Sistema de memoria y de entrada/salida en la placa S3CEV40.

Índice general

Capítulo 1

Excepciones y modos de ejecución

Una **excepción** es un mecanismo que permite atender eventos inesperados, con origen interno (ej: intento de ejecutar una instrucción no definida) o externo (ej: solicitud de interrupción externa por parte de un dispositivo). Normalmente cuando el origen es externo se utiliza el nombre de interrupción.

La idea es sencilla: cuando se produce una excepción el procesador interrumpe de forma controlada su ejecución y pasa a ejecutar una rutina específica (habitualmente denominada *Interrupt Service Routine* o simplemente *ISR* o rutina de tratamiento de interrupciones) que tratará esa excepción. Esta rutina no puede ser diseñada como una subrutina corriente, siguiendo el AAPCS. Como la excepción es un evento no controlado por el programador, la rutina de tratamiento de la excepción debe preservar el estado del procesador completo, es decir los registros R0-R15[1] y el CPSR. Así, cuando finaliza el tratamiento de la excepción puede restaurarse el estado del procesador y retomar la ejecución del programa en el punto en que se dejó. Además, debemos tener en cuenta que pueden producirse varias excepciones simultáneamente, por lo que deberán establecerse prioridades a la hora de atenderlas.

Las excepciones en los procesadores de ARM son autovectorizadas. Esto quiere decir que cuando se produce una excepción, el procesador ejecuta automáticamente la instrucción ubicada en una dirección de memoria específica, que únicamente depende del tipo de excepción. A esta dirección se la denomina vector de la excepción (o interrupción). Normalmente esta instrucción no es más que un salto al comienzo de la rutina de tratamiento de la excepción.

El procesador ARM7TDMI tiene varios modos de ejecución, en su mayoría dedicados a atender excepciones. En la siguiente sección describimos estos modos.

1.1. Modos de Ejecución

Todos los programas desarrollados en las prácticas anteriores se ejecutaban en un modo de ejecución, aunque no nos hemos preocupado de ello. Sin embargo el ARM7TDMI dispone de 7 modos de ejecución diferentes, que permiten, entre otras cosas, la gestión eficiente de excepciones.

[1] El registro R15 no se preserva exactamente, en realidad se guarda la dirección en la que se interrumpió el programa en un registro especial para poder posteriormente reanudar su ejecución a partir de ese punto.

La figura 1.1 muestra los distintos campos del registro de estado. Los cinco bits menos significativos (`M[4:0]`) codifican el modo actual del procesador, por lo que cambiando estos bits el procesador cambia de modo. Sin embargo, la manera más habitual de cambiar de modo es a través de una excepción.

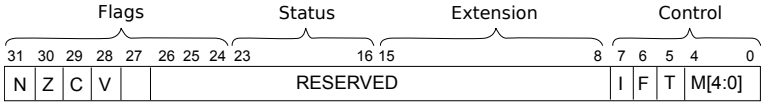

Figura 1.1: Descripción del Registro de estado (CPSR).

La tabla 1.1 describe los siete modos de ejecución del ARM7TDMI. Todos los modos excepto *User* (*usr*) son privilegiados. En los modos privilegiados no tenemos limitaciones de acceso a los recursos del procesador. En cambio, en los modos no privilegiados algunos recursos pueden estar restringidos. Concretamente, en el caso del ARM7TDMI los modos privilegiados son los únicos en los que tenemos acceso no restringido al registro de estado. En modo *usr*, en cambio, no podemos modificar directamente (con la instrucción `msr`) los bits de modo, y por tanto la única forma de cambiar de modo *usr* a cualquier otro modo es mediante una excepción.

Tabla 1.1: Modos del procesador

Modo del procesador	**Codigo**	**Uso**
usr	10000	Ejecución de código de usuario
fiq	10001	Servicio de int. rápidas
irq	10010	Servicio de int. lentas
svc	10011	Modo protegido para sistema operativo (int. sw)
abt	10111	Procesado de fallos de acceso a mem
und	11011	Manejo de instrucc. indefinidas
sys	11111	Ejecución de tareas del SO

De los modos privilegiados, cinco son conocidos como modos de excepción, debido a que están directamente relacionados con excepciones: FIQ (*fiq*), IRQ (*irq*), Supervisor (*svc*), Abort (*abt*) y Undef (*und*). El séptimo modo, System (*sys*) es diferente del resto de modos privilegiados , ya que el paso a este modo no ocurre mediante una excepción. Dicho modo lo emplea el sistema operativo cuando necesita acceder a ciertos recursos del sistema desde fuera de un modo de excepción.

Registros y modos de ejecución

En las prácticas anteriores, trabajando en modo usuario, hemos manejado 15 registros de propósito general, el PC y el registro de estado CPSR. Sin embargo, la arquitectura dispone en realidad de 37 registros de 32 bits, incluyendo el contador de programa. Estos registros se organizan en bancos parcialmente solapados, y cada modo tiene asignado uno de los bancos, como ilustra la figura 1.2.

Debemos darnos cuenta de que en todos los bancos los registros de la parte superior solapan con los del primer banco, y por tanto son los mismos que los registros del modo usuario. Sin embargo, los modos FIQ, IRQ, Supervisor, Abort y Undefined tienen algunos

registros propios, no solapados con los del modo usuario. Por ejemplo, cada uno tiene su propio puntero de pila SP (R13), lo que permite que cada modo utilice distintas zonas de memoria para la pila. Además, cada modo tiene su propio registro LR (R14). Veremos más adelante que cuando se produce una excepción el procesador cambia de modo y guarda en el LR del modo correspondiente la dirección de retorno, que servirá para retomar la ejecución del programa después de tratar la excepción. Además, cada modo, excepto el de usuario, dispone de su propio registro de sombra SPSR. Este registro se utiliza para salvar el registro de estado del programa cuando se produce una excepción y se cambia de modo de ejecución. Además, en el modo FIQ los registros R8-R12 son distintos de los del modo usuario, lo que facilita la preservación del contexto del programa (que corre en modo usuario). Finalmente, System es un modo privilegiado que utiliza los mismos registros que el modo usuario.

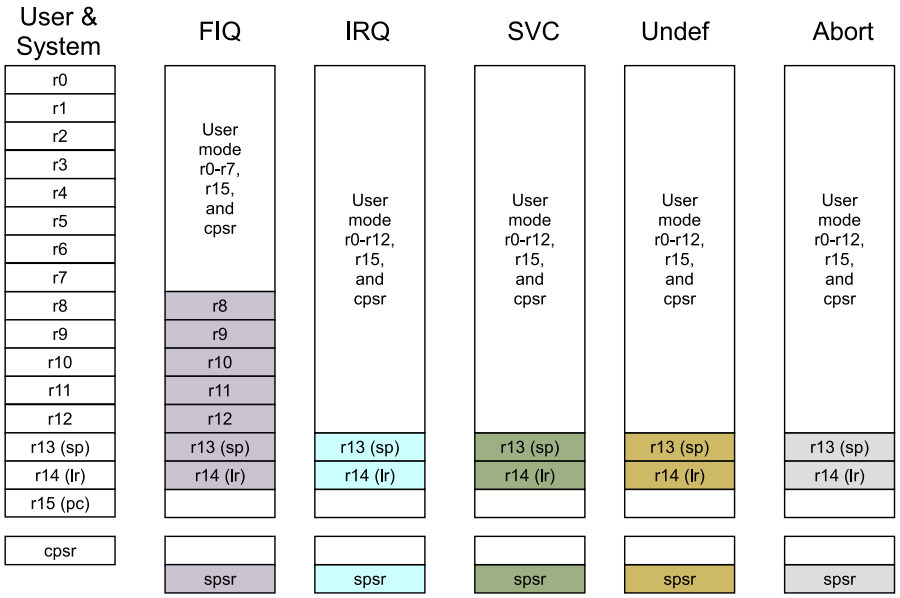

Figura 1.2: Registros visibles en cada modo de ejecución.

Como resumen, observemos que en cada modo podemos acceder a 15 registros de propósito general, llamados siempre r0-r14, el registro de estado CPSR, el registro de sombra SPSR (salvo en modo usuario) y el contador de programa PC (r15). Si un programa va a trabajar en varios modos de ejecución es necesario que inicialice el puntero de pila de cada uno de los modos que use. Por lo tanto, en las prácticas que utilicen excepciones habrá que inicializar no sólo el puntero de pila de usuario sino los de los otros modos de ejecución. Para acceder a un registro de un modo es necesario estar en ese modo de ejecución.

Cambio de modo de ejecución

Hay dos formas de cambiar de modo de ejecución: mediante una excepción o modificando los bits `M[4:0]` del registro de estado. El primer mecanismo es el único que permite el cambio de modo cuando se está en modo usuario y generalmente no es controlado por el programador. No obstante, mediante la instrucción `swi` (interrupción software), el progra-

mador puede generar una excepción que produce el cambio a modo supervisor. Éste es el mecanismo utilizado por los sistemas operativos para controlar el acceso a los recursos protegidos, y recibe el nombre de llamada al sistema. Habitualmente, esta llamada al sistema se realiza a través de una función de la biblioteca estándar de C.

El segundo mecanismo sólo está disponible cuando se está en un modo privilegiado. En este caso el cambio de modo puede realizarse escribiendo un valor adecuado en los cinco bits menos significativos del registro de estado. Para ello deben utilizarse las instrucciones de manipulación del registro de estado `mrs` y `msr`. La Tabla 1.2 resume su comportamiento.

Tabla 1.2: Instrucciones de manejo del Registro de Estado

Mnemotécnico	**Operación**
`MRS{<cond>} <Rd>, STReg`	`Rd <- STReg` donde `STReg` puede ser `CPSR` o `SPSR`.
`MSR{<cond>} STReg_campos, Op2`	donde los campos pueden ser `c,x,s,f` para los campos de control, extensión, estado y flags (ver figura 1.1), y `Op2` un registro o un inmediato. Modifica los campos indicados del registro de estado (`CPSR` o `SPSR`) con el valor de `Op2`.

Esta segunda alternativa es la que usaremos para inicializar los punteros de pila de todos los modos de ejecución. Por ejemplo, una posible secuencia de instrucciones para pasar a modo `Undef` e inicializar el puntero de pila del modo (si no lo estuviese previamente) sería la siguiente:

```
.equ    MODEMASK,   0x1f        /* Para selección de M[4:0] */
.equ    UNDEFMODE,  0x1b        /* Código de modo Undef */


mrs     r0,cpsr                 /* Llevamos el registro de estado a r0 */
bic     r0,r0,#MODEMASK         /* Borramos los bits de modo de r0 */
orr     r1,r0,#UNDEFMODE        /* Añadimos el código del modo Undef y
                                   copiamos en r1 */
msr     cpsr_cxsf,r1            /* Escribimos el resultado en el registro de
                                   estado, cambiando de éste los bits del
                                   campo de control, de extension, de estado y
                                   los de flag.  */
ldr     sp,=UndefStack          /* Una vez en modo Undef copiamos la
                                   dirección de comienzo de la pila */

                    /* Estamos en modo Undef con su pila inicializada */
```

Cuestión ¿Sería igualmente válido emplear la instrucción `mov cpsr_cxsr,r1`?¿Y la instrucción `msr cpsr_c,r1`?

1.2. Excepciones

La arquitectura ARM7TDMI (ARM V4T) reconoce, además de la excepción Reset típica de todos los procesadores, 6 excepciones adicionales. Veamos una breve descripción (para más información consultar [arm]):

Reset Se produce cuando se activa la señal externa de reset del sistema.

Undef Se produce cuando se intenta ejecutar una instrucción no definida. Si la condición de la instrucción no se cumple (recordemos que todas las instrucciones son condicionales) entonces la excepción no se produce.

SWI Se produce cuando se ejecuta la instrucción `swi` (interrupción software).

IRQ Se produce cuando se activa la línea de interrupciones externas IRQ.

FIQ Se produce cuando se activa la línea de interrupciones externas rápidas FIQ.

Abort Se distinguen dos tipos de excepción:

- **Prefetch Abort** (`PAbort`) Cuando se realiza la búsqueda (fetch) de una instrucción en una dirección no válida. El controlador de memoria es el responsable de generar la excepción.
- **Data Abort** (`DAbort`) Cuando se intenta acceder a memoria en una posición no válida, para lectura o escritura de datos. Es el controlador de memoria el responsable de generar la excepción.

La tabla 1.3, ordenada de mayor a menor prioridad, muestra la correspondencia entre las excepciones, los modos de ejecución y los vectores. Observemos que cuando se inicializa el sistema (Reset) el modo de ejecución es SVC. es decir. el sistema arranca en modo supervisor, sin restricción alguna.

Tabla 1.3: Correspondencia entre excepciones, modos y vectores.

Prioridad	**Excepción**	**Modo**	**Vector**
1	Reset	SVC	`0x00`
2	Data Abort	Abort	`0x10`
3	FIQ	FIQ	`0x1C`
4	IRQ	IRQ	`0x18`
6	Prefetch Abort	Abort	`0x0C`
7	Instruccion no definida	Undef	`0x04`
8	SWI	SVC	`0x08`

Cuando se produce una excepción el procesador realiza automáticamente (por hardware) los siguientes pasos:

1. Almacena la dirección de retorno en el registro r14 propio del modo de ejecución asociado a la excepción. En realidad el valor almacenado depende del tipo de excep-

ción[2] (consultar [arm]) lo que hace que el retorno de cada rutina de tratamiento de excepción sea distinto[3] , como veremos más adelante.

```
R14_<modo_de_excepcion>  = direccion de retorno
```

2. Copia el registro de estado (CPSR) en el registro `SPSR` del modo de ejecución correspondiente a la excepción.

```
SPSR_<modo_de_excepcion> = CPSR
```

3. Pone el código del modo de ejecución correspondiente a la excepción en los bits `M[4:0]` del registro de estado.

```
CPSR[4:0] = código del modo de excepción
```

4. Cambia al estado ARM, si no lo estuviese ya[4].

```
CPSR[5] = 0 /* Cambiar a estado ARM */
```

5. Si el modo para el tratamiento de la excepción es Reset o FIQ, el procesador deshabilita las interrupciones rápidas.

```
if <modo_de_excepcion> == Reset or FIQ then
    CPSR[6] = 1 /* Deshabilitar interrupciones rápidas */
/* else CPSR[6] no se cambia */
```

6. Deshabilita las interrupciones normales.

```
CPSR[7] = 1 /* Deshabilitar interrupciones normales */
```

7. Copia en el PC el vector correspondiente a la interrupción.

```
PC = dirección del vector de excepción
```

Resumiendo, lo que sucede ante una excepción es que el procesador guarda el registro de estado en el registro de sombra del modo y ejecuta la instrucción que está almacenada en memoria en la dirección indicada por el vector de la excepción (ver tabla 1.3). Esta instrucción debe ser un salto a la rutina encargada de tratar la excepción o, como veremos más adelante, a una rutina que lea de memoria el lugar donde se encuentra dicha rutina y realice el salto definitivo a ésta.

