Práctica 4

Sistema de E/S y dispositvos sencillos (LEDs, pulsadores, display 8-segmentos y temporizadores)

4.1. Objetivos de la práctica

En esta práctica vamos a estudiar el sistema de E/S de la placa S3CEV40 y el mecanismo de interrupciones para la gestión de la E/S. Para ello utilizaremos los dispositivos más sencillos de los que disponemos: unos LEDs, unos pulsadores, un display de ocho segmentos y unos temporizadores. Los principales objetivos de la práctica son:

- Conocer el sistema de E/S de la placa S3CEV40.
- Programar rutinas de tratamiento de interrupción.
- Aprender a programar algunos dispositivos básicos como LEDs, pulsadores, display 8-segmentos y temporizadores.

4.2. Sistema de E/S del ARM7TDMI

Como hemos visto en la práctica anterior, el procesador ARM7TDMI tiene la E/S ubicada en memoria. Esto quiere decir que tanto la memoria como la E/S comparten un mismo espacio de direcciones. Por consiguiente, cuando el procesador realiza una lectura o escritura en una dirección no es capaz de discriminar si ésta se realiza sobre la memoria o sobre un dispositivo de E/S, por lo que se necesita un dispositivo externo al procesador para distinguirlos.

Además, el procesador ARM7TDMI dispone de una línea de reset, que permite generar una excepción de RESET, y dos líneas, IRQ y FIQ, que permiten a un dispositivo externo conectado a ellas generar excepciones IRQ y FIQ respectivamente. Suele utilizarse el nombre de interrupciones para referirse a estas excepciones generadas externamente.

Todas las excepciones, incluyendo IRQ y FIQ, son autovectorizadas. Esto quiere decir que el procesador genera automáticamente un vector de interrupción, que es cargado en el contador de programa (ver la práctica anterior). En esta dirección de memoria deberemos poner un salto a la rutina de tratamiento de la excepción/interrupción.

Si conectamos varios dispositivos a las líneas IRQ y FIQ, la rutina de tratamiento ejecutada debe comprobar qué dispositivo ha producido la interrupción. Para hacerlo, debe leer los registros de estado de los dispositivos para comprobar si tienen una interrupción pendiente. A este proceso se le denomina *identificación por encuesta*. Asimismo, si hay varios dispositivos que hayan generado la interrupción simultáneamente, es la rutina la que tiene que decidir a cuál de ellos se atiende. Se dice en este caso que el arbitraje es software.

Finalmente, el registro de estado (CPSR) permite enmascarar las interrupciones externas. Si los bits F (6) e I (7) tienen el valor 1, las interrupciones por las líneas FIQ e IRQ quedan deshabilitadas. Es importante enmascarar estas interrupciones en la inicialización del sistema, cuando aún no lo hemos preparado para que éstas interrupciones sean tratadas por las rutinas apropiadas.

4.3. Sistema de E/S del S3C44B0X

La figura 4.1 representa un esquema de bloques del interior del sistema en chip S3C44B0X. Como vemos, el procesador ARM7TDMI se conecta a través del bus de sistema con una serie de controladores integrados dentro del mismo chip. Ya estudiamos en la práctica anterior el controlador de memoria y, como recordaremos, el espacio de direcciones asignado a cada uno de los dispositivos integrados dentro del chip está en el rango `0x01C0_0000-0x01F_FFFF`, correspondiente al banco cero (ver la práctica anterior). En esta práctica vamos a estudiar los siguientes dispositivos:

- El controlador de interrupciones (Interrupt CONT.).
- El controlador de pines de E/S multifunción (GPIO).
- Los temporizadores (PWM Timer).

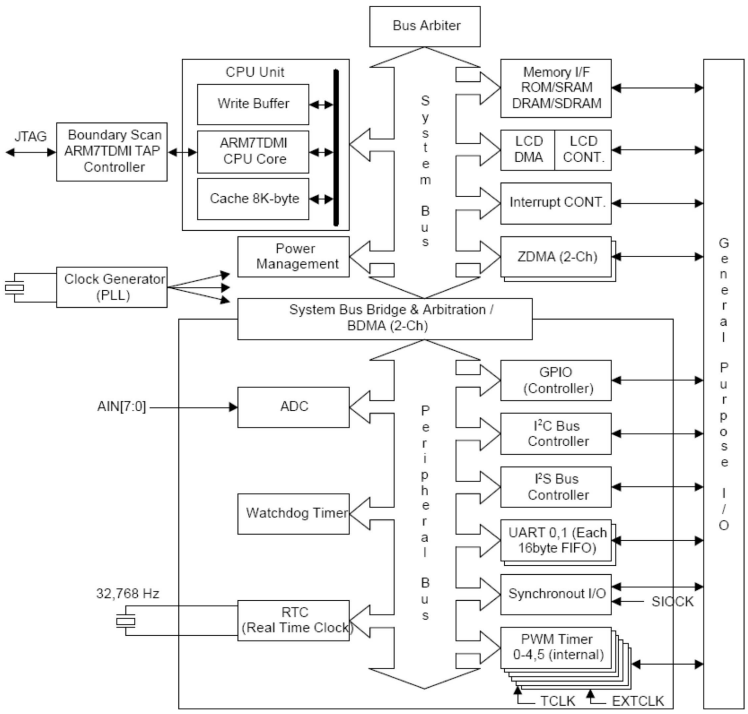


Figura 4.1: Esquema de bloques del interior del sistema en chip S3C44B0X

4.3.1. Controlador de interrupciones

El controlador de interrupciones permite convertir las dos líneas de interrupciones externas en 26 líneas independientes con distintas prioridades. Tiene una estructura jerárquica compuesta de un maestro y cuatro esclavos, como ilustra la figura 4.2. Como podemos ver, permite gestionar 30 fuentes de interrupciones aunque algunas de ellas (UERR0/1 y

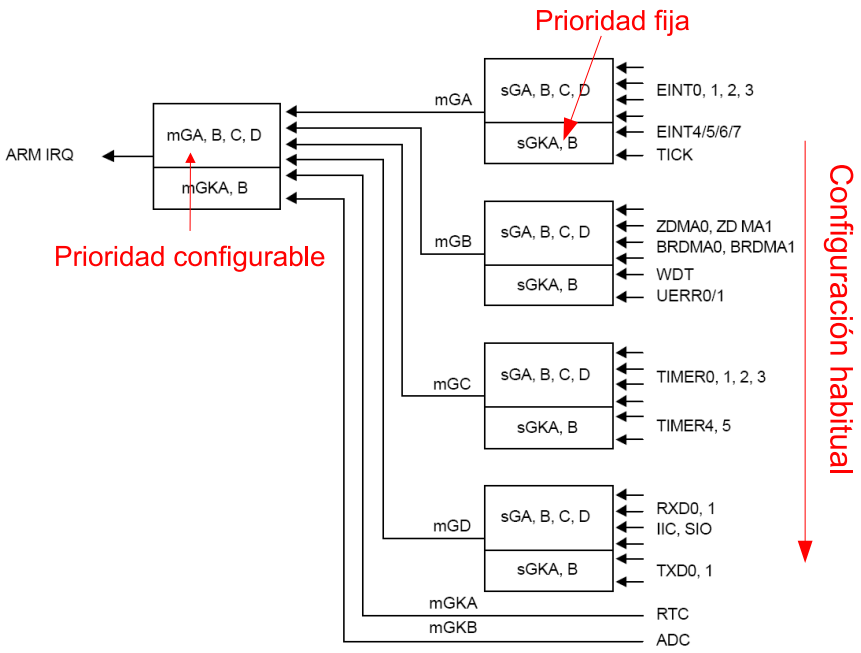


Figura 4.2: Esquema de bloques del controlador de interrupciones.

EINT4/5/6/7) comparten línea. Se puede configurar de forma que cada una de estas 26 líneas active o bien la línea IRQ o bien la línea FIQ.

