aproximadamente el 9 de Noviembre
- Segunda parte: portar vuestro código
 - Aproximadamente el 23 de Noviembre
 - Los turnos de corrección aparecerán en Moodle

Material disponible

- En Moodle disponéis de:
 - Un proyecto que utiliza los pulsadores, leds, 8led y el timer0 de la placa:
 - debéis seguir el mismo esquema que este proyecto
 - Cuadernos de prácticas escritos por compañeros de la Universidad Complutense:
 - **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
 - Documentación original de la placa

Esquema

- Objetivos y descripción de la práctica
- **Sistema de memoria de la placa S3CEV40**
- Sistema de E/S de la placa S3CEV40

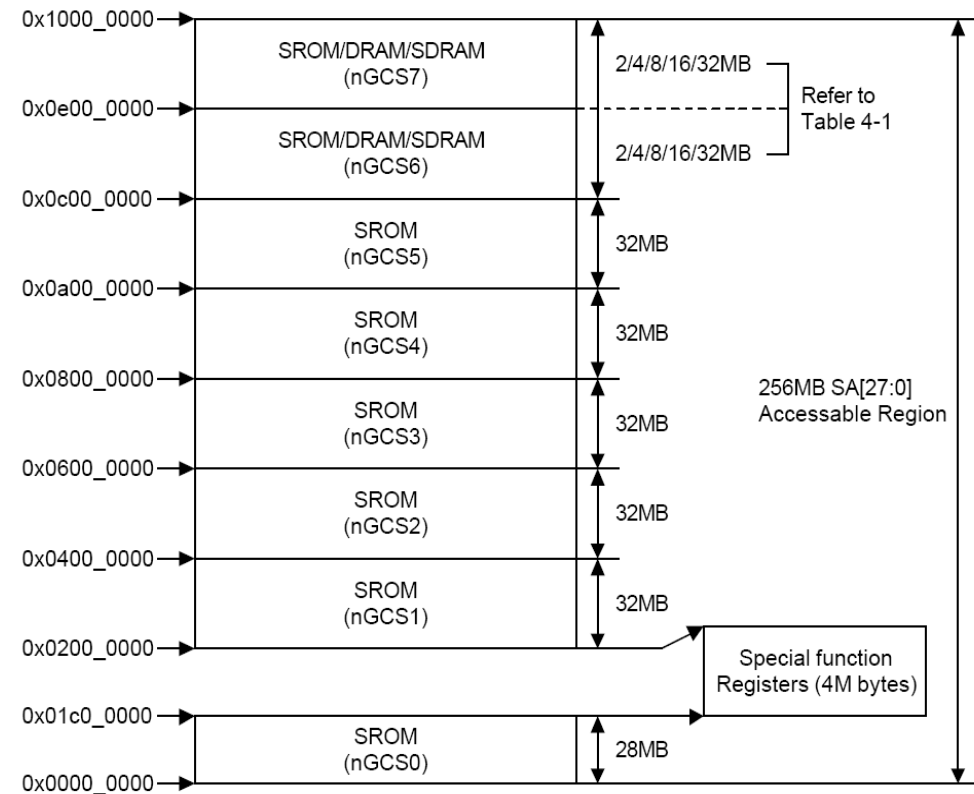
Espacio de direcciones ARM7TDMI

- 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

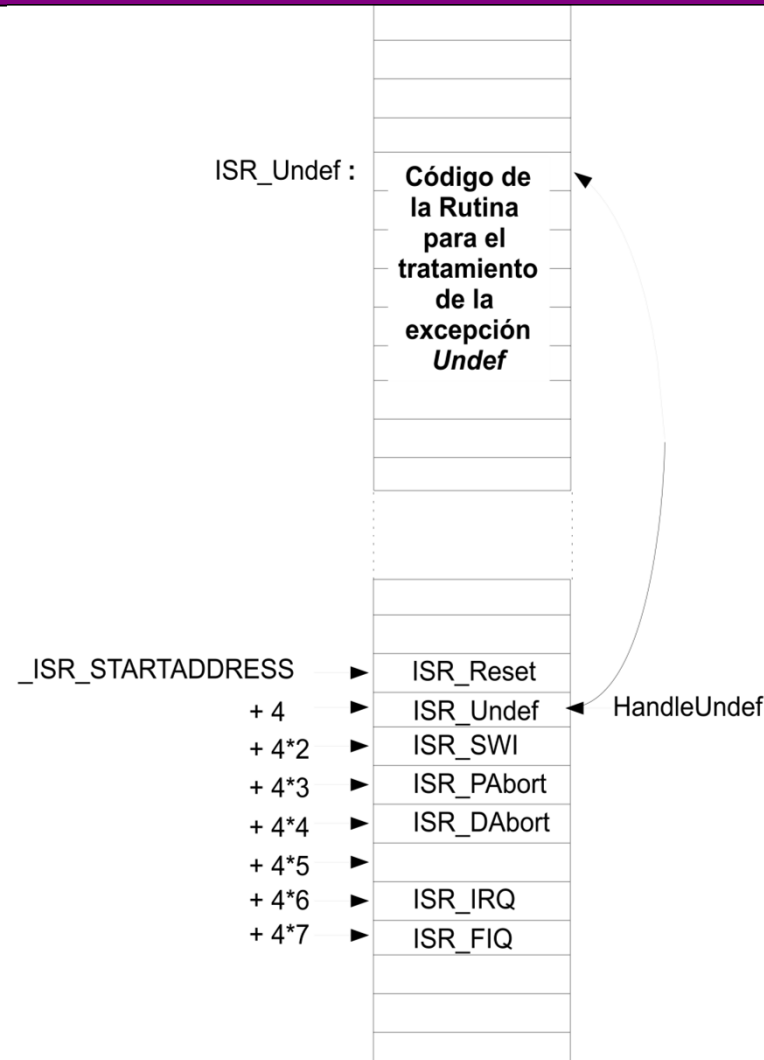
Espacio de direcciones del S3C44B0X

- 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

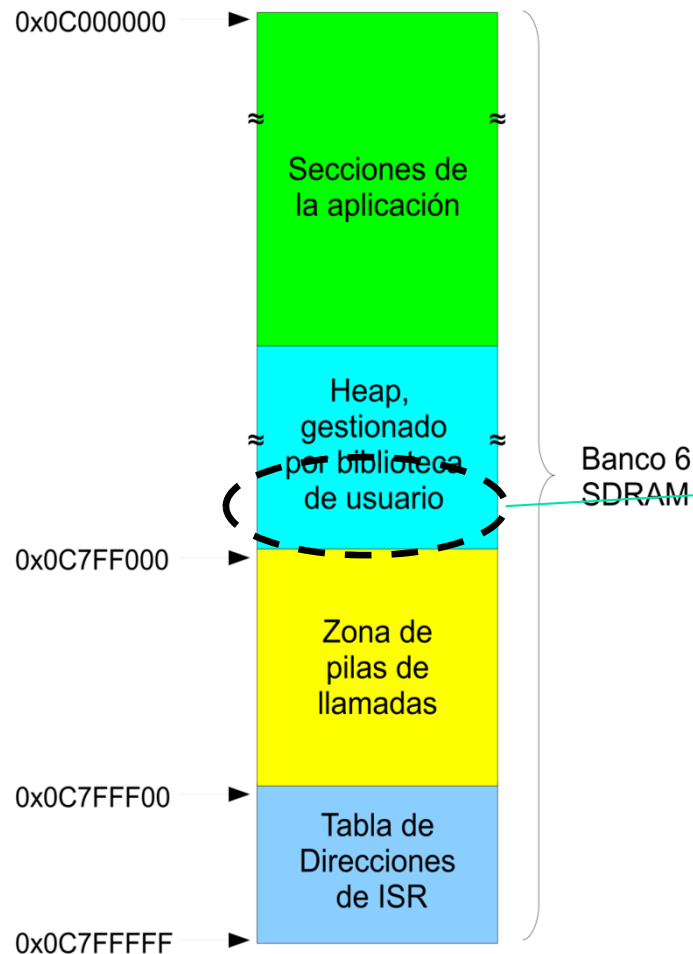
Configuración de la tabla de vectores de interrupción

- 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 la función ResetHandler dentro del 44binit.s

Esquema

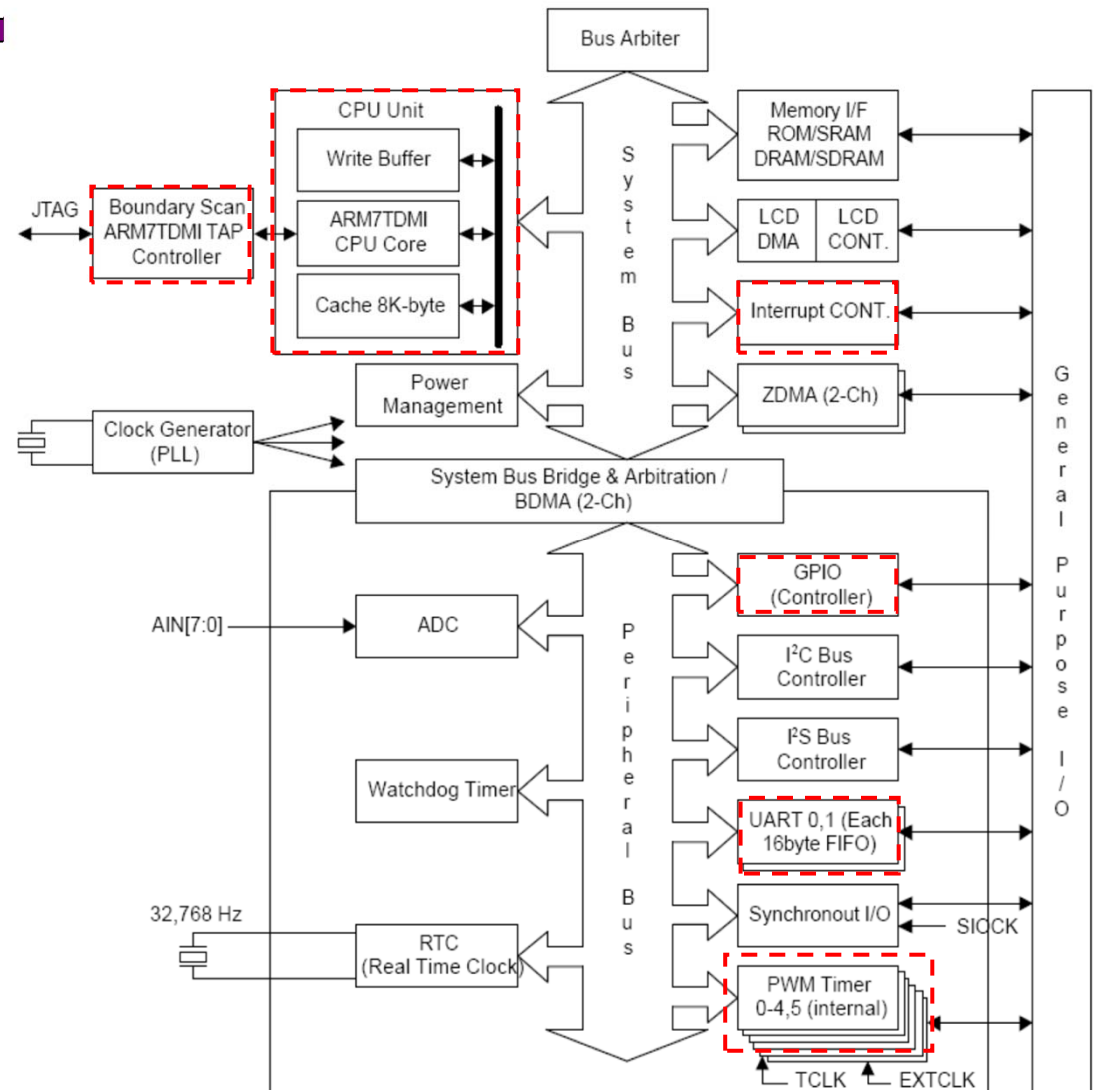
- Objetivos y descripción de la práctica
- Sistema de memoria de la placa S3CEV40
- **Sistema de E/S de la placa S3CEV40**