[2]Cada excepción se detecta en una etapa distinta del procesador.
[3]Dependiendo de la excepción el retorno debe realizarse a la propia instrucción o la siguiente.
[4]Las rutinas de tratamiento de excepción no pueden implementarse con el repertorio compacto *Thumb*.

Rutinas de tratamiento de excepción

De la descripción anterior podemos deducir que una rutina de tratamiento de excepción debe preservar, como mínimo, el valor de los registros arquitectónicos `R0-R12`, ya que:

- No se modifica el registro `R14` del modo usuario, debido a que el registro de enlace utilizado es propio de cada modo de ejecución.
- Tampoco se modifica el registro `r13` (`SP`), que también es propio de cada modo.
- La preservación del `PC` se consigue escribiendo correctamente la dirección de retorno, que podemos obtener a partir de `R14_modo`.

Sin embargo, en nuestras prácticas optaremos por ser conservadores haciendo que las rutinas de tratamiento de excepción guarden en la pila el valor de todos los registros arquitectónicos.

Como cada modo tiene su propio registro de pila, es habitual que cada modo utilice un área de memoria distinto para la pila. Para que esto sea posible es necesario inicializar los registros de pila de cada modo con una dirección distinta.

Hay que tener en cuenta que, para regresar desde una rutina de tratamiento de excepción al punto donde se había interrumpido la ejecución del programa, hay que hacer **simultáneamente** (de lo contrario el retorno no sería correcto) dos cosas:

- restaurar el valor del `CPSR` a partir del valor guardado en el `SPSR`.
- escribir en `PC` la dirección de retorno, que podemos calcular a partir del valor almacenado en `LR` siguiendo las indicaciones de la tabla 1.4. Observemos que el cálculo concreto depende de la excepción.

Tabla 1.4: Instrucción de retorno de excepción usual.

Excepción	**Inst. Retorno**
Reset	NA
Data Abort	`SUBS PC, R14_abt, #8`
FIQ	`SUBS PC, R14_fiq, #4`
IRQ	`SUBS PC, R14_irq, #4`
Prefetch Abort	`SUBS PC, R14_abt, #4`
Undef	`MOVS PC, R14_und`
SWI	`MOVS PC, R14_svc`

Hay dos mecanismos válidos para realizar correctamente el retorno:

- Se realiza el retorno mediante una instrucción de procesamiento de datos con el bit `S` activo (modificación del registro de estado) y empleando como registro destino `PC`[5] (ej. `SUBS PC, LR`).

[5]Cuando se utiliza el PC como destino en una instrucción que modifica el registro de estado, el hardware automáticamente restaura el valor de `CPSR` a partir del valor de `SPSR`

- Se realiza el retorno mediante mediante una instrucción de load múltiple (`LDM`) con el bit `S` activo (modificación del registro de estado) y empleando `PC` como uno de los registros destino[5]. Es preciso señalar que para `LDM` la activación de `S` se lleva a cabo poniendo un acento circunflejo al final de la instrucción (ej. `LDMDB FP, {R0-R13, PC}^`).

El cuadro 1 muestra una posible estructura para una rutina de tratamiento de excepción por la línea IRQ de acuerdo con la primera alternativa.

Cuadro 1 Rutina de tratamiento de IRQ para retorno `subs`

```
/* prólogo */
push {r0-r10,fp,ip,lr}   @ Basta con apilar sólo los registros modificados
                         @ r0-r3,ip,lr también si se llama a subrutina
add  fp, sp, #(4*NumRegistrosApilados-4)

/* cuerpo de la rutina */

/* epílogo */
sub  sp,fp, #(4*NumRegistrosApilados-4)
pop  {r0-r10,fp,ip,lr}   @ Restauramos contexto y retornamos
subs pc, lr, #4          @ La constante a restar depende de la excepción
```

Cuestión ¿Qué cambiaría respecto al código del Cuadro 1, el tratamiento la excepción *Data Abort*?

Escritura de rutinas de tratamiento de excepciones en C

Si queremos implementar las rutinas de tratamiento de excepción como funciones de C, debemos informar al compilador de que la función se utilizará para el tratamiento de una determinada excepción, de forma que genere el código con la estructura adecuada. En gcc esto se consigue añadiendo a la declaración de la función una directiva `__attribute__` del siguiente modo:

```
ret_val fun_name( params ) __attribute__((interrupt ( TYPE )));
```

donde `TYPE` puede ser `IRQ`, `FIQ`, `ABORT`, `UNDEF` o `SWI`.

Procediendo de esta forma gcc creará un rutina con una estructura similar a la descrita por el cuadro1, en lugar de utilizar el prólogo y el epílogo de una función C.

1.2.1. Gestión de excepciones en la placa S3CEV40

Como hemos mencionado anteriormente, las excepciones en el ARM7TDMI son autovectorizadas, es decir, el vector de interrupción se genera de forma automática en función de la excepción (ver tabla 1.3). Por lo tanto cuando se produce una excepción el procesador ejecuta la instrucción que está almacenada en memoria en la dirección indicada por el vector. Podemos comprobar en la tabla 1.3 que los vectores corresponden a direcciones

del comienzo del mapa de memoria, situadas en la ROM Flash. Dicha memoria contiene un programa de test suministrado por el fabricante, que comienza con un código similar al descrito en el cuadro 2, dónde la macro `HANDLER` y los símbolos utilizados son los del cuadro 3.

Cuadro 2 Estructura del programa ubicado en la memoria ROM Flash, en la dirección 0x00 del mapa de memoria.

```
/*Comienzo del programa en dirección 0x00*/
start:
    b ResetHandler          /* 0x00 : vector de reset    */
    b HandlerUndef          /* 0x04 : vector de Undef    */
    b HandlerSWI            /* 0x08 : vector SWI         */
    b HandlerPabort         /* 0x0C : vector de Pabort   */
    b HandlerDabort         /* 0x10 : vector de Dabort   */
    b .                     /* 0x14 : utilizado en ARMv6 */
    b HandlerIRQ            /* 0x18 : vector de IRQ      */
    b HandlerFIQ            /* 0x1C : vector de FIQ      */

    /*Más código que veremos en la práctica siguiente */
    .align

    HandlerFIQ:     HANDLER HandleFIQ
    HandlerIRQ:     HANDLER HandleIRQ
    HandlerUndef:   HANDLER HandleUndef
    HandlerSWI:     HANDLER HandleSWI
    HandlerDabort:  HANDLER HandleDabort
    HandlerPabort:  HANDLER HandlePabort

    /*Más código que veremos en la práctica siguiente */

ResetHandler:
    /* Código de la rutina de reset */
```

Como vemos, para la excepción *Reset* la instrucción en la dirección indicada por el vector (`0x00`) realiza un salto – relativo al PC – a la dirección dada por la etiqueta `ResetHandler`, donde comienza la rutina encargada de gestionar la excepción *Reset*. Podemos comprobar que esta rutina se encuentra ubicada en la misma ROM Flash, por lo que no podremos sustituir la rutina de tratamiento de *Reset* sin reprogramar la Flash.

El resto de excepciones son tratadas de otro modo. Como ejemplo vamos a analizar en detalle lo que sucede en el caso de que se produzca una excepción *Undef*. Para facilitar la comprensión del proceso, la figura 1.3 ilustra la estructura de memoria del sistema de laboratorio, mostrando con mayor detalle las partes que intervienen en el procesamiento de la excepción *Undef*.

Al producirse la excepción *Undef* el hardware pone en `PC` el valor `0x04` (vector de *Undef*), lo que produce un salto a la instrucción almacenada en esta dirección. Esta ins-

Cuadro 3 Macro `HANDLER` y símbolos utilizados por el programa de test ubicado en la ROM Flash

```
.equ    _ISR_STARTADDRESS,  0xc7fff00       /* GCS6:64M DRAM/SDRAM  */

.equ    UserStack,   _ISR_STARTADDRESS-0xf00         /* c7ff000 */
.equ    SVCStack,    _ISR_STARTADDRESS-0xf00+256     /* c7ff100 */
.equ    UndefStack,  _ISR_STARTADDRESS-0xf00+256*2   /* c7ff200 */
.equ    AbortStack,  _ISR_STARTADDRESS-0xf00+256*3   /* c7ff300 */
.equ    IRQStack,    _ISR_STARTADDRESS-0xf00+256*4   /* c7ff400 */
.equ    FIQStack,    _ISR_STARTADDRESS-0xf00+256*5   /* c7ff500 */

.equ    HandleReset,    _ISR_STARTADDRESS
.equ    HandleUndef,    _ISR_STARTADDRESS+4
.equ    HandleSWI,      _ISR_STARTADDRESS+4*2
.equ    HandlePabort,   _ISR_STARTADDRESS+4*3
.equ    HandleDabort,   _ISR_STARTADDRESS+4*4
.equ    HandleReserved, _ISR_STARTADDRESS+4*5
.equ    HandleIRQ,      _ISR_STARTADDRESS+4*6
.equ    HandleFIQ,      _ISR_STARTADDRESS+4*7

.macro HANDLER HandleLabel
    sub   sp,sp,#4         /* Decrementamos sp en 4 */
    stmfd sp!,{r0}         /* Salvamos r0 en la pila */
    ldr   r0,=\HandleLabel /* Cargamos en r0 el valor del símbolo pasado como
                              argumento a la macro */
    ldr   r0,[r0]          /* Cargamos en r0 el contenido de esta dirección,
                              que será la dirección de comienzo de la rutina
                              de tratamiento de la excepción */
    str   r0,[sp,#4]       /* Almacenamos esta dirección en la posición de la
                              pila que reservamos al comienzo */
    ldmfd sp!,{r0,pc}      /* Saltamos a la dirección restaurando r0 */
.endm
```

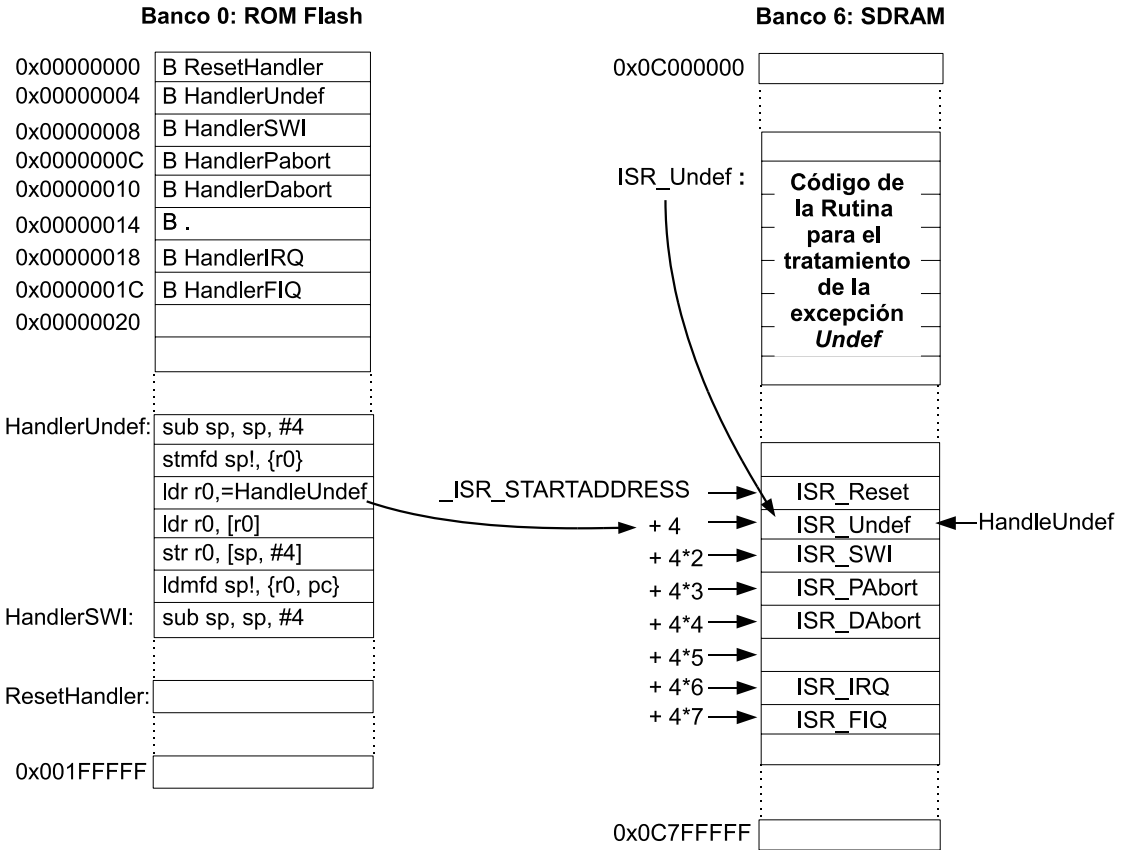

Figura 1.3: Datos e instrucciones involucrados en el procesamiento de una excepción *Undef*. El símbolo `_ISR_STARTADDRESS` indica el comienzo de la Tabla de Direcciones de ISRs y el símbolo `HandleUndef` indica la entrada de esta tabla correspondiente a la ISR de la excepción *Undef*.

trucción es un salto – relativo a `PC` – a la posición de memoria de la ROM Flash identificada por la etiqueta `HandlerUndef`, que marca el comienzo del fragmento de código encargado de obtener la dirección de comienzo de la ISR de *Undef* y realizar el salto a ella. Este fragmento de código ha sido generado mediante la macro `HANDLER`, descrita en el cuadro 3, del siguiente modo:

```
HandlerUndef HANDLER HandleUndef
```

Si analizamos este código (ver figura 1.3) observaremos que, después de salvar en la pila el registro `R0` para conservar su valor, lee de memoria el valor del símbolo `HandleUndef`, que se pasó como argumento a la macro, y lo guarda en `R0`. Este símbolo se corresponde con una dirección de la SDRAM que contiene a su vez la dirección de la rutina de tratamiento de la excepción *Undef* representada por el símbolo `ISR_Undef`. La instrucción `ldr r0,[r0]` se encarga de guardar esta dirección en el registro `R0` y la siguiente instrucción (`str r0, [sp, #4]`) se encarga a su vez de guardarla en la pila. Por último, se restaura el valor de `R0` y se salta a `ISR_Undef` mediante la instrucción `ldmfd sp!,{r0,pc}`.