El controlador puede configurarse de dos modos distintos:

- *No Vectorizado*. En este modo cuando una o más líneas externas se activan, el controlador activa la línea del procesador FIQ si alguna de la líneas tiene la tiene asociada y IRQ, en otro caso. Recordemos que entonces el procesador ARM7TDMI generará el vector correspondiente a la línea activa (FIQ o IRQ) y saltará a la rutina programada para esta línea.

- *Vectorizado*. En este modo cuando una o más líneas externas se activan:

  - Si alguna línea activa tiene asociada la interrupción FIQ, entonces se comporta como en el modo no vectorizado.
  - Si todas las líneas activas tiene asociada la interrupción IRQ, cuando el procesador ARM7TDMI vuelca en el bus el vector 0x18, el controlador de interrupciones inhibe al controlador de memoria e inserta en el bus de datos una instrucción de salto a una posición fija que depende de la línea externa activa. El efecto conseguido con esto es como si cada línea externa estuviese vectorizada. El vector sería la dirección destino del salto insertado en el bus. Los vectores generados para cada línea del controlador están recogidos en la tabla 4.1.

En el modo no vectorizado la rutina de tratamiento de cada línea (IRQ o FIQ) debe consultar los registros del controlador de interrupciones para averiguar cuál de las líneas externas es la que ha generado la interrupción. Es preciso resaltar que el proceso es mucho

Tabla 4.1: Vectores asociados a las 26 líneas de interrupción gestionadas por el controlador de interrupciones (externas al procesador). La primera columna indica el bit que corresponde a cada línea en los registros de configuración del controlador de interrupciones.

Bit	Línea	Vector
25	EINT0	0x20
24	EINT1	0x24
23	EINT2	0x28
22	EINT3	0x2c
21	EINT4/5/6/7	0x30
20	INT_TICK	0x34
19	INT_ZDMA0	0x40
18	INT_ZDMA1	0x44
17	INT_BDMA0	0x48
16	INT_BDMA1	0x4c
15	INT_WDT	0x50
14	INT_UERR0/1	0x54
13	INT_TIMER0	0x60
12	INT_TIMER1	0x64
11	INT_TIMER2	0x68
10	INT_TIMER3	0x6c
9	INT_TIMER4	0x70
8	INT_TIMER5	0x74
7	INT_URXD0	0x80
6	INT_URXD1	0x84
5	INT_IIC	0x88
4	INT_SIO	0x8c
3	INT_UTXD0	0x90
2	INT_UTXD1	0x94
1	INT_RTC	0xa0
0	INT_ADC	0xc0

más rápido que sin el controlador de interrupciones, ya que el procesador sólo tiene que consultar un registro del propio controlador, en lugar de consultar uno para cada dispositivo conectado a la línea (IRQ o FIQ). Sin embargo, si la línea externa activa es UERR0/1 o EINT4/5/6/7, la rutina deberá consultar además más registros para deducir la fuente exacta de la interrupción. Trataremos esto de nuevo en la Sección 4.3.2 cuando estudiemos los pines multifunción.

La configuración y el manejo del controlador de interrupciones se hace a través de una serie de registros. Muchos de ellos tienen un bit por línea de interrupción, siguiendo la asignación bit-línea de la tabla 4.1. La relación de registros del controlador de interrupciones que necesitamos para la práctica y su descripción es la siguiente:

INTCON *Interrupt Control Register* (`0x01E00000`). Registro de cuatro bits en el que sólo se utilizan tres de ellos:

- V (bit [2]) = 0, habilita las interrupciones vectorizadas

- I (bit [1]) = 0, habilita la línea IRQ
- F (bit [0]) = 0, habilita la línea FIQ

INTMOD *Interrupt Mode Register* (`0x01E00008`). Registro con un bit por línea: a 0 para que active IRQ o a 1 para que active FIQ.

INTPND *Interrupt Pending Register* (`0x01E00004`). Registro con un bit por línea: a 0 si no hay solicitud y a 1 si hay una solicitud. Este registro se actualiza incluso cuando la línea está enmascarada y no se traslada la interrupción al procesador.

INTMSK *Interrupt Mask Register* (`0x01E0000C`). Registro con 28 bits. El bit 27 está reservado. El bit 26 permite enmascarar todas las lineas (máscara global). El resto de los bits, uno por línea, cuando toman valor 0 habilitan la interrupción de la línea correspondiente y cuando toman valor 1 la enmascaran.

I_ISPR *IRQ Interrupt Service Pending Register* (`0x01E00020`). Registro con un bit por línea. Indica la interrupción que se está sirviendo actualmente. Aunque haya varias peticiones pendientes, cada una con su correspondiente bit del registro *INTPND* activo, sólo uno de los bits del registro *I_ISPR* estará activo (el más prioritario).

I_ISPC *IRQ Int. Service Pending Clear register* (`0x01E00024`). Registro con un bit por línea. Permite borrar el bit correspondiente del INTPND escribiendo 1 en la posición correspondiente. Indica al controlador que ha finalizado la rutina de servicio.

F_ISPC *FIQ Int. Service pending Clear register* (`0x01E0003C`). Igual que I_ISPC pero para la línea FIQ.

Para más información sobre estos registros es preciso consultar el manual de referencia del S3C44B0X [um-].

Cuando se emplean interrupciones vectorizadas, la prioridad entre las distintas líneas de interrupción se resuelve por hardware. Supongamos que se activan simultáneamente dos líneas A y B, la selección de la línea más prioritaria se realiza de acuerdo a las siguientes reglas:

- Si A está configurada para activar FIQ y B para activar IRQ, A es más prioritaria.
- Si A y B están en diferentes grupos maestro y la prioridad del grupo maestro de A es mayor que la prioridad del grupo maestro de B, A es más prioritaria que B. Los grupos maestros son los etiquetados como `mG*` en la figura 4.2.
- Si A y B están en el mismo grupo maestro y la fuente A tiene mayor prioridad que la fuente B, A es más prioritaria. Las fuentes son las etiquetadas como `SG*` en la figura 4.2.
- Las prioridades de las fuentes `sGA`, `sGB`, `sGC` y `sGD` son siempre mayores que las de `sGKA` y `sGKB`. Las prioridades entre las fuentes `sGA`, `sGB`, `sGC` y `sGD` son programables.
- Las prioridades de los grupos maestros `mGA`, `mGB`, `mGC` y `mGD` son siempre mayores que las de `mGKA` y `mGKB`. Las prioridades entre las fuentes `mGA`, `mGB`, `mGC` y `mGD` son programables.

La configuración del esquema de prioridades descrito puede configurarse a través de los registros *I_PSLV* y *I_PMST*. Para obtener una descripción precisa de ellos consultar el manual del S3C44B0X [um-]. Por defecto, el controlador de interrupciones emplea el esquema de prioridades mostrado en la figura 4.2 (EINT0 es la línea más prioritaria y ADC la menos), que se corresponde con el orden de los bits en los registros de configuración.

4.3.2. Controlador de E/S

El controlador de pines de E/S (GPIO) mantiene el control de 71 pines multifuncionales, es decir, cada uno de estos pines puede ser utilizado para más de una función distinta. El GPIO permite configurar la funcionalidad escogida para cada uno de estos pines.

Los 71 pines están divididos en 7 grupos, a los que denominamos puertos, que son:

- 2 grupos de 9 bits de E/S (Puertos E y F)
- 2 grupos de 8 bits de E/S (Puertos D y G)
- 1 grupo de 16 bits de E/S (Puerto C)
- 1 grupo de 11 bits de E/S (Puerto B)
- 1 grupo de 10 bits de E/S (Puerto A)

Cada puerto es gestionado mediante 2-4 registros. El número concreto depende del puerto. Al realizar un Reset estos registros se cargan a un valor seguro para no dañar ni el sistema ni el hardware que pueda haber conectado por estos pines. En esta práctica vamos a utilizar sólo los pines de E/S de los puertos B y G, que describiremos a continuación. Para obtener una descripción completa es preciso consultar el manual del S3C44B0X [um-].