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

■ Esquema



Sistema de E/S del S3C44B0X

- Puertos de E/S
 - E/S localizada en memoria (ARM7TDMI):
 - Los registros que controlan a los distintos elementos de la placa tienen asignados una dirección de memoria
 - La entrada/salida se gestiona escribiendo/leyendo en esas direcciones
 - Registros especiales (internos) tienen reservado el último tramo del banco-0 [0x01C0_0000 - 0x01F_FFFF]
 - El resto de dispositivos (externos) suelen estar ubicados en el banco-1 [0x0200_0000 - 0x03FF_FFFF]

S3CEV40: Sistema de E/S

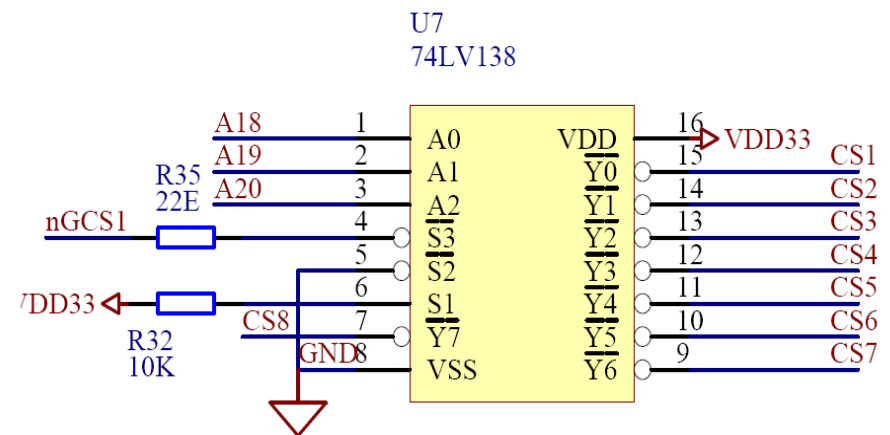
- Dos tipos de dispositivos:
 - Accedidos mediante pines de E/S del S3C44B0X
 - Accedidos mediante direcciones de memoria

Dispositivo	CS	Dirección
USB	CS1	0x0200_0000 – 0x0203_FFFF
Nand Flash	CS2	0x0204_0000 – 0x0207_FFFF
IDE (IOR/W)	CS3	0x0208_0000 – 0x020B_FFFF
IDE (KEY)	CS4	0x020C_0000 – 0x020F_FFFF
IDE (PDIAG)	CS5	0x0210_0000 – 0x0213_FFFF
8-SEG	CS6	0x0214_0000 – 0x0217_FFFF
ETHERNET	CS7	0x0218_0000 – 0x021B_FFFF
LCD	CS8	0x021C_0000 – 0x021F_FFFF
Teclado	nGCS3	0x0600_0000 – 0X07FF_FFFF

S3CEV40: Sistema de E/S

■ Selección de chips

A2	A1	A1	CS	Modulo
0	9	8		
0	0	0	CS1	USB
0	0	1	CS2	Nand Flash
0	1	0	CS3	IDE
0	1	1	CS4	IDE
1	0	0	CS5	IDE
1	0	1	CS6	8-SEG
1	1	0	CS7	ETHERNET
1	1	1	CS8	LCD

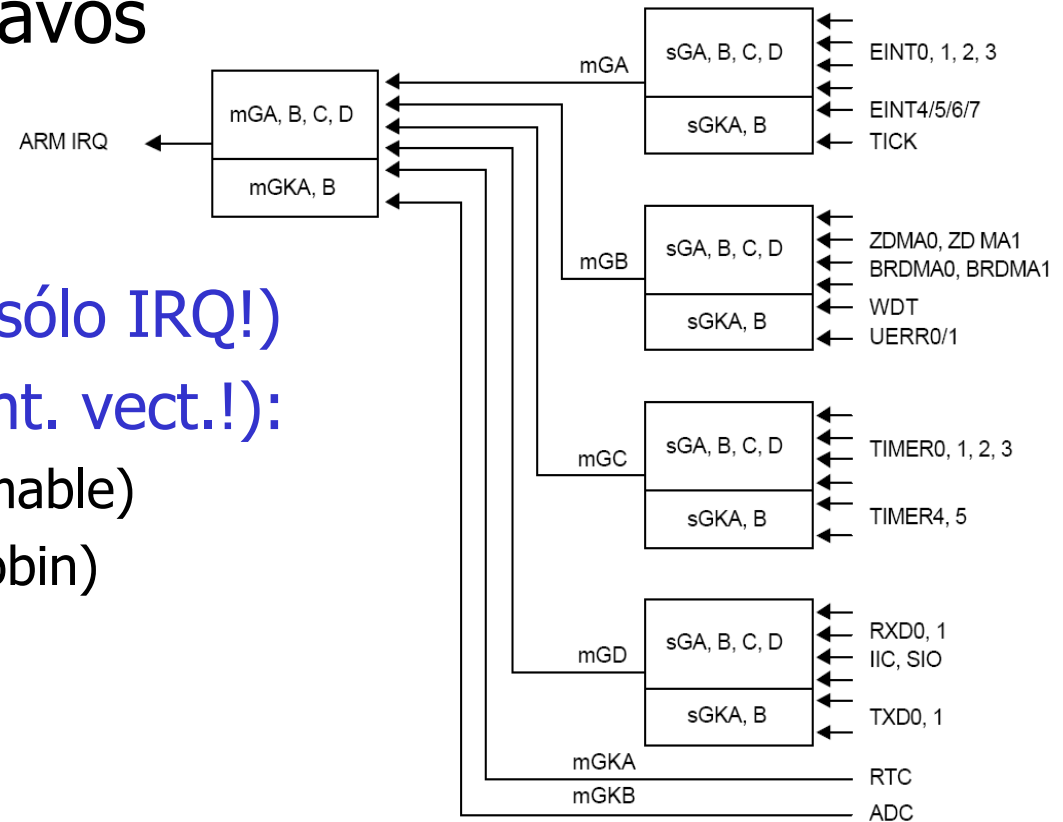


Sistema de E/S del S3C44B0X

- 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** (¡Solo 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 por línea
 - 0 = no hay solicitud; 1 = hay una solicitud
 - INTMSK (Interrupt Mask Register), 1 bit por línea
 - 0 = int. disponible; 1 = int. enmascarada
 - I_ISPC (IRQ Int. Service Pending Clear register), 1 bit por 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 register), 1 bit por línea
 - Ídem

Controlador de interrupciones

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


...