Resumiendo, para todas las excepciones salvo *Reset*, las direcciones de las rutinas encargadas de tratar las excepciones se leen de unas posiciones fijas de la memoria

SDRAM, que habitualmente denominaremos *Tabla de Direcciones de ISRs*. Podemos ver además que estas posiciones de memoria comienzan en la dirección dada por el símbolo `_ISR_STARTADDRESS`, cuyo valor definido en el cuadro 3 es `0xc7fff00`.

Concluyendo, si queremos tratar las excepciones (salvo Reset) sin tener que modificar el contenido de la flash, debemos escribir en la tabla que comienza en la dirección `0x0c7fff00` las direcciones de comienzo de nuestras rutinas de tratamiento de excepción.

Cuestiones

- En el *SoC* del laboratorio, con la *ROM* tal y como la entrega el fabricante (por defecto), al recibir una excepción *Undef* se salta a la dirección 0x00000004. ¿Qué instrucción se encuentra en esa dirección? Dicho de otro modo, ¿en qué dirección de memoria comienza la rutina HANDLER para la excepción *UNDEF*?
- ¿En qué dirección está la entrada de la tabla que consulta la rutina HANDLER para la excepción *UNDEF* en la que encontraremos el comienzo de la ISR escrita por nosotros para tratar esa excepción? ¿Qué ocurre si no hemos inicializado esa posición de memoria y se produce esta excepción?

Capítulo 2

Sistema de de memoria

La figura 2.1 representa de forma esquemática el subsistema de memoria de la placa empleada en el laboratorio[1]. El funcionamiento del sistema de memoria depende en realidad de tres componentes de nuestro kit de laboratorio, el procesador (ARM7TDMI), el sistema en chip en el que se integra (S3C44BOX) y la placa sobre la que se coloca este último (S3CEV40). A continuación describiremos las principales características de cada uno de sus componentes.

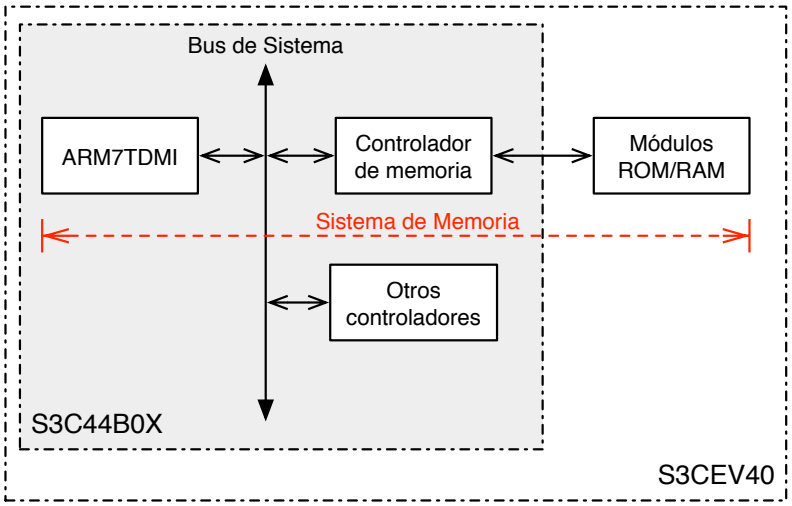

Figura 2.1: Sistema de memoria de la placa de laboratorio (Embest S3CEV40).

2.1. El ARM7TDMI

El bus de direcciones del ARM7TDI es de 32 bits, por lo que es capaz de direccionar potencialmente un espacio total de 4GB de memoria. Es preciso señalar además que este espacio de direcciones es compartido por el sistema de E/S (*E/S localizada en memoria*),

[1] Aunque en el *SoC* S3C44B0X dispone de un *Write Buffer* y una *Cache* de 8K, por defecto ambos están desactivados.

por lo que las direcciones pueden referirse tanto a posiciones de memoria principal como a elementos internos de los controladores de E/S (registros o memoria local). En ambos casos el ARM7TDMI realiza el acceso del mismo modo y es responsabilidad del programador saber a qué dispositivo está asociada cada dirección. Esta asignación de rangos de direcciones a dispositivos suele recibir el nombre de *mapa de memoria del sistema*.

2.2. El controlador de memoria

El controlador de memoria es el responsable de actuar de interfaz entre los módulos de memoria externos (ROM o RAM) y el bus del sistema. Es fácil adivinar que el *mapa de memoria del sistema* viene determinado en gran medida por las características y configuración de este elemento del sistema de memoria.

El comportamiento del controlador de memoria podría resumirse del siguiente modo. Cuando el procesador pone una dirección en el bus:

- Si ésta se encuentra dentro del rango del controlador, él se encarga de generar las señales necesarias para realizar el acceso al módulo de memoria que corresponda.
- Si, por el contrario, la dirección queda fuera de su rango de competencia, el controlador se inhibe, ya que supuestamente es responsabilidad de algún controlador de E/S atender a dicho acceso (lectura/escritura de un registro o memoria local).

Si el acceso a memoria falla (p.ej. se realiza el acceso a un banco para el que no hay asignado módulo de memoria), es responsabilidad del controlador detectar el error y generar la correspondiente excepción de *Abort*.

2.2.1. El controlador de memoria del S3C44B0X

En la placa del laboratorio (Embest S3CEV40), tanto el procesador ARM7TDMI como el controlador de memoria, así como otros controladores y dispositivos de E/S, se encuentran integrados dentro un mismo chip, el *System on Chip* S3C44B0X de Samsung.

El controlador de memoria del S3C44B0X reduce el espacio de direcciones efectivo a 256MB y lo divide en 8 fragmentos independientes, denominados *bancos*, tal y como ilustra la figura 2.2. Cada banco puede ser asignado a un chip de memoria externo distinto (módulo), aunque sólo los dos últimos bancos admiten cualquier tipo de memoria (SROM, SRAM o SDRAM). Cuando se accede a una dirección dentro del rango de uno de estos bancos, además de las señales necesarias para realizar el acceso al módulo al que está asociado, el controlador de memoria activa una señal nGCS[2]. Estas señales se utilizan externamente para seleccionar/activar el chip correspondiente.

De todo el espacio de memoria, la parte alta del primer banco, correspondiente al rango `0x01C00000-0x01FFFFFF`, está reservada para los puertos de E/S de los controladores integrados dentro del S3C44B0X. Como hemos mencionado previamente, cuando el procesador realiza un acceso dentro este rango el controlador de memoria se inhibe (i.e. no genera señal alguna).

[2]Por convenio, las señales cuyo nombre comienza con "n" se activan en baja.

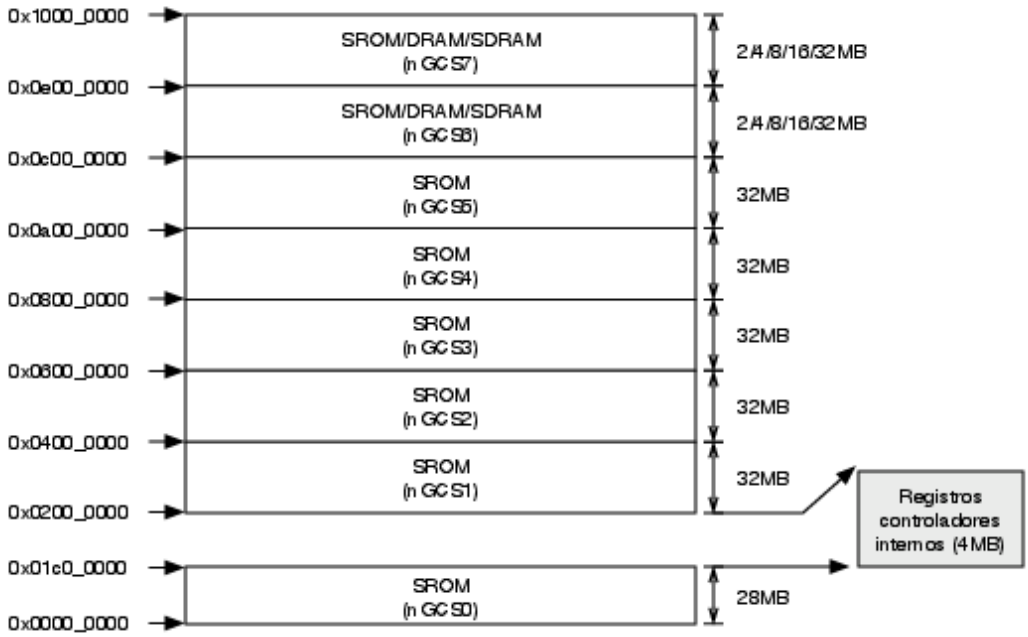

Figura 2.2: Mapa de memoria del S3C44B0X (SROM se refiere tanto a ROM como a SRAM).

2.3. Los módulos de memoria

La placa que se emplea en el laboratorio (Embest S3CEV40) dispone de dos módulos de distinto tipo:

- Memoria ROM Flash de 1Mx16bits, ubicada en el banco 0 del S3C44B0X y cuyo rango de direcciones es `[0x00000000-0x001FFFFF]`.
- Memoria SDRAM de 4Mx16bits, ubicada en el banco 6 del S3C44B0X y cuyo rango de direcciones es `[0x0C000000-0x0C7FFFFF]`.

Como podemos ver, el espacio de 256MB permitido por el controlador de memoria se reduce en nuestro caso a 10MB, 2 de ROM Flash y 8 de SDRAM.

Capítulo 3

Sistema de Entrada/Salida

La figura 3.1 representa de forma esquemática el sistema de entrada/salida de la placa empleada en el laboratorio. Sus principales componentes son el procesador ARM7TDMI, el controlador de interrupciones, los controladores de E/S – tanto internos como externos al Samsung S3C44B0X –, y la lógica de selección, que se sirve de las señales generadas por el controlador de memoria para seleccionar el chip externo con el que se desea trabajar. A continuación describiremos brevemente las principales características de cada uno de sus componentes.

3.1. Controladores y dispositivos de E/S (GPIO)

Salvo contadas excepciones, los dispositivos de E/S se gestionan a través de controladores específicos de E/S. Estos son los que actúan de interfaz entre el dispositivo y el bus de sistema y el controlador de interrupciones. Para comunicarse con el dispositivo, el procesador lee y escribe en los registros internos de estos controladores, a los que nos referiremos habitualmente como puertos de E/S.

Selección de dispositivos/controladores

Como hemos mencionado previamente, el procesador ARM7TDMI emplea E/S localizada en memoria y por lo tanto no es capaz de distinguir si una dirección se corresponde con un puerto de E/S o con una posición memoria. Anteriormente vimos que el controlador de memoria juega un papel esencial a la hora de discriminar entre ambos: asigna un rango de direcciones a los registros correspondientes a los controladores internos al S3C44B0X (de E/S, de memoria, de interrupciones, ...) y el resto de los 256MB del espacio de direcciones lo asigna a memoria y se encarga de su gestión, generando las señales necesarias para realizar la comunicación con el módulo y activando la señal nGCS correspondiente al banco (nGCS0-nGCS7).

Además de los controladores internos, existen en el sistema controladores de E/S externos. Los puertos de E/S correspondientes a estos controladores externos al S3C44B0X se ubican fuera del rango de direcciones correspondiente a los controladores internos y requieren de una lógica adicional para su selección (ver figura 3.1). Obviamente, no pueden

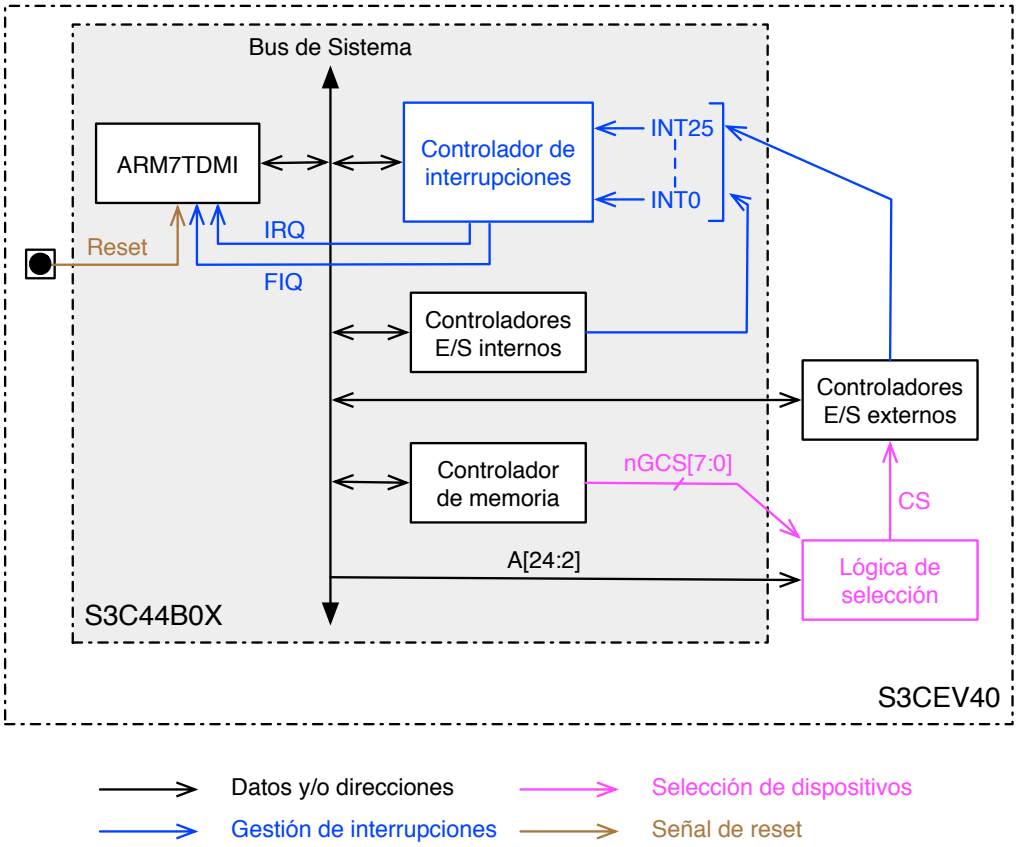

Figura 3.1: Sistema de E/S de la placa de laboratorio.

asignarse puertos de E/S a bancos que tengan asignados módulos de memoria. Esta lógica de selección es la encargada de activar la señal de *Chip Select* (CS) correspondiente al dispositivo que se pretende acceder en función del valor de las señales nGCS0-7 y de los bits de direcciones del bus del sistema.