Puerto B

El puerto B tiene 11 bits que admiten dos funcionalidades: podemos utilizar estos pines como pines de salida o podemos conectarlos a algunas señales generadas por el controlador de memoria. La configuración del puerto se realiza a través de dos registros del GPIO:

- *PCONB*, registro de control que selecciona la funcionalidad de cada uno de los pines. Su dirección es `0x01D20008` y su descripción está en la tabla 4.2.
- *PDATB*, registro de datos que permite escribir en el puerto. Su dirección es `0x01D2000C`.

En esta práctica utilizaremos el puerto B como puerto de escritura, para activar o desactivar dos leds conectados a dos patillas de dicho puerto.

Puerto G

El puerto G tiene siete bits que pueden configurarse de cuatro formas diferentes. La configuración del puerto se realiza mediante tres registros del GPIO:

Tabla 4.2: Configuración de los pines del puerto B.

PCONB	Bit	Descripción
PB10	[10]	0 = Output 1 = nGCS5
PB9	[9]	0 = Output 1 = nGCS4
PB8	[8]	0 = Output 1 = nGCS3
PB7	[7]	0 = Output 1 = nGCS2
PB6	[6]	0 = Output 1 = nGCS1
PB5	[5]	0 = Output 1 = nWBE3/nBE3/DQM3
PB4	[4]	0 = Output 1 = nWBE2/nBE2/DQM2
PB3	[3]	0 = Output 1 = nSRAS/nCAS3
PB2	[2]	0 = Output 1 = nSCAS/nCAS2
PB1	[1]	0 = Output 1 = SCLK
PB0	[0]	0 = Output 1 = SCKE

- *PCONG*, registro de control que permite seleccionar la funcionalidad de cada pin, tal y como se describe en la tabla 4.3. Su dirección es `0x01D20040`.
- *PDATG*, registro de datosque permite escribir o leer del puerto. Su dirección es `0x01D20044`.
- *PUPG*, regisrto de configuración que permite activar o no una resistencia de *pull-up* por cada bit. Su dirección en `0x01D20048`.

Tabla 4.3: Configuración de los pines del puerto G.

PCONG	Bits	Descripción
PG7	15:14	00 = Input 01 = Output 10 = IISLRCK 11 = EINT7
PG6	13:12	00 = Input 01 = Output 10 = IISDO 11 = EINT6
PG5	11:10	00 = Input 01 = Output 10 = IISDI 11 = EINT5
PG4	9:8	00 = Input 01 = Output 10 = IISCLK 11 = EINT4
PG3	7:6	00 = Input 01 = Output 10 = nRTS0 11 = EINT3
PG2	5:4	00 = Input 01 = Output 10 = nCTS0 11 = EINT2
PG1	3:2	00 = Input 01 = Output 10 = VD5 11 = EINT1
PG0	1:0	00 = Input 01 = Output 10 = VD4 11 = EINT0

Como vemos, podemos configurar los pines del puerto para que activen las líneas `EINT*` del controlador de interrupciones. Esto permite que algún hardware externo conectado a estos pines pueda generar una interrupción por estas líneas. La interrupción puede generarse

cuando se detecte un nivel de voltaje en el pin o cuando se detecte un determinado flanco. Esto se configura en el registro del GPIO *EXTINT*, cuya dirección es `0x01D20050`, según indica la tabla 4.4. Además, el controlador GPIO permite descomponer en cuatro la línea EXTINT4/5/6/7 del controlador de interrupciones. La línea EXTINT4/5/6/7 se activa cuando alguna de estas cuatro líneas nuevas genera una interrupción (se hace una OR). El registro *EXTINTPND*, como describe la tabla 4.5, permite saber cuál de las cuatro líneas es la que ha generado la interrupción. Su dirección es `0x01D20054`. Alternativamente, podemos utilizar los bits del puerto G como pines de entrada o de salida o conectarlos a algunas señales internas del sistema.

Tabla 4.4: Registro *EXTINT*.

EXTINT	Bit	Descripción
EINT7	[30:28]	Establece el método de señalización de EINT7. 000 = Interrupción por nivel bajo 001 = interrupción por nivel alto 01x = Disparado por flanco de bajada 10x = Disparado por flanco de subida 11x = Disparado por ambos flancos
EINT6	[26:24]	Establece el método de señalización de EINT6. 000 = Interrupción por nivel bajo 001 = interrupción por nivel alto 01x = Disparado por flanco de bajada 10x = Disparado por flanco de subida 11x = Disparado por ambos flancos
EINT5	[22:20]	Establece el método de señalización de EINT5. 000 = Interrupción por nivel bajo 001 = interrupción por nivel alto 01x = Disparado por flanco de bajada 10x = Disparado por flanco de subida 11x = Disparado por ambos flancos
EINT4	[18:16]	Establece el método de señalización de EINT4. 000 = Interrupción por nivel bajo 001 = interrupción por nivel alto 01x = Disparado por flanco de bajada 10x = Disparado por flanco de subida 11x = Disparado por ambos flancos
EINT3	[14:12]	Establece el método de señalización de EINT3. 000 = Interrupción por nivel bajo 001 = interrupción por nivel alto 01x = Disparado por flanco de bajada 10x = Disparado por flanco de subida 11x = Disparado por ambos flancos
EINT2	[10:8]	Establece el método de señalización de EINT2. 000 = Interrupción por nivel bajo 001 = interrupción por nivel alto 01x = Disparado por flanco de bajada 10x = Disparado por flanco de subida 11x = Disparado por ambos flancos
EINT1	[6:4]	Establece el método de señalización de EINT1. 000 = Interrupción por nivel bajo 001 = interrupción por nivel alto 01x = Disparado por flanco de bajada 10x = Disparado por flanco de subida 11x = Disparado por ambos flancos
EINT0	[2:0]	Establece el método de señalización de EINT0. 000 = Interrupción por nivel bajo 001 = interrupción por nivel alto 01x = Disparado por flanco de bajada 10x = Disparado por flanco de subida 11x = Disparado por ambos flancos

4.3.3. Temporizadores

El S3C44B0X tiene 6 temporizadores de 16 bits, conectados según indica la figura 4.3. Cada uno puede ser configurado para operar por interrupciones o por DMA. Los

Tabla 4.5: Significado de los bits del registro *EXTINTPND*.

EXTINTPND	Bit	Descripción
EXTINTPND3	[3]	Si EINT7 se activa, EXINTPND3 se pone a 1, y INTPND[21] se pone a 1.
EXTINTPND2	[2]	Si EINT6 se activa, EXINTPND2 se pone a 1, y INTPND[21] se pone a 1.
EXTINTPND1	[1]	Si EINT5 se activa, EXINTPND1 se pone a 1, y INTPND[21] se pone a 1.
EXTINTPND0	[0]	Si EINT4 se activa, EXINTPND0 se pone a 1, y INTPND[21] se pone a 1.

temporizadores 0, 1, 2, 3 y 4 tienen asociada la capacidad de generar una onda cuadrada de la frecuencia con la que haya sido programado el temporizador, modulada por ancho de pulsos (PWM, pulse width modulation), son las señales TOUT* de la figura 4.3. Estas señales pueden sacarse por algún pin para transmitir una señal por modulación de ancho de pulsos, controlar un motor, etc. Esta funcionalidad no está disponible en el temporizador 5, que sólo puede utilizarse para temporización interna y no tiene pin de salida.

Funcionamiento de los temporizadores

Los temporizadores son contadores descendentes que pueden ser inicializados con un determinado valor. Una vez configurados e inicializados, cada ciclo de reloj interno del procesador se decrementan. Cuando llegan a 0 generan una interrupción que podemos utilizar para realizar algunas tareas de forma periódica.