S3C44B0X: Controlador de interrupciones

- Desde lenguaje C:

```
#include "44b.h"
```

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



```
...
```

```
void
```

```
mi_handler_EINT1(void) __attribute__((interrupt ("IRQ")));
```

```
...
```

```
pISR_EINT1 = (unsigned) mi_handler_EINT1;
```

Recordatorio: Marco de pila

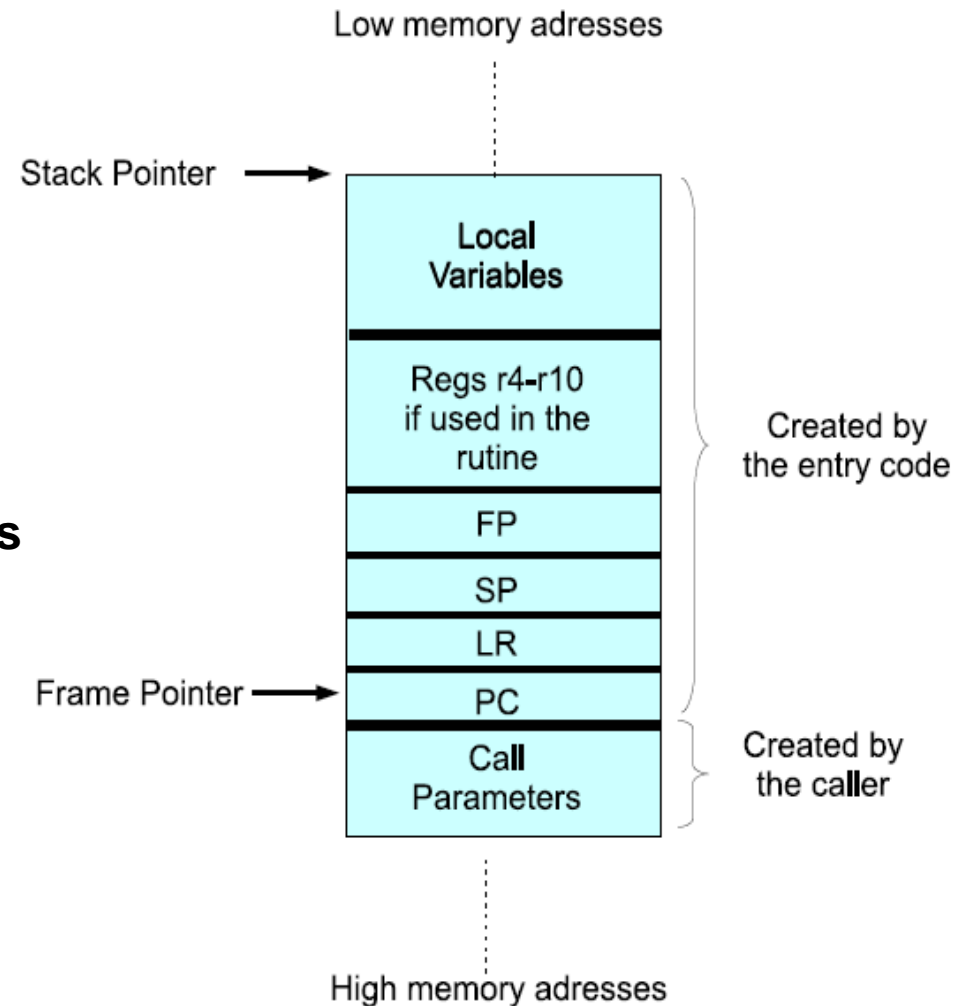
■ Estructura de una rutina:

■ Prólogo

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

■ Epílogo:

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

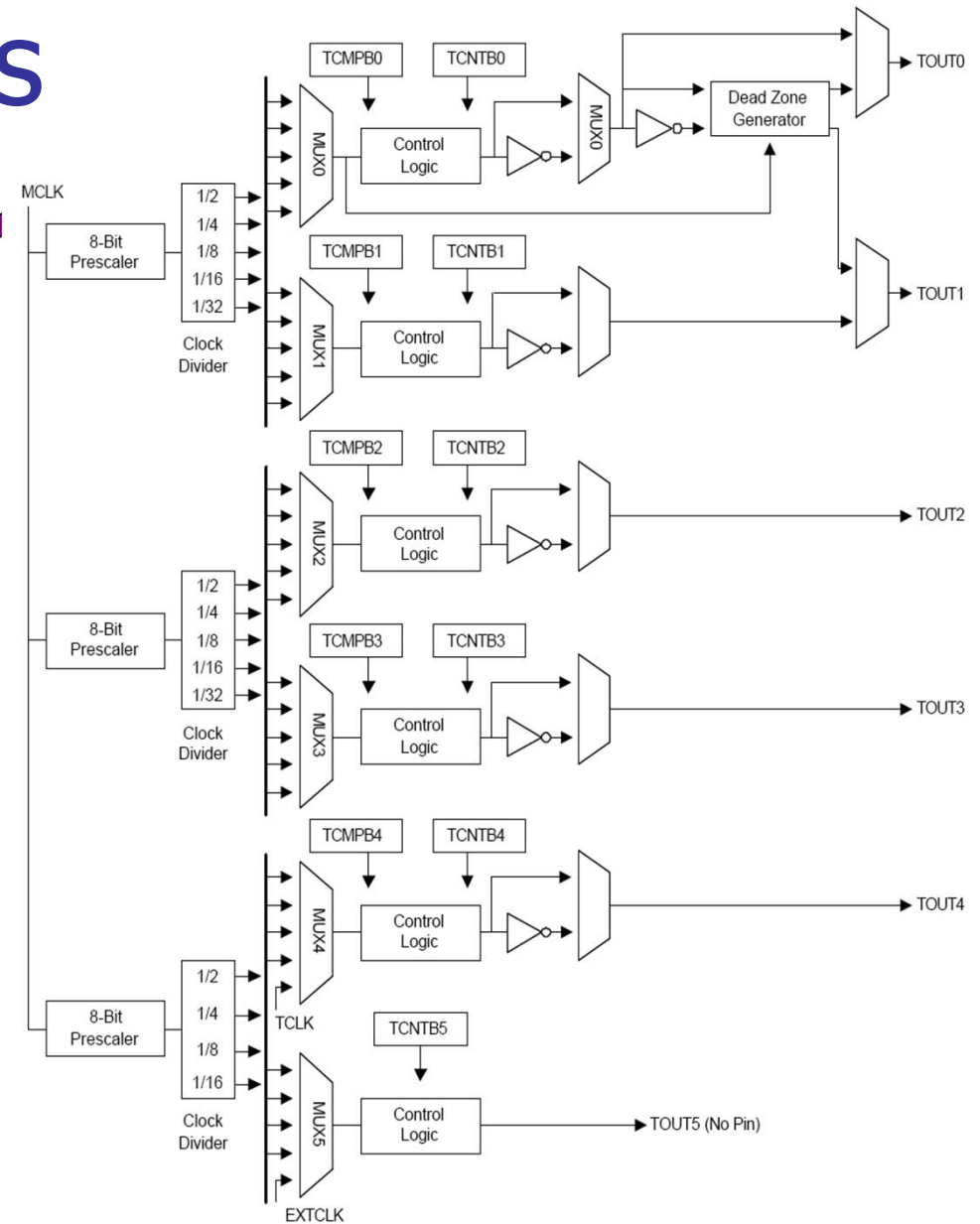


Marco de pila para excepciones

- 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

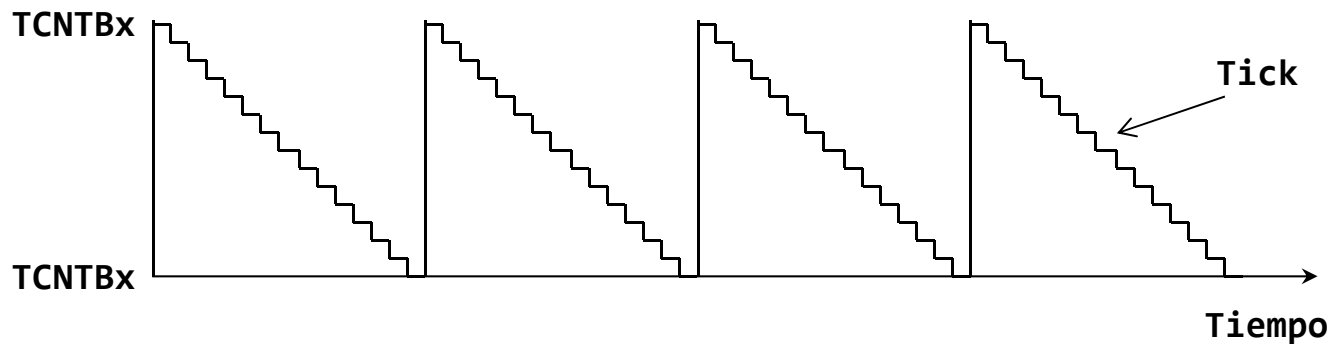
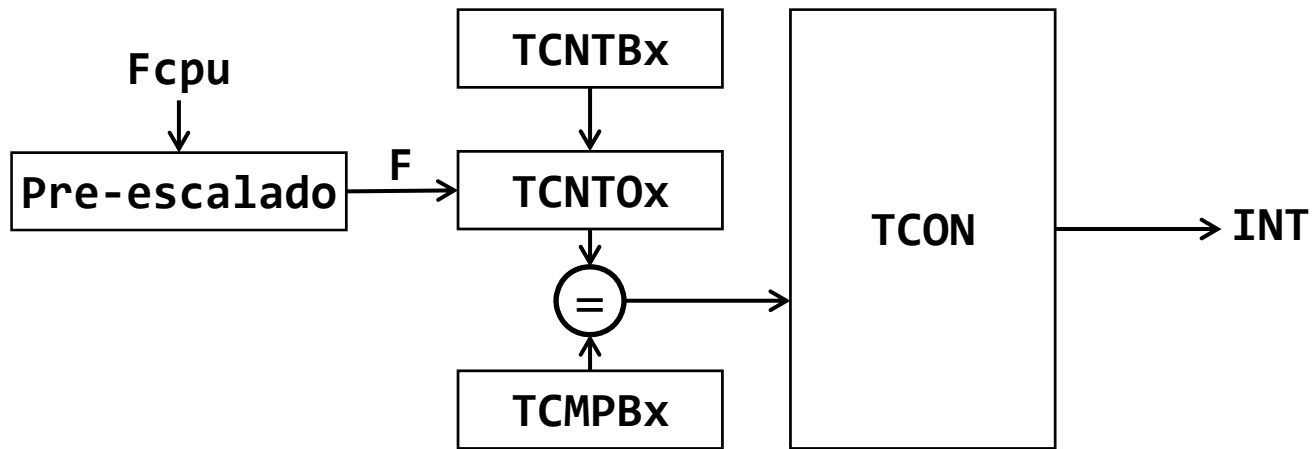
Temporizadores