Capítulo 4

Entrada/salida mediante interrupciones

El procesador ARM7TDMI dispone de tres líneas que permiten interrumpir su ejecución de manera externa: una línea de RESET de comportamiento asíncrono, que genera una excepción de Reset, y dos líneas, IRQ y FIQ de comportamiento síncrono y enmascarables, que excepciones IRQ y FIQ respectivamente. Este tipo de excepciones generadas por elementos externos al procesador, suelen denominarse genéricamente interrupciones.

La excepción de RESET tiene por propósito la ejecución de un código que realiza la inicialización del sistema. Las excepciones de IRQ y FIQ, por su parte, tienen por propósito ejecutar el código necesario para atender al dispositivo que ha solicitado la interrupción. Ambas juegan por lo tanto un papel esencial en la gestión de la entrada/salida.

Las líneas IRQ y FIQ pueden enmascararse mediante dos bits del registro de estado (CPSR), I y F respectivamente (ver figura 1.1). Cuando estos bits toman el valor `'1'` las solicitudes de interrupción efectuadas por su correspondiente línea no son atendidas por el procesador. Durante la inicialización del sistema es crucial enmascarar ambas líneas ya que éste no se encuentra aún preparado para que las interrupciones sean atendidas.

4.1. Identificación de la fuente de interrupción

Desde el punto de vista del procesador, todas las excepciones, incluyendo IRQ y FIQ, son autovectorizadas. Esto quiere decir que el procesador genera automáticamente para cada tipo de excepción un vector (dirección), que es cargado directamente en el contador de programa. Como vimos anteriormente, en esta dirección de memoria debe almacenarse una instrucción de salto a la rutina de tratamiento correspondiente a la excepción.

Si varios dispositivos comparten la línea de petición de interrupción, ya sea IRQ o FIQ, generarán el mismo tipo de excepción y ejecutarán la misma rutina de tratamiento. En este caso, por lo tanto, es necesario que esta rutina identifique cuál ha sido el dispositivo que ha solicitado la interrupción. Para hacerlo, debe leer los registros de estado de los controladores de E/S asociados para comprobar si tienen una interrupción pendiente. A este proceso se le denomina *identificación por encuesta*. Asimismo, si hay varios dispositivos que hayan solicitado una interrupción simultáneamente, es la propia rutina la que tiene que

decidir a cuál de ellos se atiende primero. Se dice en este caso que el *arbitraje es software*, y habitualmente la prioridad viene dada por el orden en el que se consultan los registros de estado.

4.2. El controlador de interrupciones

La función de un controlador de interrupciones es la de mejorar/ampliar la gestión de interrupciones del procesador, en nuestro caso del ARM7TDMI. Para ello ha de situarse entre el procesador y los controladores de E/S encargados de gestionar los dispositivos, como podemos apreciar en la figura 3.1 .

Aumento de líneas de interrupción

El controlador de interrupciones incluido en el Samsung S3C44B0X permite desdoblar las dos líneas de interrupción, IRQ y FIQ, en 26 líneas independientes enmascarables que pueden ser configuradas de forma que activen una de estas dos líneas, como veremos posteriormente. La figura 3.1 muestra de forma esquemática las principales señales que emplea el controlador. La tabla 4.1 muestra, entre otras cosas, el nombre que recibe cada una de estas líneas y el tipo de fuente de interrupción (controlador de DMA, UART, etc.).

Mejora del proceso de identificación

Este controlador permite además mejorar el proceso de identificación de la fuente de interrupción de dos formas distintas según el modo en el que haya sido configurado:

- *Modo No Vectorizado.*
  En este modo cuando una o más líneas del controlador se activan, el controlador activa – siempre que no estén enmascaradas – la línea FIQ o IRQ si alguna de ellas la tiene asociada. Recordemos que entonces el procesador ARM7TDMI generará el vector correspondiente a FIQ o IRQ según el caso[1], y saltará a la rutina programada para ese tipo de excepción. Para llevar a cabo la identificación, esta rutina debe consultar un registro del controlador de interrupciones que le indica cuáles son las líneas con interrupciones pendientes. Es preciso resaltar que este proceso de identificación es mucho más rápido que sin la presencia del 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). Por supuesto, si alguna de las líneas del controlador es compartida por varios dispositivos, para llevar a cabo la identificación es preciso consultar sus registros cuando la línea se active.

- *Modo Vectorizado.*
  En este modo cuando una o más líneas del controlador se activan:

  - Si alguna línea activa tiene asociada la interrupción FIQ, entonces se comporta como en el modo no vectorizado.

[1] Si ambas líneas estuviesen activas simultáneamente, se atendería primero a FIQ ya su excepción es más prioritaria.

- Si todas las líneas activas tienen asociada la interrupción IRQ, se producirá un excepción IRQ y se realizará un salto a la dirección 0x18 – vector de IRQ –, ubicación en la que está almacenada la instrucción encargada de saltar a la ISR de IRQ. No obstante, cuando el procesador pone en el bus esta dirección para leer esta instrucción, el controlador de interrupciones inserta en el bus otra instrucción de salto alternativa, evitando que se acceda a la que está almacenada en memoria. Este mecanismo, permite al controlador redirigir la ejecución de código a una posición de memoria distinta en función de la línea que se haya activado. El efecto conseguido con esto es como si cada línea del controlador estuviese autovectorizada. Los vectores generados para cada línea del controlador están recogidos en la tabla 4.1. Es conveniente señalar que en este modo, puesto que la identificación de la línea la hace el propio controlador, el abitraje (prioridad) debe también realizarlo él. Por defecto, cuando hay varias líneas activas – no enmascaradas – el controlador de interrupciones del S3C44B0X atiende en primer lugar a aquella línea cuyo número de línea sea mayor (EINT0 es la línea de mayor prioridad y ADC la de menor).

4.3. Configuración del controlador de interrupciones

La configuración y uso 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 se recoge en la tabla tabla 4.2, 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 el *Modo Vectorizado*
- 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 por lo tanto 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 líneas (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). Es esencial cuando se están en *Modo Vectorizado* ya que el arbitraje es hardware.

Tabla 4.1: Descripción de las 26 líneas de interrupción gestionadas por el controlador de interrupciones (externas al procesador). Estas líneas permiten gestionar 30 fuentes de interrupción distintas. La línea 21 es compartida por 4 fuentes de interrupción y la línea 14 por 2, el resto de líneas tienen asociada una única fuente de interrupción.

Nº/Bit	**Nombre**	**Fuente**	**Vector**
25	EINT0	Interrupción externa 0	0x20
24	EINT1	Interrupción externa 1	0x24
23	EINT2	Interrupción externa 2	0x28
22	EINT3	Interrupción externa 3	0x2c
21	EINT4/5/6/7	Interrupciones externas 4, 5, 6 y 7	0x30
20	TICK	Interrupción de *tick* del RTC	0x34
19	ZDMA0	Interrupción del ZDMA0	0x40
18	ZDMA1	Interrupción del ZDMA1	0x44
17	BDMA0	Interrupción del BDMA0	0x48
16	BDMA1	Interrupción del BDMA1	0x4c
15	WDT	Interrupción del Watch-Dog Timer	0x50
14	UERR0/1	Interrupciones de error de las UART0/1	0x54
13	TIMER0	Interrupción del Timer0	0x60
12	TIMER1	Interrupción del Timer1	0x64
11	TIMER2	Interrupción del Timer2	0x68
10	TIMER3	Interrupción del Timer3	0x6c
9	TIMER4	Interrupción del Timer4	0x70
8	TIMER5	Interrupción del Timer5	0x74
7	URXD0	Interrupción de recepción de la UART0	0x80
6	URXD1	Interrupción de recepción de la UART1	0x84
5	IIC	Interrupción de controlador de bus IIC	0x88
4	SIO	Interrupción del controlador SIO	0x8c
3	UTXD0	Interrupción de envío de la UART0	0x90
2	UTXD1	Interrupción de envío de la UART1	0x94
1	RTC	Interrupción de alarma del RTC	0xa0
0	ADC	Interrupción *EOC* del conversor ADC	0xc0

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. Si lo que se escribe es un '0' el bit correspondiente de INTPND permanece inalterado. Es preciso resaltar que mediante la escritura en este registro se indica al controlador de interrupciones que ha finalizado la rutina de servicio y, que por lo tanto, puede comenzar a atender una nueva solicitud (en otras arquitecturas esta acción se conoce mediante *End of Interrupt* o simplemente EIO).

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-].

Tabla 4.2: Registros del controlador de interrupciones

Registro	**Dirección**	**R/W**	**Valor de reset**
INTCON	0x01E00000	R/W	0x7
INTPND	0x01E00004	R	0x00000000
INTMOD	0x01E00008	R/W	0x00000000
INTMSK	0x01E0000C	R/W	0x07FFFFFF
I_ISPR	0x01E00020	R	0x00000000
I_ISPC	0x01E00024	W	Undef
F_ISPC	0x01E0003C	W	Undef

Capítulo 5

Controlador de pines de E/S (GPIO)

El controlador de pines de E/S de propósito general (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 a estos pines. En las prácticas vamos a utilizar sólo algunos de los pines de E/S de los puertos B y G. La tabla 5.1 recoge una relación de sus registros de datos y control, que describiremos en detalle a continuación. Para obtener una descripción completa es preciso consultar el manual del S3C44B0X [um-].

Tabla 5.1: Registros de los puertos B y G

Registro	**Dirección**	**R/W**	**Valor de Reset**
PCONB	0x01D20008	R/W	0x7FF
PDATB	0x01D2000C	R/W	Undef
PCONG	0x01D20040	R/W	0x00
PDATG	0x01D20044	R/W	Undef
PUPG	0x01D20048	R/W	0x00
EXTINT	0x01D20050	R/W	0x00000000
EXTINTPND	0x01D20054	R/W	0x00

5.1. Puerto B

El puerto B tiene 11 bits, que admiten dos funcionalidades: podemos utilizar estos pines como pines de salida (para escribir un valor directamente en ellos) o podemos conectarlos a algunas señales generadas por el controlador de memoria (*i.e.* hacemos que estas señales salgan por estos pines fuera del S3C44B0X). 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 descripción está en la tabla 5.2.
- *PDATB*, registro de datos que permite escribir los bits que se pretenden sacar por el puerto (sólo útil para aquellos pines configurados como salida).

Tabla 5.2: Configuración de los pines del puerto B.

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

En las prácticas utilizaremos dos pines del puerto B configurados como salida (los pines 9 y 10, como veremos más adelante), para encender o apagar dos LEDs conectados a ellos.

5.2. Puerto G

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

- *PCONG*, registro de control que permite seleccionar la funcionalidad de cada pin, tal y como se describe en la tabla 5.3.
- *PDATG*, registro de datos que permite escribir o leer del puerto.
- *PUPG*, registro de configuración que permite activar (bit a '0') o no (bit a '1') una resistencia de *pull-up*[1] por cada pin.

[1]Una resistencia de *pull-up* pone la entrada a uno cuando está desconectada, evitando que se quede en un valor indefinido. De forma similar, una resistencia de *pull-down* ponde la entrada a cero cuando está desconectada.

Tabla 5.3: Configuración de los pines del puerto G.

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

Como vemos en la tabla, podemos utilizar los bits del puerto G como pines de entrada o de salida o conectarlos a algunas señales internas del sistema. Alternativamente podemos configurar los pines del puerto para que activen las líneas `EINT*` de petición de interrupción. Esto permite que algún hardware externo conectado a estos pines pueda generar una interrupción por estas líneas. Cuando opera de esta forma este puerto admite diversas opciones para desencadenar la interrupción. La interrupción puede generarse cuando se detecte un nivel de voltaje en el pin o cuando se detecte un determinado flanco. Esto se configura mediante un cuarto registro del GPIO, *EXTINT* cuyo uso se describe en la tabla 5.4.

El controlador GPIO permite además descomponer en cuatro la línea EINT4/5/6/7 del controlador de interrupciones, cuyo funcionamiento se explicará más adelante. La línea EINT4/5/6/7 procedente del controlador de interrupciones se activa cuando alguna de estas cuatro líneas genera una interrupción (en realidad se hace una OR). El registro *EXTINTPND* del puerto G permite saber cuál de las cuatro líneas es la que ha generado la interrupción como describe la tabla 5.5.

Es preciso resaltar que para borrar la petición de interrupción por una de estas líneas de interrupción, la rutina de servicio debe borrar el bit correspondiente de este registro (escribiendo un '1', no un '0') antes de borrar el bit de INTPND del controlador de interrupciones.

Más adelante se explicará cómo utilizar el puerto G para leer los pulsadores o para generar una interrupción cuando se detecte que se ha pulsado uno de ellos.