Como podemos ver en la figura 4.3, cada par de temporizadores (0-1, 2-3, 4-5) comparten un módulo de pre-escalado y un divisor de frecuencia. El divisor de los 4 primeros temporizadores tiene cinco señales divididas distintas: $1/2, 1/4, 1/8, 1/16$ y $1/32$. Los temporizadores 4 y 5 tienen un divisor con cuatro señales de reloj divididas: $1/2, 1/4, 1/8$ y $1/16$, y una entrada adicional TCLK/EXTCLK, que sirve para realizar la cuenta de una señal distinta que la señal de reloj (por ejemplo para contar pulsos externos al sistema, obtenidos de un pin). Como vemos, los divisores se alimentan con la salida de los módulos de pre-escalado con lo que la frecuencia de cuenta se construye con un proceso de división en dos etapas, como detallaremos posteriormente.

Cada temporizador (excepto el 5) tiene un par de registros asociados, TCNTn y TCMPn. El registro de cuenta (TCNTn) es inicializado con la carga de un valor y comienza la cuenta descendente cuando se habilita. El registro de comparación (TCMPn) se inicializa con un valor que es comparado con el valor del registro de cuenta correspondiente, activando una señal. Esta señal se utiliza para generar los cambios de la señal modulada por anchura de pulsos. Al comienzo de cada cuenta la señal PWM tiene un valor 0, cuando el contador alcanza el valor del registro de comparación la señal PWM toma el valor 1. De esta forma podemos controlar exactamente el número de ciclos que debe estar a 0 y a 1.

Los temporizadores pueden funcionar en dos modos: *auto-reload* y *one-shot*. En modo *auto-reload* cuando el temporizador llega a cero, su valor inicial se vuelve a cargar automáticamente y se comienza una nueva cuenta atrás. Esto sirve para generar de forma muy precisa interrupciones periódicas. En el modo *one-shot*, cuando el contador llega a cero se para la cuenta.

La figura 4.4 representa un diagrama temporal del funcionamiento habitual de los temporizadores. Se utiliza un sistema de doble buffer que permite al usuario modificar el valor de recarga del temporizador mientras está contando, sin alterar o parar el con-

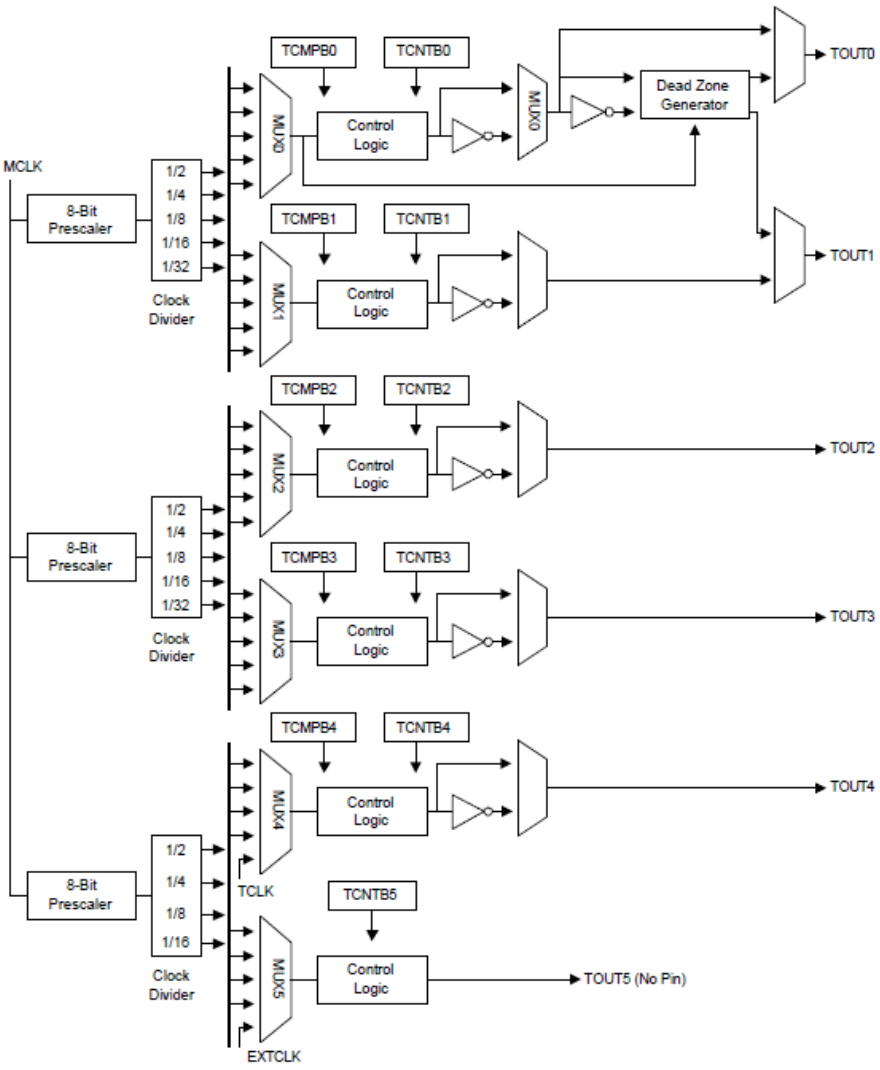


Figura 4.3: Esquema de conexión de los temporizadores del S3C44B0X.

tador. Normalmente el temporizador se inicializa escribiendo un valor en el registro de cuenta (TCNTn) y activando el *manual update bit*, como veremos en el siguiente apartado. Mientras el contador está en marcha, se modifica el valor del registro de buffer asociado (TCNTBn) y el valor de la cuenta no se verá alterado. Este nuevo valor se utilizará para recargar TCNTn cuando la cuenta llegue a cero (si está en modo *auto-reload*). Lo mismo sucede con el registro de comparación (TCMPn), que será recargado a partir del registro de buffer (TCMPBn).

El valor actual de cuenta puede ser leído del registro de observación (TCNTOn). Si se lee TCNTBn no se obtiene el valor actual de la cuenta sino el valor de recarga.

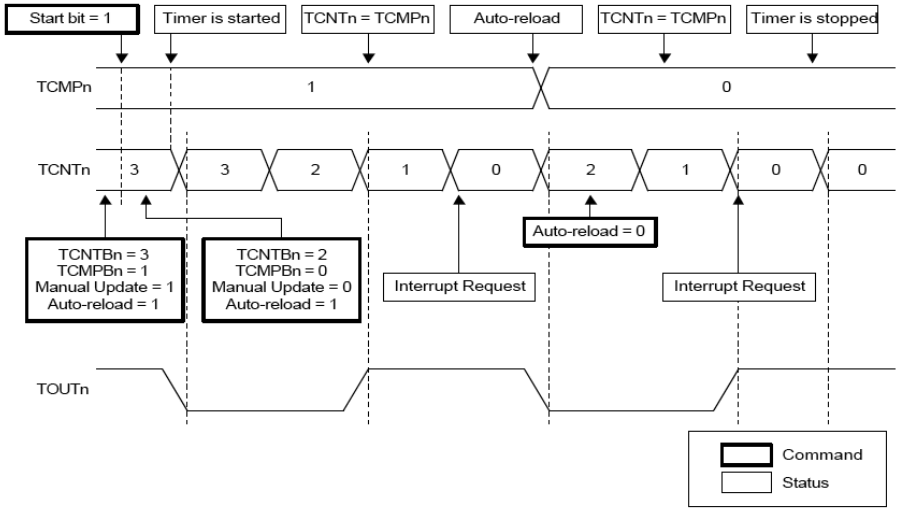


Figura 4.4: Diagrama temporal del funcionamiento de los temporizadores del S3C44B0X.

Inicialización de los temporizadores

Como el valor del registro TCMPBn se copia en el registro de cuenta TCNTn sólo cuando TCNTn alcanza cero, el temporizador no puede ser inicializado simplemente escribiendo en TCMPBn. Para que el valor que escribimos en TCMPBn pase directamente a TCNTn debemos activar el *manual update bit*. La secuencia de inicialización del temporizador sería por tanto:

1. Escribir los valores iniciales de cuenta y comparación en TCNTBn y TCMPBn respectivamente.
2. Activar el *manual update bit* del temporizador.
3. Activar el bit de comienzo (*start bit*) al mismo tiempo que se desactiva el *manual update bit*.