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



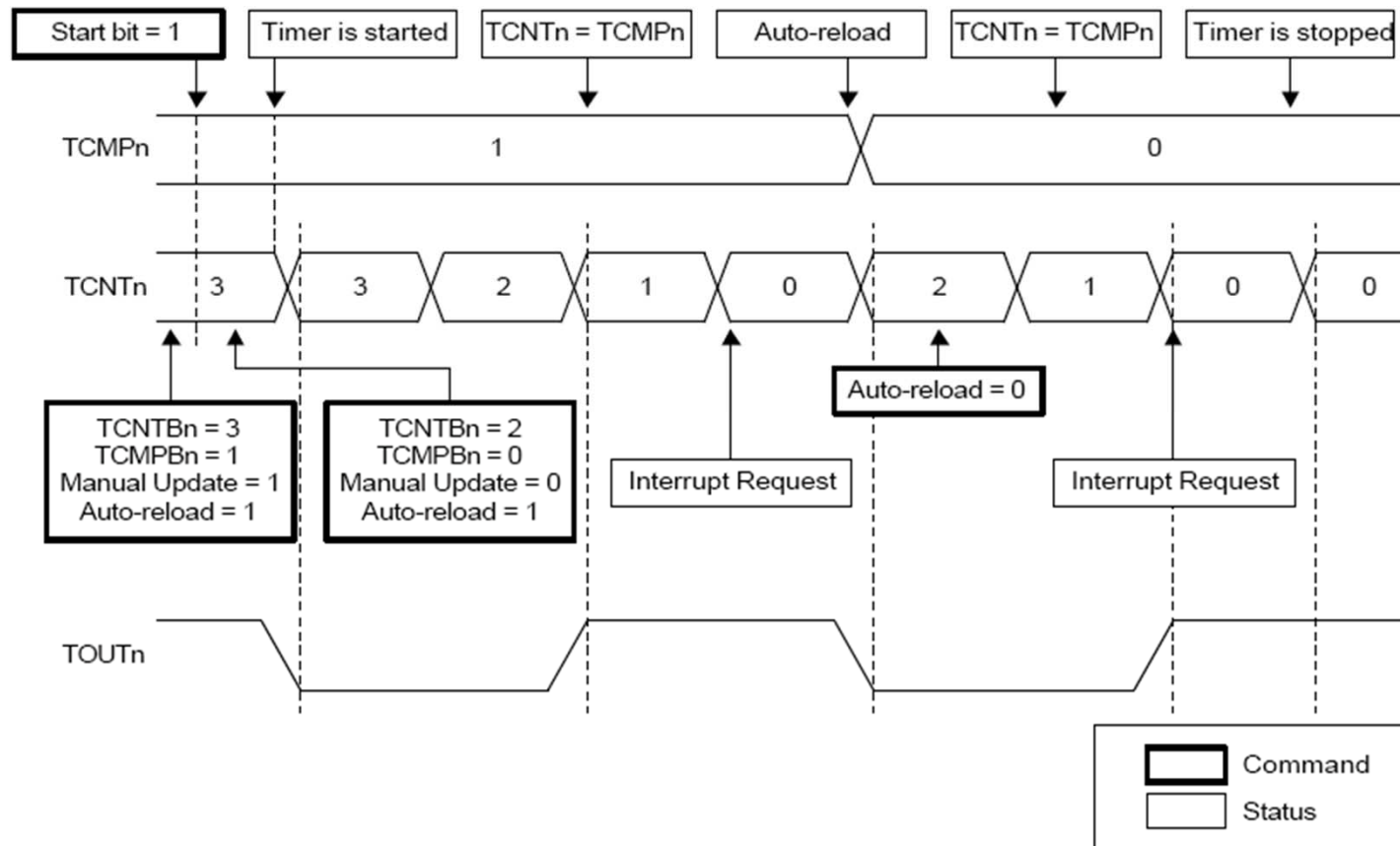
Temporizadores

- 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
- TCNT00-5 (Count observation register 0-5)
 - Registro que permite consultar el valor actual del temporizador

Controlador de pines de E/S (GPIO)

- 71 pines multifuncionales agrupados en 7 puertos (A, B, C, D, E, F y G)
- De momento sólo nos interesan los puertos B y G
- Se manejan mediante 2/4 registros dependiendo del puerto (habitualmente 3)
- Configurados por defecto a un valor seguro

Pulsadores y LEDs

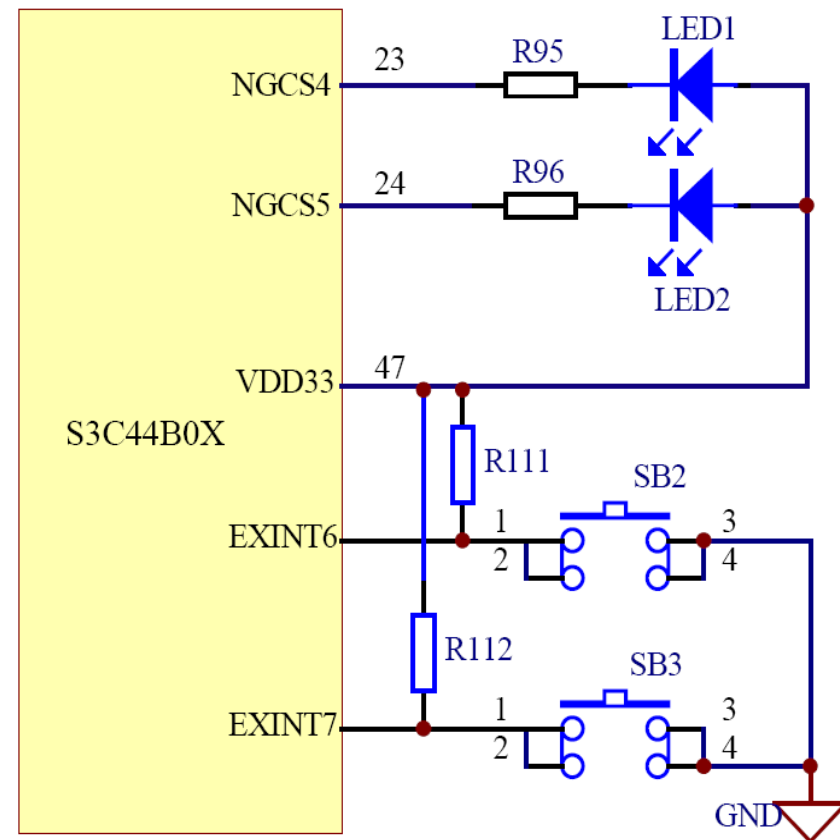
■ Comportamiento:

■ LEDs:

- Si nGS4=0, LED1 on
- Si nGS5=0, LED2 on

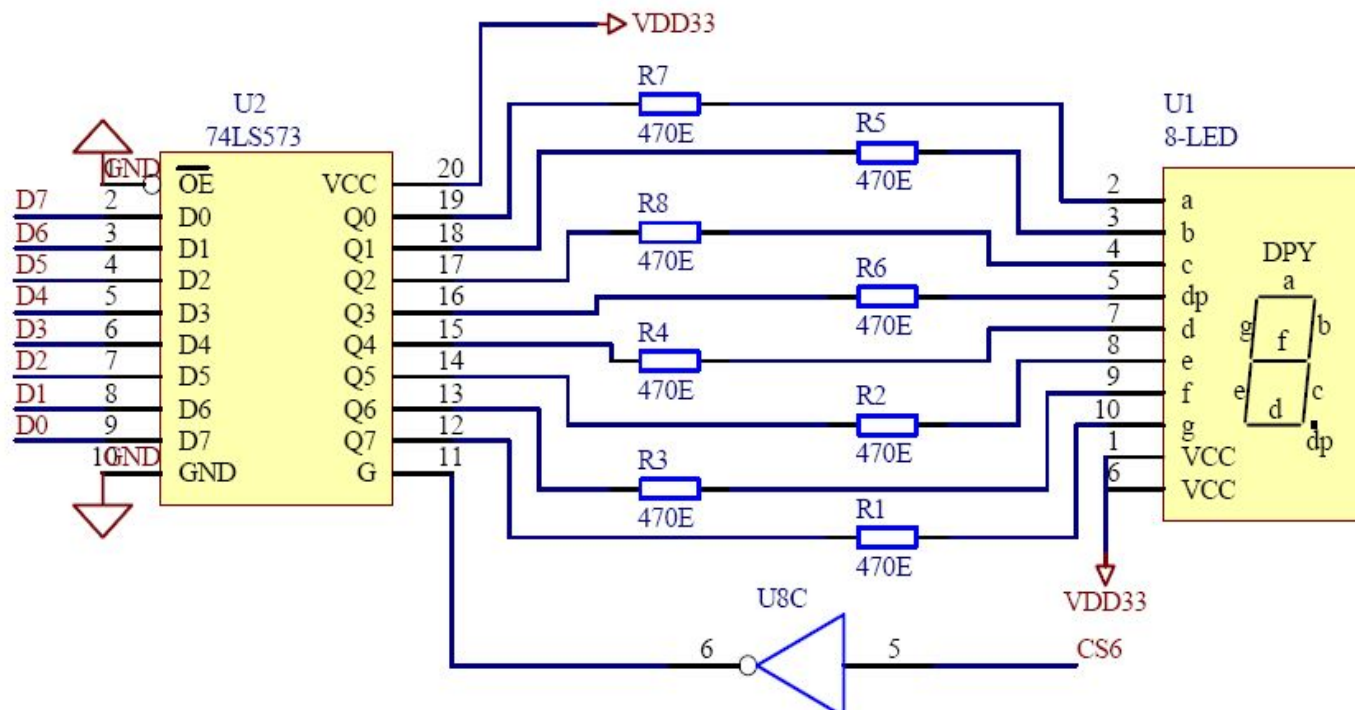
■ Pulsadores:

- SB2 pulsado, EXINT6=0
- SB3 pulsado, EXINT7=0

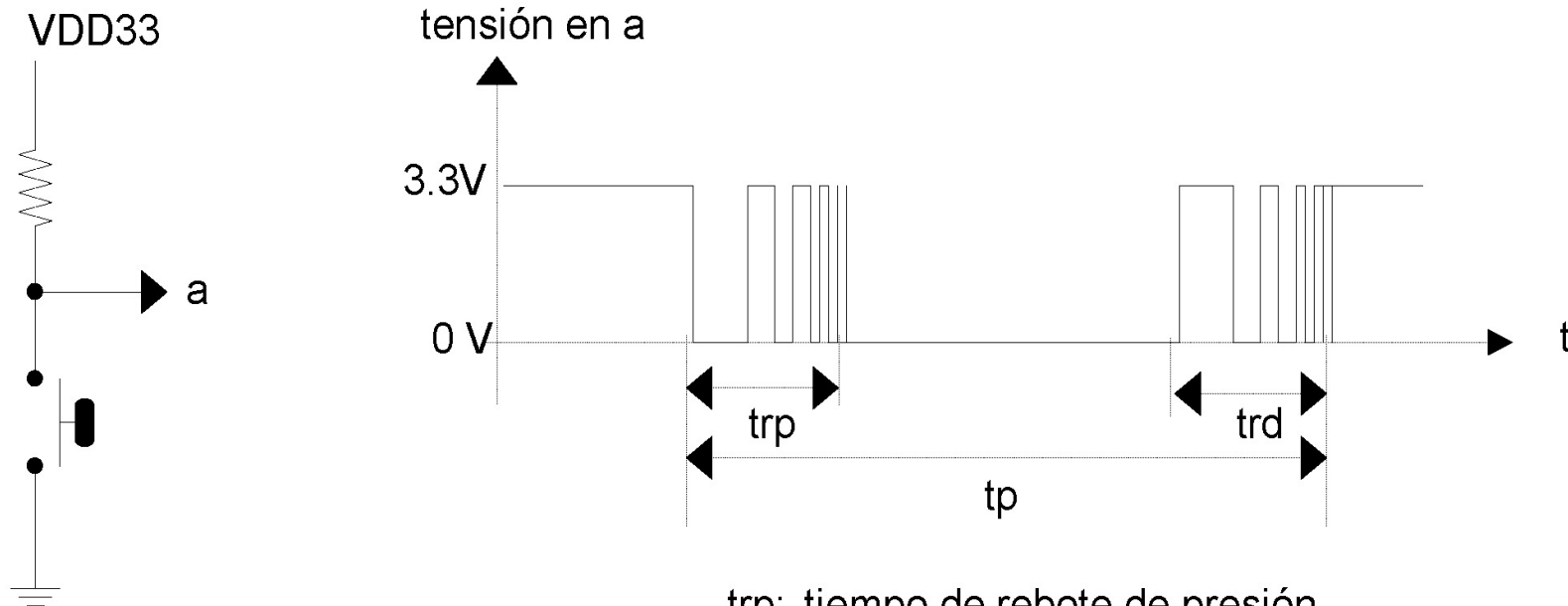


Display 8 segmentos

- Compuesto por 8 diodos
- Se conecta al byte menos significativo del bus de datos mediante latches