5.3. LEDs y pulsadores

En la placa S3CEV40 hay dos pulsadores y dos LEDs conectados al sistema S3C44BOX como indica la figura 5.1.

Tabla 5.4: 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

Los LEDs se conectan a los pines 9 y 10 del puerto B. Como puede apreciarse en la figura 5.1, su ánodo se conecta a la alimentación (Vdd) por lo que para iluminar estos LEDs es necesario configurar los pines 9 y 10 como sólo salida (ver tabla 5.2) y escribir en ellos un '0' mediante el registro *PDATB*.

Los dos pulsadores se conectan a los pines 6 y 7 del puerto G. Configurando estos pines como entrada leeremos un 0 en el bit correspondiente del registro PDATG cuando el pulsador está pulsado. Para asegurarnos que los pines tomen el valor 1 lógico (Vdd) cuando el pulsador no está pulsado es necesario activar las resistencias de pull-up de estos pines mediante el registro PUPG. Sin embargo, cuando se produzca una pulsación en realidad no se producirá una bajada limpia de la tensión de Vdd a 0, sino que la tensión oscilará (entre Vdd y 0) hasta que finalmente se estabilice en 0. Solemos decir que se producen rebotes de la señal. Esto puede hacer que detectemos varias pulsaciones cuando sólo se ha producido una. Para filtrar estos rebotes debemos esperar un tiempo cuando detectemos una pulsación. El tiempo a esperar puede ajustarse empíricamente, pero 100ms suele ser

Tabla 5.5: 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.

suficiente. Podemos hacer esta espera con una función Delay que básicamente consiste en dos bucles anidados que no hacen nada (cuerpo del bucle interno vacío).

Alternativamente, podemos hacer llegar los pines 6 y 7 del puerto G, conectados a los pulsadores, a las líneas EXINT6 y EXINT7 del controlador de interrupciones estableciendo el valor adecuado de *PCONG* (ver tabla 5.3). Para generar interrupciones mediante estos pulsadores, es necesario, además, activar las resistencias de *pull-up* de estos pines mediante el registro *PUPG* (para mantener constante el valor de entrada cuando el pin está al aire), o de lo contrario el comportamiento sería imprevisible. Asimismo es necesario configurar el tipo de interrupción (nivel alto/bajo, flanco subida/bajada/ambos) mediante el registro *EXTINT* del puerto.

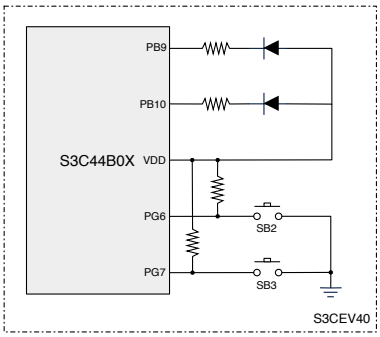

Figura 5.1: Esquema de conexión de pulsadores y LEDs de la placa Embest S3CEV40.

Capítulo 6

Display 8-segmentos

En la placa S3CEV40 también tenemos un display de 8 segmentos, que consta de 7 LEDs para conformar cualquier dígito hexadecimal y otro LED adicional para el punto decimal. Este display es de tipo ánodo común, lo significa que para encender los LEDs hay que ponerlos a cero. La Figura 6.1 muestra la conexión del display a la lógica de selección externa. Los LEDs del display se conectan al bus de datos del procesador a través de un latch triestado (el chip 74LS573). La selección de este dispositivo se lleva a cabo por medio de la señal CS6, generada por un decodificador de 3 a 8 (chip 74LV138), que es habilitado con la señal nGCS1 según el esquema de la figura 6.1. Esta lógica de selección es común a todos los dispositivos ubicados en el Banco-1. La tabla 6.1 muestra el rango de direcciones y la señal de selección que le asigna a cada uno de ellos. Como podemos ver, el latch conectado al display de 8 segmentos se activa con la señal CS6, cada vez que accedemos a alguna dirección del rango 0x0214_0000-0x0217_FFFF. Mientras la entrada LE (Latch Enable) permanezca activa el latch dejará pasar el byte definido por los 8 bits menos significativos del bus de datos. Cuando esta señal se desactiva el latch mantiene el último valor que le ha llegado. Sin embargo, el banco 1 tiene una anchura de 8 bits. Esto quiere decir que si hacemos una escritura de tamaño palabra el controlador de memoria enviará por separado los 4 bytes que componen la palabra. Como el procesador se configura como little-endian se escribirían los bytes en orden de su peso (0,1,2 y 3), quedando en el latch el byte más significativo, en lugar del menos significativo. Por este motivo es muy importante que siempre se realicen escrituras de tamaño byte en el display de 8 segmentos, utilizando la instrucción strb.

Tabla 6.1: Rango asignado a cada uno de los dispositivos ubicados en el Banco-1.

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

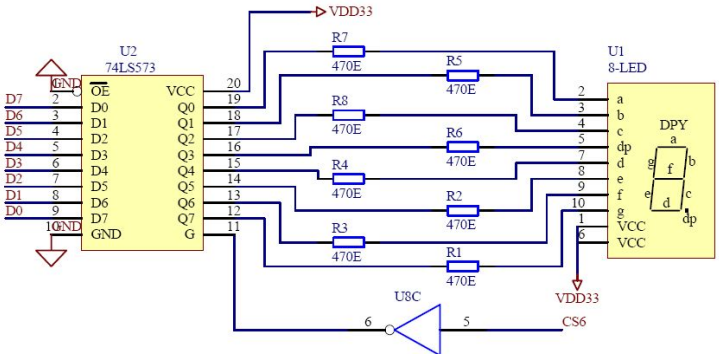

Figura 6.1: Esquema de conexión del display de 8 segmentos en la placa S3CEV40. A[20:18] son bits de direcciones y D[7:0] son bits de datos del bus del sistema del S3C44B0X. Es conveniente recordar que la señal nGCS1 del controlador de memoria se activa a baja cuando se referencia una dirección del Banco-1.

Capítulo 7

Temporizadores

El S3C44B0X tiene 6 temporizadores de 16 bits, conectados según indica la figura 7.1. Cada uno puede ser configurado para operar por interrupciones o por DMA. Los 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 7.1. 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. Los registros utilizados para el manejo de los temporizadores se recogen en la tabla 7.1.

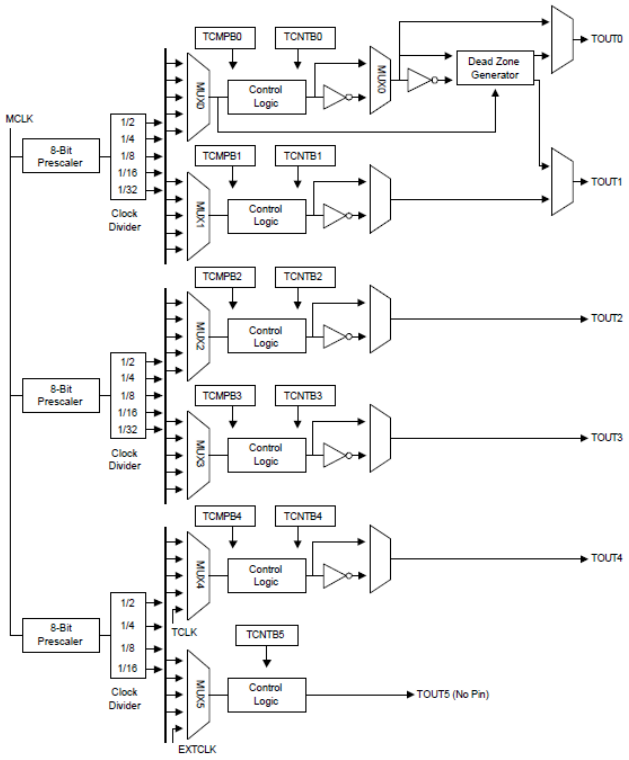

Figura 7.1: Esquema de conexión de los temporizadores del S3C44B0X.

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 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 7.1, cada par de temporizadores (0-1, 2-3, 4-5) comparte 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 de 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.

Tabla 7.1: Registros para la utilización de los temporizadores

Registro	Dirección	R/W	Valor de Reset
TCFG0	0x01D50000	R/W	0x00000000
TCFG1	0x01D50004	R/W	0x00000000
TCON	0x01D50008	R/W	0x00000000
TCNTB0	0x01D5000C	R/W	0x00000000
TCMPB0	0x01D50010	R/W	0x00000000
TCNTO0	0x01D50014	R	0x00000000
TCNTB1	0x01D50018	R/W	0x00000000
TCMPB1	0x01D5001C	R/W	0x00000000
TCNTO1	0x01D50020	R	0x00000000
TCNTB2	0x01D50024	R/W	0x00000000
TCMPB2	0x01D50028	R/W	0x00000000
TCNTO2	0x01D5002C	R	0x00000000
TCNTB3	0x01D50030	R/W	0x00000000
TCMPB3	0x01D50034	R/W	0x00000000
TCNTO3	0x01D50038	R	0x00000000
TCNTB4	0x01D5003C	R/W	0x00000000
TCMPB4	0x01D50040	R/W	0x00000000
TCNTO4	0x01D50044	R	0x00000000
TCNTB5	0x01D50048	R/W	0x00000000
TCNTO5	0x01D5004C	R	0x00000000

Cada temporizador (excepto el 5) tiene un par de registros asociados, TCNTn y TCMPn. El registro de cuenta (TCNTn) se inicializa a un determinado valor y se decrementa en cada ciclo mientras el temporizador esté activo. El registro de comparación (TCMPn) se inicializa a otro valor y como su propio nombre indica se emplea para comparar con el registro de cuenta. El resultado de esta comparación se emplea para controlar la señal de salida. Al comienzo de cada cuenta esta señal tiene un valor 0, cuando el contador alcanza el valor del registro de comparación la señal toma el valor 1. De esta forma podemos controlar exactamente la anchura de los pulsos (PWM), es decir 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 detiene la cuenta.

La figura 7.2 representa el diagrama temporal de un ejemplo de 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 contador. 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).

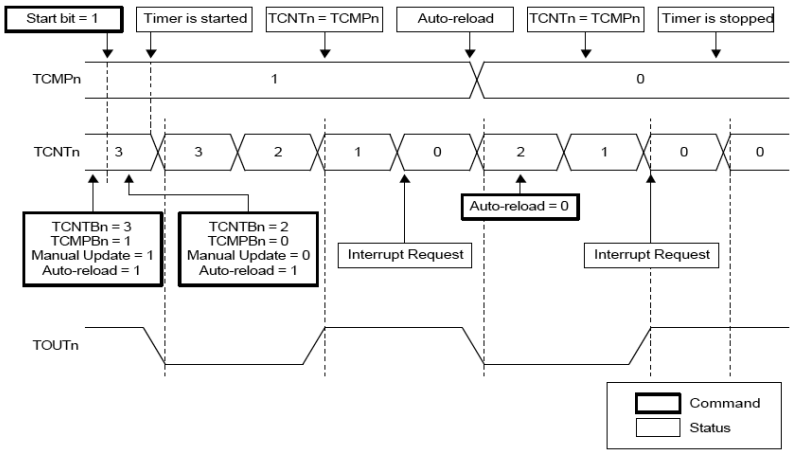

Figura 7.2: Diagrama temporal del funcionamiento de los temporizadores del S3C44B0X.

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.

Inicialización de los temporizadores

Como el valor del registro TCNTBn se copia en el registro de cuenta TCNTn sólo cuando TCNTn alcanza cero, el temporizador no puede ser inicializado simplemente escribiendo en TCNTBn. Para que el valor que escribimos en TCNTBn 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, descrito en la tabla 7.2.

Tabla 7.2: 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.

El valor del divisor se configura con el registro TCFG1, así como la asignación de un temporizador al canal de petición del DMA. La descripción del registro aparece en la tabla 7.3.

El control de los temporizadores (puesta en marcha, parada, actualización, modo auto-reload, etc) se realiza mediante el registro TCON. Su funcionalidad está descrita en la tabla 7.4.

Finalmente, para manejar los temporizadores debemos utilizar los registros de buffer, tanto de cuenta como de comparación, y el registro de observación.

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

Tabla 7.3: Registro *TCFG1*.

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

Tabla 7.4: Registro *TCON*.

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 *manual 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]	Este bit determina el inversor de salida para el Temporizador 4. 0 = Inverter off 1 = Inverter on for TOUT4
Timer 4 manual update	[21]	Este bit determina el *manual 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]	Este bit determina el inversor de salida para el Temporizador 3. 0 = Inverter off 1 = Inverter on for TOUT3
Timer 3 manual update	[17]	Este bit determina el *manual update* del Temporizador 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]	Este bit determina el inversor de salida para el Temporizador 2. 0 = Inverter off 1 = Inverter on for TOUT2
Timer 2 manual update	[13]	Este bit determina el *manual 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 Temporizador 1. 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 *manual 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 *manual 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

Capítulo 8

Teclado matricial

8.1. Decripción

El maletín del laboratorio dispone un teclado matricial. En este tipo de teclados las teclas están dispuestas según un array bidimensional como el mostrado en el Figura 8.1: el teclado es un array de líneas horizontales y líneas verticales junto con 16 interruptores (SB1-SB16) que al ser pulsados conectan una línea horizontal con una línea vertical. Se trata por lo tanto de un dispositivo extremadamente sencillo, que requiere de una lógica adicional para su uso.