Configuración de los temporizadores

La señal de reloj efectiva para los temporizadores es:

$$F = MCLK/\left((\text{valor de pre-escalado} + 1)(\text{valor del divisor})\right)$$

donde, MCLK es la señal interna de reloj, el valor de pre-escalado está en el intervalo $0-255$ y el valor del divisor: $2, 4, 8, 16, 32$. El valor de pre-escalado para los temporizadores se configura en el registro TCFG0 (`0x01D50000`), descrito en la tabla 4.6.

El valor del divisor se configura con el registro TCFG1 (`0x01D50004`), así como la asignación de un temporizador al canal de petición del DMA. La descripción del registro aparece en la tabla 4.7.

El control de los temporizadores (puesta en marcha, parada, actualización, modo auto-reload, etc) se realiza mediante el registro TCON (`0x01D50008`). Su funcionalidad está descrita en la tabla 4.8.

Tabla 4.6: Registro *TCFG0*.

Función	Bits	Descripción
Longitud de la zona muerta	[31:24]	Estos 8 bits determinan la zona muerta. La unidad de tiempo de la zona muerta es la misma que la del temporizador 0.
Pre-escalado 2	[23:16]	Estos ocho bits determinan el factor de pre-escalado de los temporizadores 4 y 5.
Pre-escalado 1	[15:8]	Estos ocho bits determinan el factor de pre-escalado de los temporizadores 2 y 3.
Pre-escalado 0	[7:0]	Estos ocho bits determinan el factor de pre-escalado de los temporizadores 0 y 1.

Finalmente, para manejar los temporizadores debemos utilizar los registros de buffer, tanto de cuenta como de comparación, y el registro de observación. Sus direcciones están en la tabla 4.9.

Para ampliar la información sobre los temporizadores es preciso consultar el manual del S3c44B0X [um-].

4.4. Dispositivos básicos de la placa S3CEV40

La figura 4.5 muestra la ubicación de los dispositivos conectados al sistema en chip S3C44BOX en la placa S3CEV40 utilizada en el laboratorio. La mayor parte de estos dispositivos externos tiene su espacio de direcciones ubicado en el banco uno, en el rango `0x0200_0000-0x03FF_FFFF`. La tabla 4.10 detalla el intervalo asignado a cada dispositivo. Cuando el procesador accede a alguno de estos dispositivos el controlador de memoria genera la señal de habilitación del banco correspondiente (nGCS1 o NGCS3). Estas señales se utilizan fuera del chip S3C44BOX para habilitar el dispositivo apropiado. La activación de las señales de Chip Select de los dispositivos ubicados en el banco 1 se realiza mediante un decodificador 3 a 8, el chip 74LV138, que es habilitado con la señal nGCS1 según el esquema de la figura 4.6.

4.4.1. LEDs y pulsadores

En la placa S3CEV40 hay dos pulsadores y dos leds conectados al sistema S3C44BOX como indica la figura 4.7. Los dos pulsadores se conectan a los pines 6 y 7 del puerto G, que podemos seleccionar como EXINT6 y EXINT7. Si las resistencias de *pull-up* de dichos pines se activan (ver 4.3.2) podemos generar una interrupción por flanco de bajada cada vez que se pulse uno de los pulsadores (para ello es necesario configurar además el registro EXTINT de forma adecuada). Los leds en cambio se conectan a los pines 9 y 10 del puerto B. Como vemos, su ánodo se conecta a la alimentación (Vdd) por lo que para iluminar estos leds tendríamos que configurar los pines 9 y 10 como sólo salida y escribir en ellos un cero.

Tabla 4.7: Registro *TCFG1*.

Función	Bits	Descripción
modo DMA	[27:24]	Select DMA request channel 0000 = No seleccionado 0001 = Temporizador0 0010 = Temporizador1 0011 = Temporizador2 0100 = Temporizador3 0101 = Temporizador4 0110 = Temporizador5 0111 = Reservado
MUX 5	[23:20]	Select MUX input for PWM Timer5. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = EXTCLK
MUX 4	[19:16]	Select MUX input for PWM Timer4. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = TCLK
MUX 3	[15:12]	Select MUX input for PWM Timer3. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = 1/32
MUX 2	[11:8]	Select MUX input for PWM Timer2. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = 1/32
MUX 1	[7:4]	Select MUX input for PWM Timer1. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = 1/32
MUX 0	[3:0]	Select MUX input for PWM Timer0. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = 1/32

4.4.2. Display 8-segmentos

En la placa S3CEV40 también tenemos un display de 8 leds (7 segmentos para conformar cualquier dígito hexadecimal más un punto decimal). Este dispositivo se habilita cuando se accede a alguna dirección en el rango `0x0214_0000 - 0x0217_FFFF` por medio de la señal CS6, generada como hemos visto por el circuito de la figura 4.5. Los leds del display se conectan al bus de datos del procesador a través de un latch triestado, el chip 74LS543, según ilustra la figura 4.8. Este latch almacenará el valor escrito hasta que se vuelva a escribir un valor distinto. El display es de ánodo común, esto quiere decir que para encender los leds hay que ponerlos a cero.

4.5. Estructura general de un proyecto de prácticas

En las prácticas anteriores hemos construido todo el proyecto desde cero, aunque fuese de manera guiada. Toda la funcionalidad ha sido proporcionada por nosotros. A partir de ahora vamos a utilizar un esquema común para los proyectos, donde parte de la funcionalidad vendrá suministrada por una serie de ficheros de biblioteca que tendremos que aprender a utilizar correctamente. Estos ficheros jugarán el papel que juegan en un entorno estándar el sistema operativo, el cargador y la biblioteca estándar de C. Los ficheros

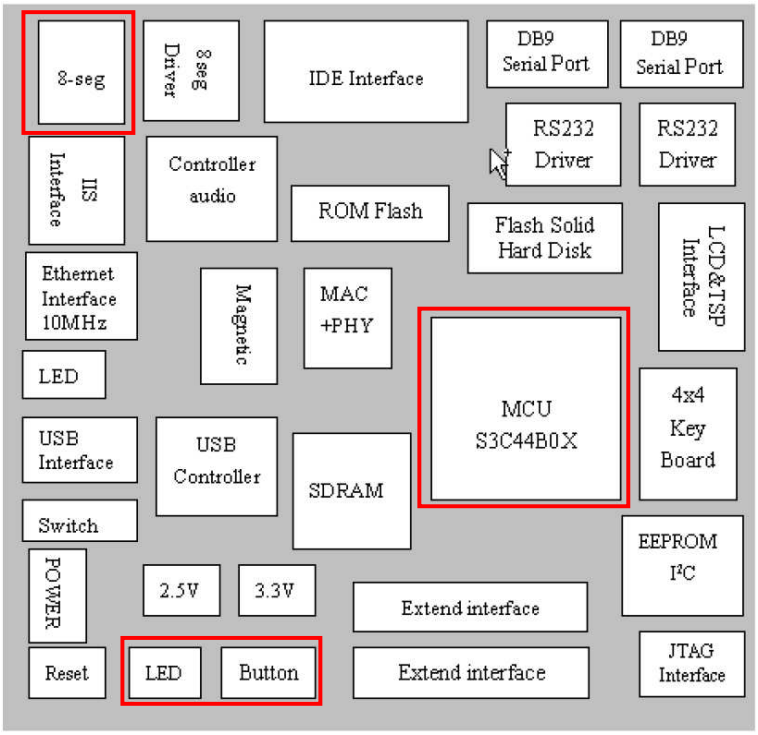


Figura 4.5: Ubicación de los dispositivos básicos conectados al sistema en chip S3C44BOX en la placa S3CEV40.

comunes que utilizaremos serán:

44b.h y option.h Ficheros de cabecera que contienen macros con las direcciones de los distintos dispositivos del sistema.