Nuestras pulsaciones generan rebotes



trp: tiempo de rebote de presión
trd: tiempo de rebote de depresión
tp: tiempo de presión

El sistema funcionará mal si no se gestionan los rebotes

- Una pulsación puede generar muchas interrupciones:
 - Gestión ineficiente: haces varias veces lo mismo
 - Malfuncionamiento: haces cosas que no deberías hacer

Se pueden solucionar introduciendo retardos

- Rebotes de presión
 - Esperar un tiempo antes de efectuar la identificación de tecla
- Rebotes de depresión
 - Opción 1: Poner un “wait” tras identificar la tecla
 - **¡No es robusta y no nos sirve!:** no sabemos cuándo levantará el dedo el usuario.
 - Opción 2: detectar flanco de subida y de bajada
 - Tendremos una interrupción al presionar y al levantar
 - Opción 3: comprobar periódicamente si la tecla está pulsada o no
 - Podemos hacerlo mirando el bit correspondiente del registro **PDATG**
 - Al pulsar generamos un retardo antes de identificar la tecla
 - Al levantar generamos un retardo antes de habilitar de nuevo la interrupción correspondiente
 - **Vamos a seguir este esquema**

Los retardos se pueden gestionar con timers

- Hay dos formas de gestionar los retardos:
 - Espera activa: el procesador entra en un bucle en el que ejecuta NOPs el tiempo que sea necesario
 - Es muy **ineficiente** porque no puede hacerse trabajo útil durante la espera, y **peligrosa** porque si no se tiene cuidado puede generar bloqueos: **¡no nos sirve!**
 - Programar un temporizador:
 - Le pedimos a un temporizador que nos avise
 - Mientras tanto el procesador puede hacer otras cosas

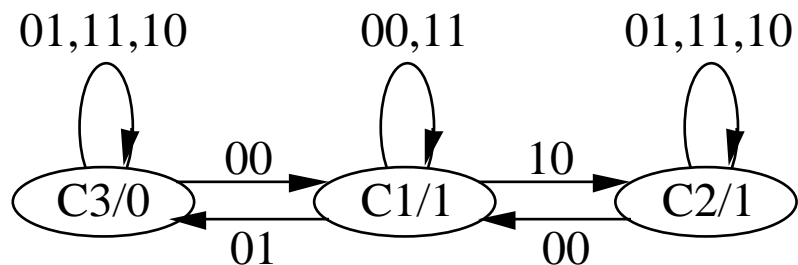
Pasos a seguir en la gestión de los rebotes:

- Al detectar la pulsación:
 - deshabilitamos las interrupciones de los botones
 - programamos un temporizador para introducir un retardo inicial
- Tras el retardo inicial:
 - identificamos el botón pulsado
 - Programamos un temporizador para que cada 10 ms compruebe si sigue pulsado
- Cuando se levante el dedo:
 - introducir un retardo final
 - habilitar la IRQ del botón.

Autómatas. Implementación

- Ej.: Detección sentido contrario

- **MOORE**
- Entradas nivel muestreadas (síncronas)
- Salidas asíncronas

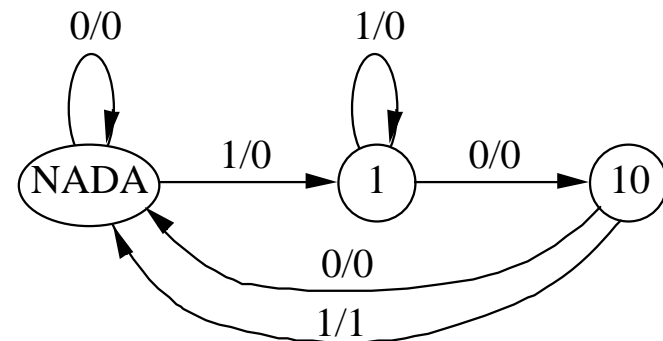


```
// Estado pertenece al conjunto de estados
Entrada = Leer_Entrada ();
switch (Estado)
{
    case C1 : Salida(NO_ALARMA) ;
        switch (Entrada) {
            case I01 : Estado = C3 ; break ;
            case I10 : Estado = C2 ; break ;
            default : }
        break ;
    case C2 : Salida (NO_ALARMA) ;
        if (Entrada == I00) Estado = C1 ;
        break ;
    case C3 : Salida (ALARMA) ;
        if (Entrada == I00) Estado = C1 ;
        break ;
}
```

Autómatas. Implementación

```
Espera_Sincronismo () ;
Entrada = Leer_Bit () ;
switch (Estado)
{
  case NADA : if (Entrada==0) {Salida=0; Estado=NADA;}
              else if (Entrada==1) {Salida=0; Estado=E1;}
              break ;
  case E1 :   if (Entrada==0) {Salida=0; Estado=E10;}
              else if (Entrada==1) {Salida=0; Estado=E1;}
              break ;
  case E10 :  if (Entrada==0) {Salida=0; Estado=NADA;}
              else if (Entrada==1) {Salida=1; Estado=E101;}
              break ;
  case E101 : if (Entrada==0) {Salida=0; Estado=NADA;}
              else if (Entrada==1) {Salida=0; Estado=E1;}
}
Genera (Salida) ;
```

Ej: reconocedor de cadenas: **MEALY**,



Podéis encontrar una forma más eficiente de implementar MSF en:
<http://johnsantic.com/comp/state.html>

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