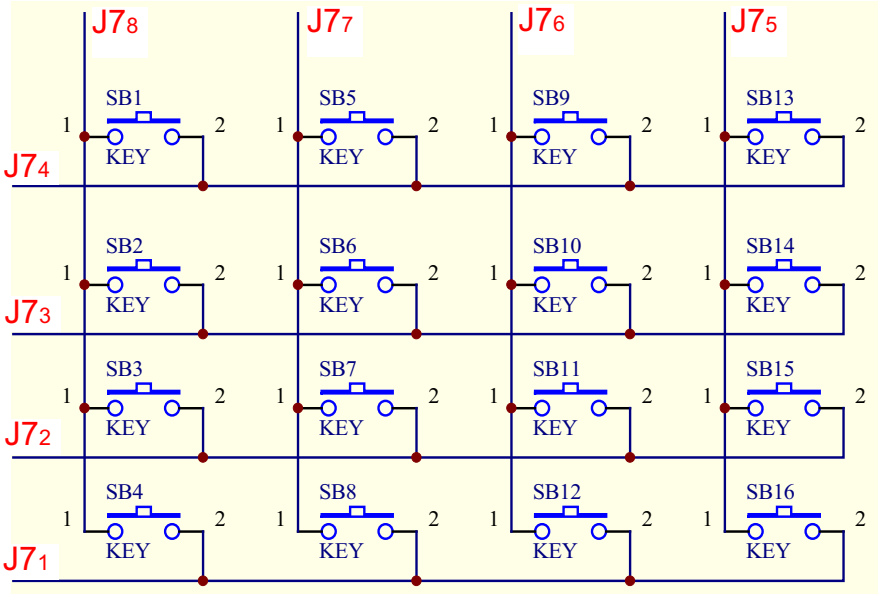

Figura 8.1: Circuito del teclado 4x4.

8.2. Conexión del teclado

Este teclado, situado en una placa suplementaria, se conecta al resto de componentes a través del conector con nombre de instancia J7, situado en la parte inferior derecha de la placa. Dicho conector posee ocho pines y está conectado a los componentes que se muestran en la Figura 8.2. A diferencia de los controladores de E/S al uso, está lógica no dispone de registros de control/estado/configuración y su manejo, aunque sencillo, es poco intuitivo.

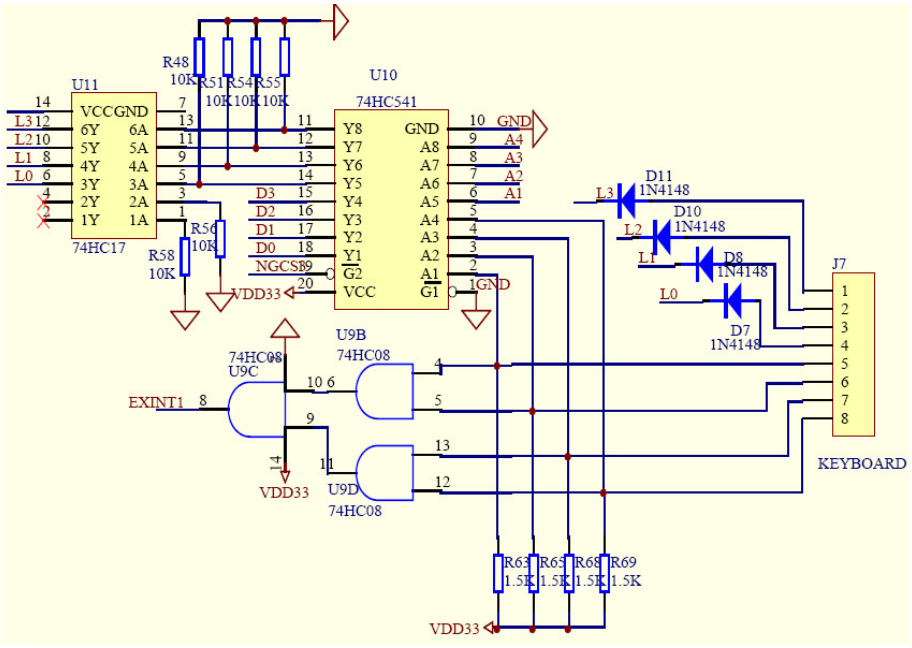

Figura 8.2: Conexión del teclado.

Los pines 5-8 del conector J7 están asociados a las columnas del teclado (véase Figura 8.1) y los pines 1-4 están asociados a las filas. Por otro lado, los pines 5-8 están conectados a resistencias de "*pull-up*" a fin de que, en ausencia de driver, la tensión en ellos sea una tensión de alta (VDD33). Estos pines, a través del circuito integrado U9[1], generan la señal EINT1 del controlador de interrupciones. Asimismo, estos pines están conectados a U10[2]. Las salidas de este CI se conectan a U11[3].

Los pines 1-4 están conectados a resistencias de "*pull-down*" para que, en ausencia de driver, la tensión en ellos sea una tensión de baja. No es una conexión directa sino que se realiza a través de varios componentes. Así, los pines 1-4 están conectados al ánodo de cuatro diodos (DN7/8/10/11)[4] y los cátodos de estos dispositivos están conectado a la señales L0-L3. Obsérvese que estas señales provienen de las salidas 3Y-6Y de U11 y las entradas correspondientes (3A-6A) están conectadas a resistencias de "*pull-down*". De esta forma cuando U10 esté deshabilitado (es decir, cuando $NGCS3 = \overline{G2} = 1$) sus salidas,

[1] 74HC08: es un CI que contiene puertas AND implementadas en tecnología CMOS y con compatibilidad con tecnología Schottky TTL.

[2] 74HC541A: array de 8 bufferes tri-estado no inversores diseñados para ser usados en buses. Internamente la señal output enable es una puerta AND de dos entradas complementadas ($\overline{G1}$ y $\overline{G2}$).

[3] 74HC17: array de 6 bufferes no inversores diseñados para ser usados con diodos.

[4] 1N4148: diodos de conmutación para aplicaciones de alta velocidad.

Y1-Y8, estarán a alta impedancia y por lo tanto las entradas 3A-6A de U11 estarán a una tensión de baja debido a las resistencias de "*pull-down*".

Por último, indicar que los pines Y1-Y4 de U10 están conectados a las líneas D0-D3 del bus de datos, mientras que los pines A5-A8 de U10 están conectados a las líneas A1-A4 del bus de direcciones.

8.3. Funcionamiento del teclado

Como ya mencionamos en la sección anterior, mientras $NGCS3 = 1$ los pines Y1-Y8 de U10 estarán a alta impedancia y los pines 3A-6A de U11 estarán a una tensión de baja debido a las resistencias de "*pull-down*". En consecuencia, las líneas L0-L3 tendrán una tensión de baja, los diodos DN7/8/10/11 estarán en conducción y en los pines 1-4 de J7 habrá una tensión de baja. Cuando una tecla es pulsada, el interruptor asociado cortocircuita la columna con la fila correspondiente, de forma que la columna tendrá una tensión de baja. Luego uno de los pines 5-8 tendrá una tensión de baja que, al propagarse a través de U9, provocará que la señal EINT1 tome el valor de baja. Si el puerto G y el controlador de interrupciones se configuran de manera adecuada (ver sección 5.2) este flanco de bajada desencadenará una interrupción por la línea IRQ.

Una vez la interrupción ha sido detectada, la rutina de tratamiento de interrupción debe identificar la tecla pulsada. En general, hay dos formas de hacerlo:

- *Scanning*: Esta técnica consiste en enviar una tensión de baja por una línea horizontal y una tensión alta por el resto de líneas horizontales. Cuando una línea vertical tome la tensión baja (tenga un 0 lógico), entonces la tecla que se encuentre en la intersección de ambas líneas, horizontal y vertical, será la tecla pulsada.

- *Inversión*: Esta técnica consiste en enviar una tensión de baja (0 lógico) sobre las líneas verticales y leer las líneas horizontales. Si una línea horizontal está a valor bajo indicará que una tecla ha sido pulsada en aquella fila. A continuación se repite el procedimiento sobre las líneas horizontales y se leen las líneas verticales. Si una línea vertical tomó el valor de tensión bajo entonces indicará que una tecla ha sido pulsada en dicha columna. La tecla situada en la intersección de la fila y la columna será la tecla pulsada.

En las prácticas se va a aplicar el método de "*scanning*", el único posible con las conexiones de la placa S3CEV40.

Para identificar la columna y la fila, y quedar así unívocamente identificada la tecla, tendremos que proceder al envío secuencial de un "0" por cada uno de los pines 1-4 de J7 (estando los restantes pines asociados a filas a "1") y a la lectura de los pines 5-8; cuando en alguno de ellos se reciba el "0" significará que la fila emisora tiene continuidad eléctrica con la columna receptora y, por consiguiente, la tecla situada en su intersección es la buscada.

El envío secuencial de un "0" por cada uno de los pines 1-4 de J7 se realizará escribiendo secuencialmente los patrones 4'b0111, 4'b1011, 4'b1101, 4'b1110 sobre las líneas A4-A1 del bus de direcciones. Para cada uno de estos patrones se procederá a la lectura de los pines 5-8, cuyos valores estarán en las líneas D0-D3 del bus de datos. Para que ello sea posible,

los valores escritos en el bus de direcciones deberán atravesar U10. U10 habilita sus salidas cuando sus dos entradas $\overline{G1} = \overline{G2} = 0$, luego se debe habilitar U10 haciendo $nGCS3 = 0$. La señal $nGCS3 = 0$ es generada por el controlador de memoria del S3C44B0X cuando se escribe en el bus de direcciones alguna dirección perteneciente al rango asociado al banco-3, 0x0600_0000-0x07FF_FFFF. En otras palabras, para escribir el patrón 4b'0111 en las líneas A4-A1 y que se propague por U10 es necesario escribir en el bus de direcciones la dirección 0x0600_00EF; para escribir el patrón 4'b1011 se debe escribir la dirección 0x0600_00F7, y así sucesivamente.

Una vez esté habilitado U10 los valores escritos en las líneas A1-A4 del bus de direcciones llegarán a los pines de salida 3Y-6Y de U11. Cuando haya un "1" en uno de los pines, el diodo al que está conectado dejará de conducir y por lo tanto la fila asociada estará "flotando" (no tendrá ni una tensión de alta, VDD, ni una tensión de baja, GND). Si la tecla pulsada se encuentra sobre esa fila entonces habrá continuidad eléctrica con la columna receptora pero ninguna de ellas (ni la fila ni la columna) está conectada a ninguna fuente de tensión por lo que el valor que habrá en el pin correspondiente de J7 vendrá impuesto por la resistencia de "*pull-up*". Es decir, sobre el bus de datos se observará un "1" sobre la línea correspondiente.

En cambio, aquel diodo conectado a un pin de U11 en el que el valor es "0", estará en conducción y, ahora sí, si la tecla está pulsada el pin de J7 asociado a la columna correspondiente tomará el valor "0" y este valor podrá observarse sobre el bus de datos.

En resumen, mientras se va procediendo al envío secuencial de un "0" por uno de los pines 1-4 de J7 (con el resto de pines 1-4 a "1") se deben observar los valores en el bus de datos. Mientras en el bus de datos esté el valor 0xF no se habrá identificado la tecla. Cuando en el bus de datos haya un valor "0" en alguna línea entonces ya tendremos identificada la fila y la columna donde está situada la tecla pulsada.

Desde el punto de vista del programador, toda la explicación anterior se puede ver como una operación de lectura sobre las posiciones de memoria 0x0600_00F7, 0x0600_00FB, 0x0600_00FD, 0x0600_00EF. Para cada una de esas lecturas hay que identificar si alguno de las bits 0-3 del dato devuelto está a "0". Cuando esto ocurra se habrá identificado la tecla pulsada.

La Tabla 8.1 muestra los valores que habrá que escribir en el bus de direcciones y leer del bus de datos cuando la tecla correspondiente haya sido pulsada. Mientras no haya identificación, en el bus de datos estará el dato 0xF.

Dirección	**Dato**				
	0x7	0xB	0xD	0xE	0xF
0xFD	SB1	SB5	SB9	SB13	-
0xFB	SB2	SB6	SB10	SB14	-
0xF7	SB3	SB7	SB11	SB15	-
0xEF	SB4	SB8	SB12	SB16	-

Tabla 8.1: Correspondencia entre dirección de lectura y dato leído para cada una de las teclas del teclado matricial.

Es importante darse cuenta de que durante el proceso de "*scanning*" se va a generar una nueva petición de interrupción para EINT1. Así, cuando se escriba un "0" sobre la fila

asociada a la tecla pulsada, este valor se propagará a través de la columna generando un "0" sobre alguno de los pines 5-8 de J7. Este valor se propagará a través de U9 generando una nueva transición a baja en EINT1 y por tanto una nueva petición de interrupción. Esta nueva petición no interrumpirá la ejecución de la RTI que se esté ejecutando en ese momento puesto que el ARM deshabilita de forma automática las interrupciones IRQ cuando está dando servicio a una interrupción IRQ (escribe CSPR[7] = "1") pero será atendida nada más finalizar la ejecución de RTI actual (puesto que al salir se recupera el CSPR que había antes de ejecutarse la RTI y por lo tanto CSPR[7]=0) . De esta forma, una única pulsación de tecla podría lanzar varias ejecuciones secuenciales de la misma RTI. Para evitar este problema el bit correspondiente a EINT1 en el registro de interrupciones pendientes (INTPND) debería borrarse justo antes de salir de la RTI.

A continuación se va a ilustrar el método de reconocimiento de tecla mediante un ejemplo. Supóngase que ha sido pulsada la tecla que conecta los pines 1 y 5 del conector J7 y que se ha dado servicio a la rutina de tratamiento de la interrupción. La RTI deberá escribir valores en el bus de direcciones de acuerdo a la Tabla 8.1 y se identificará la tecla cuando en los cuatro bits menos significativos del bus de datos se lea el valor indicado en dicha tabla:

- Lectura del contenido de la posición de memoria 0x0600_00FD (dirección dentro del rango de direcciones asignado al teclado; línea A1 del bus de direcciones a "0", líneas A2-A4 a "1"). Con este valor en el bus de direcciones el pin 1 de J7 está "flotando" (no está conectado a ninguna fuente de alimentación al estar el diodo D11 en corte). Como el pin 5 está cortocircuitado con el pin 1, entonces el pin 5 está únicamente conectado a VDD33 a través de la resistencia de "*pull-up*". Luego los pines 5-8 están todos a "1", el valor de los cuatro bits menos significativos del dato leído (proveniente de U10) es 0xF y la señal $EINT1 = 1$.

- Lectura del contenido de la posición de memoria 0x0600_00FB. Nuevamente el valor de los cuatro bits menos significativos del dato leído (proveniente de U10) es 0xF y la señal $EINT1 = 1$.