44blib.h Fichero de cabecera de una librería de sistema básica. Contiene las cabeceras de las funciones proporcionadas por la librería. Algunas funciones son: malloc, free, sys_init, Delay, DelayMs, etc.

44binit.s Fichero de inicialización del sistema. Es un fichero similar al construido en la práctica 3, que se encarga de inicializar el sistema, configurando las pilas de los distintos modos y rellenando la tabla de rutinas de tratamiento de interrupciones. Posteriormente salta a la función Main.

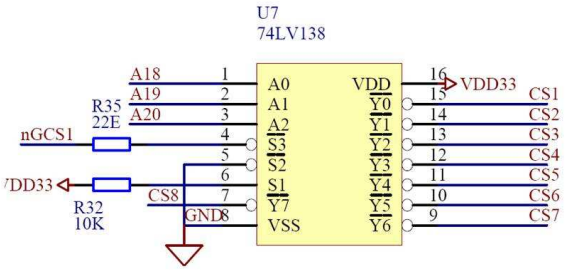
44blib.c Es nuestra biblioteca de sistema. Ofrece funcionalidades básicas como gestión de heap (gestionado en realidad como una pila), inicialización estándar de dispositivos o actuación sobre algunos.

ev40boot.cs Script de inicialización de la placa (como en la práctica 3).

ram_ice.ld Fichero de configuración del ld, como en las prácticas anteriores.

En el fichero 44b.h las definiciones de macros son del estilo:

```
#define rBANKCON0   (*(volatile unsigned *)0x1c80000)
```



A20	A19	A18	CS	Módulo
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

Figura 4.6: Generación de las señales de Chip Select para los dispositivos ubicados en el rango de direcciones del banco 1.

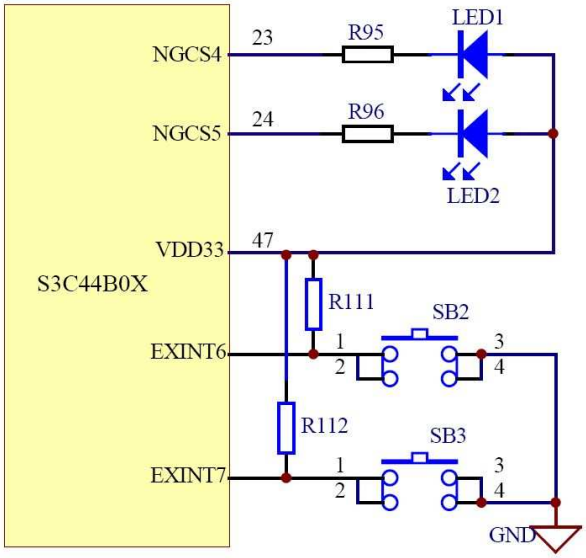


Figura 4.7: Esquema de conexión de pulsadores y leds a los pines del puerto B en la placa Embest S3CEV40.

donde se convierte la dirección a un puntero que se desreferencia. Se utiliza el modificador volatile para que siempre que se vaya a utilizar el valor se vuelva a leer, y no se asuma que, por haberlo leído antes y tener el valor leído almacenado en un registro, el valor de esa posición siga siendo el mismo. Estas macros pueden utilizarse en un programa C para leer o escribir en estas posiciones:

```
rBANKCON0 =  ... // escritura de un valor
mivar = rBANKCON0;  // lectura de la dirección (dispositivo)
```

4.6. Desarrollo de la Práctica

En esta práctica vamos a dividir el trabajo en tres partes, de las cuales las dos primeras estarán parcialmente guiadas. En la primera nos familiarizaremos con el uso de los LEDs,

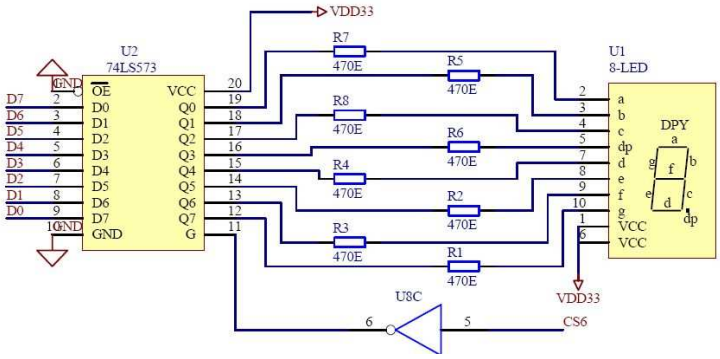


Figura 4.8: Esquema de conexión del display de 7 segmentos en la placa S3CEV40.

la gestión de interrupciones y la programación de los temporizadores. En la segunda, tomaremos contacto con el display y los pulsadores. Finalmente, en la última parte aplicaremos lo aprendido combinando diversos dispositivos.

4.6.1. Primera parte

En esta primera parte completaremos un programa que ilumina los LEDs de manera alterna a intervalos regulados mediante interrupciones del temporizador 0.

1. Creamos un workspace nuevo y le añadimos la carpeta *common*.

2. Añadimos los ficheros `44binit.s`, `44blib.c`, `ev40boot.cs` y `ram_ice.ld` a esta carpeta.

3. Añadimos los ficheros `44b.h`, `44blib.h` y `option.h` a la carpeta *Project Header Files*.

4. Creamos un fichero `main.c` en el que añadimos la función `Main()`. El código completo viene a continuación:

```
/*--- ficheros de cabecera ---*/
#include "44blib.h"
#include "44b.h"
#include "stdio.h"

/*--- variables globales ---*/
extern int switch_leds;

/*--- funciones externas ---*/
extern void leds_off();
extern void led1_on();
extern void leds_swicth();
extern void timer_init();

/*--- declaracion de funciones ---*/
```

```
void Main(void);

/*--- codigo de funciones ---*/
void Main(void)
{
    char dir = 0;

    /* Inicializar controladores */
    sys_init();        // Inicializacion de la placa, interrupciones y puertos
    timer_init();    // Inicializacion del temporizador

    /* Establecer valor inicial de los leds */
    leds_off();
    led1_on();

    while (1)
    {
        /* Cambiar los leds con cada interrupcion del temporizador */
        if (switch_leds == 1)
        {
            leds_swicth();
            switch_leds = 0;
        }
    }
}
```

5. Creamos un fichero `led.c` en el que añadimos las funciones para manejo de los LEDs. El código parcialmente completo viene a continuación:

```
/*--- ficheros de cabecera ---*/
#include "44b.h"
#include "44blib.h"

/*--- variables globales ---*/
int led_state;                                    // estado del LED

/*--- declaracion de funciones ---*/
void leds_on();                                  // todos los leds encendidos
void leds_off();                                 // todos los leds apagados
void led1_on();                             // led 1 encendido
void led1_off();                                // led 1 apagado
void led2_on();                                  // led 2 encendido
void led2_off();                                // led 2 apagado
void leds_swicth();                            // invierte el valor de los leds
void Led_Display(int LedStatus);          // control directo del LED

/*--- codigo de las funciones ---*/
```

```
void leds_on()
{
Led_Display(0x3);
}

void leds_off()
{
Led_Display(0x0);
}

void led1_on()
{
led_state = led_state | 0x1;
Led_Display(led_state);
}

void led1_off()
{
led_state = led_state & 0xfe;
Led_Display(led_state);
}

void led2_on()
{
led_state = led_state | 0x2;
Led_Display(led_state);
}

void led2_off()
{
led_state = led_state & 0xfd;
Led_Display(led_state);
}

void leds_swicth ()
{
led_state ^= 0x03;
Led_Display(led_state);
}

void Led_Display(int LedStatus)
{
        led_state = LedStatus;

        if((LedStatus&0x01)==0x01)
                // poner a 0 el bit 9 del registro de datos del puerto B
        else
```

```
                // poner a 1 el bit 9 del registro de datos del puerto B

        if((LedStatus&0x02)==0x02)
                // poner a 0 el bit 10 del registro de datos del puerto B
        else
                // poner a 1 el bit 10 del registro de datos del puerto B
}
```