- Lectura del contenido de la posición de memoria 0x0600_00F7. El valor de los cuatro bits menos significativos del dato leído (proveniente de U10) es 0xF y la señal $EINT1 = 1$.

- Lectura del contenido de la posición de memoria 0x0600_00EF. Con este valor el pin 1 de J7 está a "0" (el diodo D11 está en conducción al tener el ánodo a una tensión de baja). Ahora el pin 5 tendrá un "0", la señal $EINT1 = 0$ y la salida en el bus de datos es 0xE. Este valor coincide con el indicado en la Tabla 8.1. Luego se ha pulsado la tecla SB16.

  Al identificar la tecla, y como un efecto colateral, la señal EINT1 toma el valor "0" generando una nueva petición de interrupción que no debería ser atendida al no obedecer a ninguna pulsación de tecla. Dicha petición no es atendida inmediatamente pero queda anotada en el registro INTPND. Es preciso recordar que para evitar que vuelva a ejecutarse la RTI es necesario borrar el bit correspondiente a EINT1 del registro INTPND justo antes de finalizar la RTI.

8.4. El problema de los rebotes

Cuando se pulsa una tecla, no se origina una transición instantánea y estable (podríamos decir "limpia") de niveles de tensión entre los bornes del pulsador, como correspondería a una conmutación ideal. Esto es debido a los rebotes. En la Figura 8.3 aparece un posible diagrama temporal de un proceso de pulsación: los flancos de presión y depresión de la tecla van seguidos de una serie de rebotes.

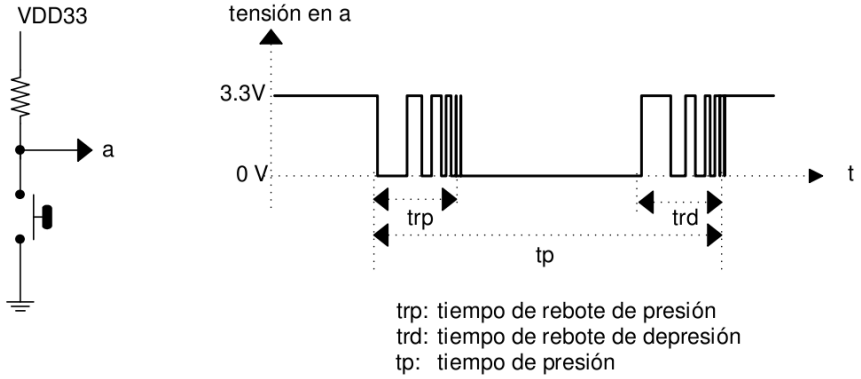

Figura 8.3: Fenómeno de rebotes al pulsar o liberar una tecla.

Para detectar correctamente una tecla es necesario esperar a que terminen los rebotes de presión. Una forma posible de hacerlo consiste en, después de haber detectado la pulsación (es decir una vez se da servicio a la RTI asociada), esperar durante un tiempo superior al trp (tiempo de rebote de presión) previsto, antes de proceder a la identificación de la tecla. Para este tipo de teclado supondremos $trp = 20ms$.

Asimismo, se debe proceder también a la eliminación de los rebotes de depresión. Como no conocemos el momento en el que el usuario va a dejar de pulsar la tecla, la forma más segura de evitar los rebotes de depresión consiste en detectar la transición de depresión, y sólo entonces esperar durante un tiempo superior al trd (tiempo de rebote de depresión) que permita eludir los rebotes de depresión antes de dar por finalizado el proceso de identificación. Para detectar la transición de depresión se esperará a que la línea EINT1 tome el valor "1" y después esperaremos $trd = 100ms$.

8.5. Teclado, interrupciones y pines E/S

Como podemos ver en la figura 8.2, cuando se pulsa una tecla con el teclado en estado de reposo se activa la línea EINT1. Como vimos con anterioridad, esta línea es accesible externamente a través del puerto G, que es como se conecta al circuito externo U9B. Para que la interrupción se genere correctamente el pin 1 del puerto G debe configurarse para activar la línea EINT1 cuando se genere un flanco de bajada por dicha línea.

Capítulo 9

Comunicación serie

La comunicación entre dos dispositivos puede ser en serie o en paralelo. La comunicación en paralelo utiliza varios cables en paralelo para transmitir simultáneamente un valor de varios bits. La comunicación digital serie se caracteriza porque la información viaja en un sólo cable, llamado línea de datos.

En la comunicación serie los bits se inyectan en la línea de datos y viajan secuencialmente, uno detrás de otro, del origen al destino. Además de la línea de datos pueden existir otras líneas de control y de alimentación. A su vez, la comunicación serie se divide en dos grandes grupos:

- *Síncrona.* En la comunicación serie síncrona los dos extremos utilizan una señal de reloj común. Esta señal se transmite normalmente del emisor al receptor por un cable adicional.

- *Asíncrona.* En la comunicación serie asíncrona cada sistema utiliza su propia señal de reloj, aunque los dos sistemas deben ponerse de acuerdo en la tasa de transferencia.

En la comunicación serie síncrona los dos sistemas se sincronizan al comienzo de la emisión. La detección de los bits se hace coincidir con los flancos de la señal de reloj. Suele permitir mayores tasas de transferencia aunque tiene el inconveniente de que la señal de reloj debe ser transmitida, limitando normalmente la distancia entre los dos equipos.

En la comunicación serie asíncrona se presenta el problema de la detección de bits. Aunque los dos sistemas se pongan de acuerdo en la frecuencia de transmisión, es prácticamente imposible que las frecuencias de reloj con las que transmiten sean iguales. Para empezar puede tratarse de equipos que funcionen con distintas frecuencias (por ejemplo PC e impresora serie) y sólo puedan generar, a partir de su reloj, una señal aproximada de la velocidad de transmisión (normalmente utilizando divisores de frecuencia como los utilizados en los temporizadores). Además, estos relojes pueden variar su frecuencia con el tiempo, por ejemplo por variaciones de temperatura. Pequeñas diferencias en esta frecuencia producen a la larga grandes desfases en la sincronización de los equipos. La única solución es limitar la cantidad de bits que podemos transmitir hasta forzar una nueva sincronización. Por eso muchos protocolos serie asíncronos se limitan al envío de pequeños paquetes. Estos paquetes reciben el nombre de tramas. Al comienzo de cada trama se

fuerza la sincronización entre los dos dispositivos. Para enviar varios bytes hay que enviar varias tramas.

La implementación habitual de un sistema de comunicación serie construye una señal de reloj para la transmisión que sea múltiplo de la señal del sistema, por ejemplo dividiendo la frecuencia de la señal del sistema entre 16. Esto permite muestrear la línea de datos para detectar el comienzo de una trama. El desfase desde que se detecta la trama hasta que se pone en marcha el sistema para la recepción será como mucho de uno o dos ciclos del reloj del sistema, lo que supone un desfase pequeño respecto de la frecuencia de transmisión.

9.1. Unidad UART del S3C44BOX

El sistema S3C44BOX dispone de dos dispositivos para la comunicación serie: un controlador de bus I^2C (síncrono) y una unidad UART (*Universal Asynchronous Receiver and Transmitter*).

La unidad UART proporciona dos puertos serie asíncronos independientes: UART0 y UART1, también llamados canales, que permiten una comunicación serie bidireccional de hasta 115.2 Kbps. La gestión de estos canales puede llevarse a cabo mediante encuesta (*polling*), interrupciones o DMA.

Cada canal consta de los siguientes elementos (ver figura 9.1):

- un generador de frecuencia (baudios).
- un módulo transmisor, con un registro de desplazamiento y un registro de buffer.
- un módulo receptor, con un registro de desplazamiento y un registro de buffer.
- una unidad de control.

El generador de frecuencia, que marca la duración de un bit (y por tanto la tasa de transferencia) emplea como entrada el reloj principal del sistema (MCLK), cuya frecuencia es dividida.

La transmisión se realiza a través de un registro de desplazamiento. Cuando el procesador quiere enviar un dato por la línea lo escribe en el buffer de emisión. Cuando la línea está lista el valor del registro se copia en el registro de desplazamiento. Este registro irá desplazando el contenido para que los bits vayan saliendo secuencialmente por la línea de datos, a la velocidad indicada por el generador de frecuencia. La recepción se realiza de forma similar por medio de otro registro de desplazamiento. La velocidad de transmisión/recepción es programable, modificando el divisor del generador de frecuencia.

Además, tanto el transmisor como el receptor incorporan una cola FIFO de 16 bytes. Si están activas, estas colas permiten desacoplar aún más la CPU y la comunicación de los datos por el puerto serie. En lugar de ir escribiendo/leyendo los caracteres uno a uno, la CPU escribe/lee bloques de hasta 16 caracteres. El contenido de la cola FIFO se va enviando por el puerto (o se recibe y se copia en la FIFO) de forma automática. La UART permite activar una interrupción cuando la cola FIFO de recepción sobrepasa un determinado umbral (o la de emisión se vacía por debajo de un umbral), de forma que el procesador tenga cierto margen para poder tratar la situación.

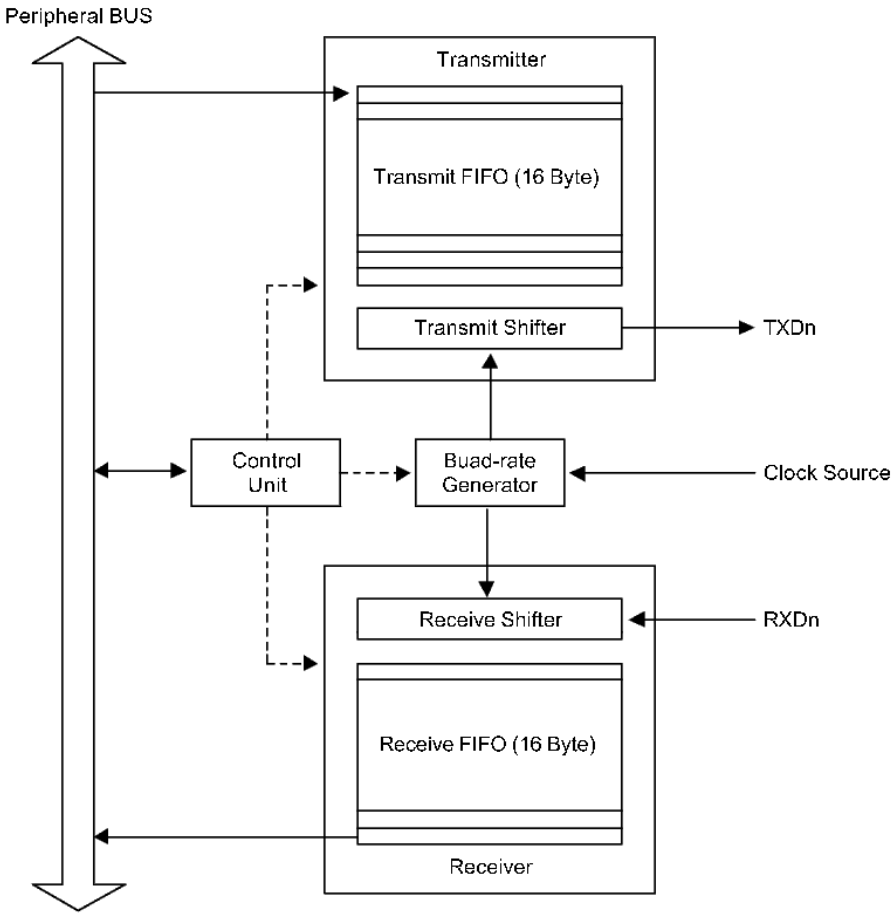

Figura 9.1: Estructura de un canal de la UART.

La comunicación a través de la línea serie se realiza carácter a carácter debido a la naturaleza asíncrona de la misma. Los registros de desplazamiento se encargan de ir transmitiendo o recibiendo los caracteres, bit a bit, por la línea correspondiente TXDn para transmisión o RXDn para recepción (donde n representa el nº del puerto).

Finalmente, la UART del S3C44BOX tiene capacidad de transmisión/recepción por infrarrojos siguiendo el estándar IrDA 1.0.

A continuación describiremos brevemente el funcionamiento de la UART. La información detallada se encuentra en el Capítulo 10 del Manual de Referencia del S3C44B0X [um-].

9.1.1. Formato de trama

El formato de trama (*frame*) enviado por la UART tiene la siguiente estructura:

- Un primer bit de *start*. Sirve para que el receptor detecte el comienzo de la trama y se sincronice. Tiene el valor contrario al valor de reposo de la línea, en nuestro caso el bit de *start* es cero.
- 5-8 bits de datos.
- 1 bit de paridad (opcional). Se pueden escoger paridad par o impar.
- 1-2 bits de stop, que tienen el valor de reposo de la línea.

La figura 9.2 muestra un ejemplo de trama con el siguiente formato: 1 bit de *Start*, carácter de 8 bits, 1 bit de *Stop* y sin bit de paridad.

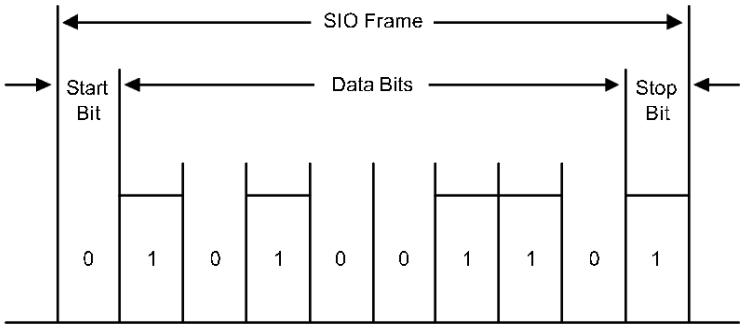

Figura 9.2: Diagrama temporal de una trama de datos serie (*SIO Frame*) con un tamaño de carácter de 8 bits, 1 bit de stop, sin paridad.

En la UART del S3C44BOX el formato de trama se configura mediante el registro de control de línea `ULCONn` (`0x01D00000` y `0x01D04000`).