6. Creamos un fichero `timer.c` en el que añadimos las funciones para manejo del temporizador. El código parcialmente completo viene a continuación:

```
/*--- ficheros de cabecera ---*/
#include "44b.h"
#include "44blib.h"

/*--- variables globales ---*/
int switch_leds = 0;

/*--- declaracion de funciones ---*/
void timer_ISR(void) __attribute__ ((interrupt ("IRQ")));
void timer_init(void);

/*--- codigo de las funciones ---*/
void timer_ISR(void)
{
        switch_leds = 1;

        // borrar bit en I_ISPC
}

void timer_init(void)
{
        /* Configuracion del controlador de interrupciones */
        // Configurar las linas como de tipo IRQ
        // Habilitar int. vectorizadas y la linea IRQ (FIQ no)
        // Emascarar todas las lineas excepto Timer0 y el bit global

        /* Establece la rutina de servicio para TIMER0 */
        pISR_TIMER0=(unsigned)timer_ISR;

        /* Configurar el Timer0 (el resto de los timers se dejan a la configuración por def
        // pre-escalado = 255
        // divisor = 1/2
        // TCNT0 = 65535
        // TCMPB0 = 12800
        // establecer manual\_update e inverter\_on */
        // inicial timer y activar modo auto-reload
}
```

7. Configuramos el proyecto del siguiente modo:

   - *Processor:* Seleccionar las opciones de la figura 4.9.

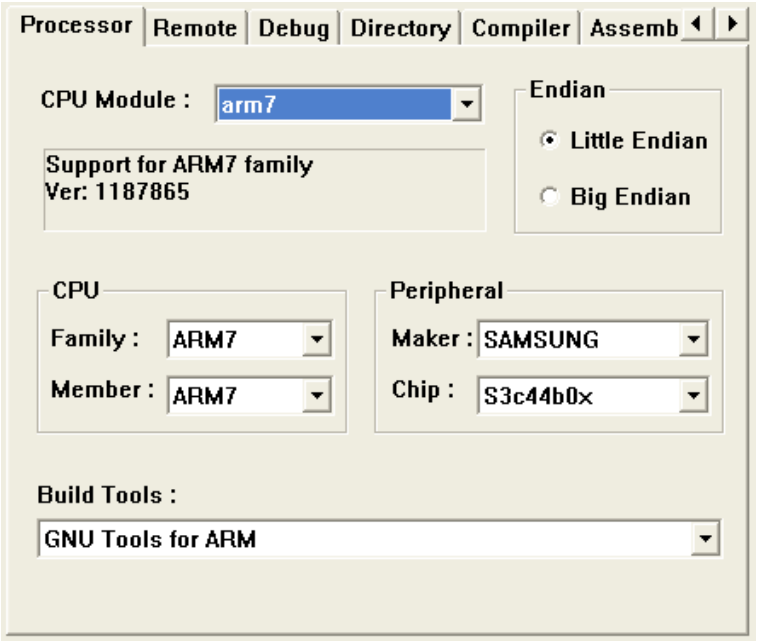


Figura 4.9: Configuración del procesador y periféricos.

   - *Remote:* Configurar como en las prácticas anteriores.
   - *Debug:* Configurar de manera análoga a la primera parte de las práctica anterior, empleando como script el fichero `ev40boot.cs` suministrado con la práctica. Recordad que es necesario definir el fichero de símbolos (.elf), la dirección de descarga (`0x0c000000`) así como el fichero de descarga (.elf).
   - *Compiler, Assembler:* Configurar de manera análoga a las prácticas anteriores.
   - *Linker:* Seleccionar como script de enlazado el fichero `ram_ice.ld`. En *Category/Add Library Searching Path* añadir los siguientes directorios, ambos ubicados en `C:\software\electronica\embest\IDEproEdu2004\`:
     - `Build\xgcc-arm-elf\arm-elf\lib`
     - `Build\xgcc-arm-elf\lib\gcc-lib\arm-elf\3.0.2`

     Añadir al final de *Link Options:* "`-lc -lgcc`"

8. Completar el código, compilar el proyecto y subirlo a la placa. Comprobar que el programa funciona correctamente.

9. Dentro del depurador, en la venta de registros, seleccionar la pestaña *Peripheral*. Pulsar con el botón derecho del ratón sobre `--- I/O` y seleccionar `Refresh S3\_C44B0\_IO`.

10. Contestad a las siguientes preguntas:

    - ¿Cuál es la configuración del puerto B antes ejecutar `sys_init()`? ¿Y justo después?

- ¿Cuál es la configuración del puerto G antes de ejecutar `sys_init()`? ¿Y justo despúes?
- ¿Qué efecto producen los siguientes comandos del depurador?:
  - `memwrite 0x1d20008 0x000001cf`
  - `memwrite 0x1d2000c 0x000001ff`

4.6.2. Segunda parte

En esta segunda parte, completaremos un programa que ilumina un segmento del borde del display y que responde a las pulsadores del siguiente modo:

- **Pulsador SB2:** cambia al siguiente segmento siguiendo el sentido de las agujas del reloj (a, b, c, d, e, g, a, ...).
- **Pulsador SB3:** cambia al siguiente segmento siguiendo el sentido contrario a las agujas del reloj (a, g, e, d, c, b, a, ...).

1. Creamos un workspace nuevo y le añadimos la carpeta `common`.
2. Añadimos los ficheros `44binit.s`, `44blib.c`, `ev40boot.cs` y `ram_ice.ld` a esta carpeta.
3. Añadimos los ficheros `44b.h`, `44blib.h` y `option.h` a la carpeta *Project Header Files*.
4. Creamos un fichero `main.c` en el que añadimos la función `Main()`. El código completo viene a continuación:

```
/*--- ficheros de cabecera ---*/
#include "44blib.h"
#include "44b.h"

/*--- variables globales ---*/
int symbol = 0;
extern int update;

/*--- funciones externas ---*/
extern void Eint4567_init(void);
extern void D8Led_Symbol(int value);

/*--- declaracion de funciones ---*/
void Main(void);

/*--- codigo de las funciones ---*/
void Main(void)
{
        sys_init();     // Inicializacion de la placa
        Eint4567_init();      // Inicializacion de EINT6/7 y del puerto G
```

```
        D8Led_symbol(symbol);   // muestra el simbolo inicial en el display

        while(1) {
            /* Actualizar el simbolo con cada pulsacion */
            if (update>0)
            {
                symbol = (symbol + update) \% 6;  // actualiza el simbolo segun proced
                D8Led_symbol(symbol);      // muestra el sÌmbolo en el display
                Delay(1000);         // espera 100ms  para evitar problemas de rebotes
                update = 0;    // restaura la variable update
        }
}
}
```

5. Creamos un fichero `8led.c` en el que añadimos las funciones para manejo del display de 8-segmentos. El código parcialmente completo viene a continuación:

```
/*--- ficheros de cabecera ---*/
#include "44b.h"
#include "44blib.h"

/*--- definicion de macros ---*/
/* Mapa de bits de cada segmento
  (valor que se debe escribir para que se ilumine el segmento) */
#define SEGMENT_A /* valor */
#define SEGMENT_B /* valor */
#define SEGMENT_C /* valor */
#define SEGMENT_D /* valor */
#define SEGMENT_E /* valor */
#define SEGMENT_F /* valor */
#define SEGMENT_G /* valor */
#define SEGMENT_P /* valor */

/*--- variables globales ---*/
/* tabla de segmentos */
int Symbol[] = { SEGMENT_A, SEGMENT_B, SEGMENT_C,
SEGMENT_D, SEGMENT_E, SEGMENT_G};

/*--- declaracion de funciones ---*/
void D8Led_init(void);
void D8Led_symbol(int value);

/*--- codigo de las funciones ---*/
void D8Led_init(void)
{
    /* Estado inicial del display con todos los segmentos iluminados
       (buscar en los ficheros de cabera la direccion implicada) */
```

```
}

void D8Led_symbol(int value)
{
    // muestra Symbol[value] en el display
}
```

6. Creamos un fichero `button.c` en el que añadimos las funciones para manejo de las interrupciones EINT6/7 (pulsadores). El código parcialmente completo viene a continuación:

```
/*--- ficheros de cabecera ---*/
#include "44blib.h"
#include "44b.h"
#include "def.h"

/*--- variables globales ---*/
unsigned int update = 0;

/*--- declaracion de funciones ---*/
void Eint4567_ISR(void) __attribute__ ((interrupt ("IRQ")));
void Eint4567_init(void);

/*--- codigo de funciones ---*/
void Eint4567_init(void)
{
       /* Configuracion del controlador de interrupciones */
       // Borra INTPND escribiendo 1s en I_ISPC
       // Borra EXTINTPND escribiendo 1s en el propio registro
       // Configura las linas como de tipo IRQ mediante INTMOD
       // Habilita int. vectorizadas y la linea IRQ (FIQ no) mediante INTCON
       // Emascara todas las lineas excepto eint4567 y el bit global (INTMSK)

       /* Establecer la rutina de servicio para Eint4567 */

       /* Configuracion del puerto G */
       // Establece la funcion de los pines (EINT0-7) mediante
       // Habilita el "pull up" del puerto
       // Configura las lineas de int. como de flanco de bajada mediante EXTINT

       /* Por precaucion, se vuelven a borrar los bits de INTPND y EXTINTPND */
}

void Eint4567_ISR(void)
{
       /* Idenficiar la interrupcion */
       int which_int = rEXTINTPND;
```

```
        /* Reflejar en update el sentido de la actualizacion
           Nota: para solucionar los problemas de rebotes en el pulsador
              solo se modifica update cuando se encuentra a cero */
        if (!update) {
            switch (which_int) {
            case 4:
                 update = 1; // actualizar al simbolo siguiente
                 break;
            case 8:
                 update = 5; // actualizar al simbolo anterior
                 break;
            default:
                 break;
            }
        }

        /* Finalizar ISR */
        // borra los bits en EXTINTPND
        // borra el bit pendiente en INTPND
}
```

7. Configurar el proyecto como en el apartado anterior.

8. Completar el código, compilar el proyecto y subirlo a la placa. Comprobar que el programa funciona correctamente.

9. **Ejercicio:** Codificar la función `D8Led_symbol()` en ensamblador.

4.6.3. Tercera parte

Empleando los códigos de las dos partes anteriores, es necesario implementar un programa que ilumine los segmentos del borde del display a intervalos de tiempo fijos, siguiendo el sentido de las agujas del reloj (a, b, c, d, e, g, a, ...). Los intervalos se programaran mediante una interrupción periódica del temporizador 0 y los pulsadores SB2 y SB3 se emplearán para variar la velocidad del cambio de segmento (actuando sobre la configuración del temporizador).

Tabla 4.8: Registro *TCON* (0x01D50008).

Función	**Bits**	**Descripción**
Timer 5 auto reload on/off	[26]	Este bit determina el auto-reload para el Temporizador 5. 0 = One-shot 1 = Interval mode (auto reload)
Timer 5 manual update	[25]	Este bit determina el *maual update* del Temporizador 5. 0 = No operation 1 = Update TCNTB5
Timer 5 start/stop	[24]	Este bit determina el *start/stop* del Temporizador 5. 0 = Stop 1 = Start for Timer 5
Timer 4 auto reload on/off	[23]	Este bit determina el auto-reload para el Temporizador 4. 0 = One-shot 1 = Interval mode (auto reload)
Timer 4 output inverter on/off	[22]	This bit determines output inverter on/off for Timer4. 0 = Inverter off 1 = Inverter on for TOUT4
Timer 4 manual update	[21]	Este bit determina el *maual update* del Temporizador 4. 0 = No operation 1 = Update TCNTB4, TCMPB4
Timer 4 start/stop	[20]	Este bit determina el *start/stop* del Temporizador 4. 0 = Stop 1 = Start for Timer 4
Timer 3 auto reload on/off	[19]	Este bit determina el auto-reload para el Temporizador 3. 0 = One-shot 1 = Interval mode (auto reload)
Timer 3 output inverter on/off	[18]	This bit determines output inverter on/off for Timer 3. 0 = Inverter off 1 = Inverter on for TOUT3
Timer 3 manual update	[17]	This bit determine manual update for Timer 3. 0 = No operation 1 = Update TCNTB3, TCMPB3
Timer 3 start/stop	[16]	Este bit determina el *start/stop* del Temporizador 3. 0 = Stop 1 = Start for Timer 3
Timer 2 auto reload on/off	[15]	Este bit determina el auto-reload para el Temporizador 2. 0 = One-shot 1 = Interval mode (auto reload)
Timer 2 output inverter on/off	[14]	This bit determines output inverter on/off for Timer 2. 0 = Inverter off 1 = Inverter on for TOUT2
Timer 2 manual update	[13]	Este bit determina el *maual update* del Temporizador 2. 0 = No operation 1 = Update TCNTB2, TCMPB2
Timer 2 start/stop	[12]	Este bit determina el *start/stop* del Temporizador 2. 0 = Stop 1 = Start for Timer 2
Timer 1 auto reload on/off	[11]	Este bit determina el auto-reload para el Temporizador1. 0 = One-shot 1 = Interval mode (auto reload)
Timer 1 output inverter on/off	[10]	Este bit determina el inversor de salida para el Temporizador1. 0 = Inverter off 1 = Inverter on for TOUT1
Timer 1 manual update	[9]	Este bit determina el *maual update* del Temporizador 1. 0 = No operation 1 = Update TCNTB1, TCMPB1
Timer 1 start/stop	[8]	Este bit determina el *start/stop* del Temporizador 1. 0 = Stop 1 = Start for Timer 1
Dead zone enable	[4]	Este bit determina la operación de zona muerta. 0 = Disable 1 = Enable
Timer 0 auto reload on/off	[3]	Este bit determina el auto-reload para el Temporizador 0. 0 = One-shot 1 = Interval mode(auto reload)
Timer 0 output inverter on/off	[2]	Este bit determina el inversor de salida para el Temporizador 0. 0 = Inverter off 1 = Inverter on for TOUT0
Timer 0 manual update on/off	[1]	Este bit determina el *maual update* del Temporizador 0. 0 = No operation 1 = Update TCNTB0, TCMPB0
Timer 0 start/stop	[0]	Este bit determina el *start/stop* del Temporizador 0. 0 = Stop 1 = Start for Timer 0

Tabla 4.9: Direcciones de los registros de buffer y de observación para los temporizadores 0-5

Temporizador	Registro	Dirección
0	TCNTB0	0x01D5000C
	TCMPB0	0x01D50010
	TCNTO0	0x01D50014
1	TCNTB1	0x01D50018
	TCMPB1	0x01D5001C
	TCNTO1	0x01D50020
2	TCNTB2	0x01D50024
	TCMPB2	0x01D50028
	TCNTO2	0x01D5002C
3	TCNTB3	0x01D50030
	TCMPB3	0x01D50034
	TCNTO3	0x01D50038
4	TCNTB4	0x01D5003C
	TCMPB4	0x01D50040
	TCNTO4	0x01D50044
5	TCNTB5	0x01D50048
	- - -	- - -
	TCNTO5	0x01D5004C

Tabla 4.10: Rango asignado a cada uno de los dispositivos de la placa S3CEV40.

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

Bibliografía

[um-] S3c44b0x risc microprocessor product overview. Accesible en http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id=229&partnum=S3C44B0. Hay una copia en el campus virtual.