El transmisor también puede producir un *frame* de pausa o *break* que consiste en una señal de 0 lógico de un frame de duración. El propósito de este tipo de marco es informar al receptor de una pausa en la comunicación.

Al igual que para la transmisión, el formato de la trama en recepción también es configurable. Como es lógico, para una correcta comunicación es necesario que la configuración en ambos dispositivos sea la misma.

9.1.2. Errores

El receptor es responsable de detectar varios tipos de error de comunicación:

- Error de superposición. Indica que un dato ha sido sobrescrito por otro más reciente antes de ser leído.
- Error de paridad. Indica que la paridad del dato recibido no concuerda con la esperada.
- Error de trama. Indica que el dato recibido no tiene un bit de *Stop* válido.
- Solicitud de *break*. Indica que se ha recibido una trama de pausa (el valor de la línea se ha mantenido inactivo por un tiempo mayor que el de una trama).
- Error de *timeout* (sólo para gestión por DMA usando colas FIFO). Indica que ha transcurrido un intervalo de tiempo de 3 tramas sin recibir ningún dato, y el nivel de ocupación no llega al umbral fijado para la transmisión. Esto impide que el sistema se quede esperando por nuevos datos que quizá no vayan a llegar.

9.1.3. Control de flujo

La UART del S3C44BOX admite control de flujo. En este caso el emisor debe esperar a que el receptor esté preparado para recibir. Esto se consigue añadiendo una línea adicional de control al puerto. Así la señal nCTS [1] (*Clear To Send*) del emisor se conecta con la señal nRTS (*Request To Send*) del receptor. Cuando éste está listo para recibir activa la señal nRTS. El emisor notará que se activa su señal nCTS y entonces podrá enviar el dato (ver figura 9.3).

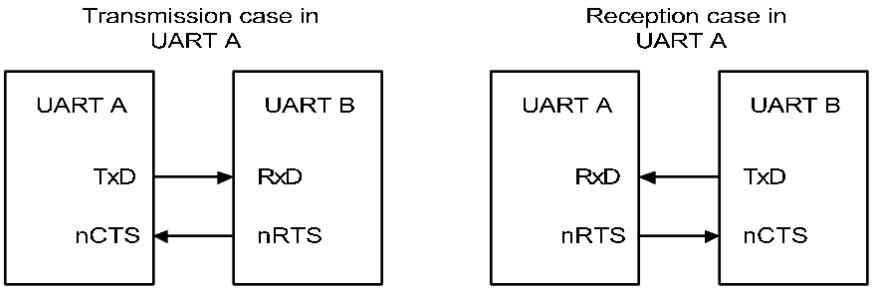

Figura 9.3: Conexión de dos UARTs empleando líneas de control de flujo.

En la UART del S3C44BOX la gestión de estas líneas puede llevarse a cabo de manera automática por HW (*Auto Flow Control*), o bien manualmente por SW, escribiendo en los registros de control y estado del módem (UMSTATn y UMCONn). Es preciso resaltar que el control de flujo automático sólo se puede emplear cuando se conectan dos UART entre sí. Para la conexión de un módem, es preciso usar el control SW, y, si se desea emplear un interfaz RS-232, es preciso usar además los puertos de E/S generales (ver figura 9.4).

[1] Es preciso destacar que aquí la "n" indica que la señal se activa a baja.

9.1.4. Velocidad en baudios

La velocidad de trabajo de la UART se establece dividiendo la señal de reloj principal (MCLK), 64MHz en nuestro caso, por un valor almacenado en el registro `UBRDIVn`. Dicho valor se obtiene mediante la fórmula:

$$UBRDIVn = (round\_off)(MCLK/(bps \times 16)) - 1$$

El resultado de esta división debe estar comprendido entre 1 y $2^{16} - 1$.

Por ejemplo si la velocidad que se desea es de 115200 bps y la frecuencia de MCLK fuese de 40 MHz:

$$\begin{aligned} UBRDIVn &= (int)(40000000/(115200 \times 16) + 0{,}5) - 1 \\ &= (int)(21{,}7 + 0{,}5) - 1 \\ &= 22 - 1 = 21 \end{aligned}$$

9.1.5. Modo loop-back

La UART del S3C44BOX proporciona un modo de test que podríamos denominar de lazo cerrado o *loop-back*, cuyo propósito es ayudar a aislar errores en la comunicación y comprobar la corrección del SW. En este modo, el dato enviado es inmediatamente recibido por la propia UART.

La activación de este modo de test se realiza mediante un bit del registro de control de la UART (UCONn).

9.1.6. Configuración de la UART

A continuación enunciaremos los principales registros de la UART, describiendo la funcionalidad de cada uno de ellos (para ampliar la descripción consultar el Manual de Referencia del S3C44BOX [um-]):

UART Line Control Register (ULCONn: 0x01D00000, 0x01D04000). Este registro se utiliza para la configuración de la línea de los puertos serie:

- Activar/desactivar modo IrDA.
- Configurar paridad (none/even/odd).
- Determinar bits de stop (1/2).
- Longitud del carácter (5/6/7/8).

La descripción del registro aparece en la tabla 9.1

Tabla 9.1: Registros de la UART ULCONn para configuración de la línea

ULCONn	Bit	Description
Reserved	[7]	
Infra-Red Mode	[6]	The Infra-Red mode determines whether or not to use the Infra-Red mode. 0 = Normal mode operation 1 = Infra-Red Tx/Rx mode
Parity Mode	[5:3]	The parity mode specifies how parity generation and checking are to be performed during UART transmit and receive operation. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/checked as 1 111 = Parity forced/checked as 0
Number of stop bit	[2]	The number of stop bits specifies how many stop bits are to be used to signal end-of-frame. 0 = One stop bit per frame 1 = Two stop bit per frame
Word length	[1:0]	The word length indicates the number of data bits to be transmitted or received per frame. 00 = 5-bits 01 = 6-bits 10 = 7-bits 11 = 8-bits

UART Control Register (UCONn: 0x01D00004, 0x01D04004). Este registro se utiliza para configurar el funcionamiento de la UART:

- Tipo de interrupción de envío/recepción (pulso/flanco).
- Activar interrupción por timeout.
- Activar interrupción por error (*break*, *frame*, *parity*, *overrun*).
- Activar modo *loop-back*.
- Mandar señal de *break*.
- Determinar modo de transmisión para envío/recepción (desactivado, interrupción/encuesta o DMA).

La descripción del registro aparece en la tabla 9.2

UART FIFO Control Register (UFCONn: 0x01D00008, 0x01D04008). Estos registros se utilizan para la configuración de las colas FIFO de la UART:

- Habilitar FIFO.
- Determinar el nivel de ocupación con el que se desencadenan las interrupciones de envío/recepción FIFO.

Tabla 9.2: Registros UCONn para configuración del funcionamiento de la UART

UCONn	Bit	Description
Tx interrupt type	[9]	Interrupt request type 0 = Pulse (Interrupt is requested the instant Tx buffer becomes empty) 1 = Level (Interrupt is requested while Tx buffer is empty)
Rx interrupt type	[8]	Interrupt request type 0 = Pulse (Interrupt is requested the instant Rx buffer receives the data) 1 = Level (Interrupt is requested while Rx buffer is receiving data)
Rx time out enable	[7]	Enable/Disable Rx time out interrupt when UART FIFO is enabled. The interrupt is a receive interrupt. 0 = Disable 1 = Enable
Rx error status interrupt enable	[6]	This bit enables the UART to generate an interrupt if an exception, such as a break, frame error, parity error, or overrun error occurs during a receive operation. 0 = Do not generate receive error status interrupt 1 = Generate receive error status interrupt
Loop-back Mode	[5]	Setting loop-back bit to 1 causes the UART to enter the loop-back mode. This mode is provided for test purposes only. 0 = Normal operation 1 = Loop-back mode
Send Break Signal	[4]	Setting this bit causes the UART to send a break during 1 frame time. This bit is auto-cleared after sending the break signal. 0 = Normal transmit 1 = Send break signal
Transmit Mode	[3:2]	These two bits determine which function is currently able to write Tx data to the UART transmit holding register. 00 = Disable 01 = Interrupt request or polling mode 10 = BDMA0 request (Only for UART0) 11 = BDMA1 request (Only for UART1)
Receive Mode	[1:0]	These two bits determine which function is currently able to read data from UART receive buffer register. 00 = Disable, 01 = Interrupt request or polling mode 10 = BDMA0 request (Only for UART0) 11 = BDMA1 request (Only for UART1)

Tabla 9.3: Registros UFCONn para configuración de las colas FIFO de la UART

UFCONn	Bit	Description	Initial State
Tx FIFO Trigger Level	[7:6]	These two bits determine the trigger level of transmit FIFO. 00 = Empty 01 = 4-byte 10 = 8-byte 11 = 12-byte	00
Rx FIFO Trigger Level	[5:4]	These two bits determine the trigger level of receive FIFO. 00 = 4-byte 01 = 8-byte 10 = 12-byte 11 = 16-byte	00
Reserved	[3]		0
Tx FIFO Reset	[2]	This bit is auto-cleared after resetting FIFO 0 = Normal 1= Tx FIFO reset	0
Rx FIFO Reset	[1]	This bit is auto-cleared after resetting FIFO 0 = Normal 1= Rx FIFO reset	0
FIFO Enable	[0]	0 = FIFO disable 1 = FIFO mode	0

Tabla 9.4: Registros UMCONn para configurar el control de flujo de la UART

UMCONn	Bit	Description
Reserved	[7:5]	These bits must be 0's
AFC(Auto Flow Control)	[4]	0 = Disable 1 = Enable
Reserved	[3:1]	These bits must be 0's
Request to Send	[0]	If AFC bit is enabled, this value will be ignored. In this case the S3C44B0X will control nRTS automatically. If AFC bit is disabled, nRTS must be controlled by S/W. 0 = 'H' level(Inactivate nRTS) 1 = 'L' level(Activate nRTS)

- Borrar FIFO.

La descripción del registro aparece en la tabla 9.3

UART MODEM Control Register (UMCONn: 0x01D0000C, 0x01D0400C). Registro utilizado para configurar el control de flujo de la UART:

- Habilitar control de flujo automático (AFC).
- Activar la señal RTS (Ready To Send) cuando el control de flujo es SW.

La descripción del registro aparece en la tabla 9.4

UART Tx/Rx Status Register (UTRSTATn: 0x01D00010, 0x01D04010). Registros de estado de transmisión y recepción, utilizados para:

- Determinar si el shifter de transmisión está vacío.
- Determinar si el buffer de transmisión está vacío.
- Determinar si el buffer de recepción tiene datos listos.

La descripción del registro aparece en la tabla 9.5.

Tabla 9.5: Registros UTRSTAT con información del estado de la transmisión y la recepción

UTRSTATn	Bit	Description
Transmit shifter empty	[2]	This bit is automatically set to 1 when the transmit shift register has no valid data to transmit and the transmit shift register is empty. 0 = Not empty 1 = Transmit holding & shifter register empty
Transmit buffer empty	[1]	This bit is automatically set to 1 when the transmit buffer register does not contain valid data. 0 =The buffer register is not empty 1 = Empty If the UART uses the FIFO, users should check Tx FIFO Count bits and Tx FIFO Full bit in the UFSTAT register instead of this bit.
Receive buffer data ready	[0]	This bit is automatically set to 1 whenever the receive buffer register contains valid data, received over the RXDn port. 0 = Completely empty 1 = The buffer register has a received data If the UART uses the FIFO, users should check Rx FIFO Count bits in the UFSTAT register instead of this bit.

UART Rx Error Status Register (UERSTATn: 0x01D00014, 0x01D04014). Determina el tipo de error que ha desencadenado la interrupción por error en la recepción. Su descripción aparece en la tabla 9.6.

UART FIFO Status Registers (UFSTATn: 0x01D00018, 0x01D04018). Registros de estado para las colas FIFO de la UART, utilizados para:

- Determinar si el FIFO de envío/recepción está lleno/vacío.
- Determinar el nº de elementos presentes en la cola FIFO.

La descripción del registro aparece en la tabla 9.7.

UART Modem Status Register (UMSTATn: 0x01D0001C, 0x01D0401C). Gestión de la señal CTS (*Clear To Send*) en el control de flujo SW. Su descripción aparece en la tabla 9.8.

UART Transmit Holding Register (UTXHn) y UART Receive Holding Register (URXHn: UTXH0, UTXH1). UTXHn son los registros en los que debemos escribir el dato que se desea transmitir. En URXHn podremos leer el dato recibido. Debemos resaltar que cuando se produce un error de superposición (*overrun*) se debe leer URXHn o de lo contrario, al recibir el siguiente dato, se volverá a producir un nuevo error. Las direcciones de estos registros están descritas en la tabla 9.9 y 9.10:

UART Baud Rate Division Register (UBRDIV: 0x01D00028, 0x01D04028). Permite establecer la frecuencia de comunicación. La fórmula para calcular el ratio y la velocidad en baudios se ha explicado anteriormente en la sección 9.1.4.

Tabla 9.6: Registros UERSTAT con información del estado de los errores en la recepción.

UERSTATn	Bit	Description
Break Detect	[3]	This bit is automatically set to 1 to indicate that a break signal has been received. 0 = No break receive 1 = Break receive
Frame Error	[2]	This bit is automatically set to 1 whenever a frame error occurs during receive operation. 0 = No frame error during receive 1 = Frame error
Parity Error	[1]	This bit is automatically set to 1 whenever a parity error occurs during receive operation. 0 = No parity error during receive 1 = Parity error
Overrun Error	[0]	This bit is automatically set to 1 whenever an overrun error occurs during receive operation. 0 = No overrun error during receive 1 = Overrun error

Tabla 9.7: Registros UFSTAT con información del estado de las colas FIFO de la UART.

UFSTATn	Bit	Description
Reserved	[15:10]	
Tx FIFO Full	[9]	This bit is automatically set to 1 whenever transmit FIFO is full during transmit operation 0 = 0-byte ≤ Tx FIFO data ≤ 15-byte 1 = Full
Rx FIFO Full	[8]	This bit is automatically set to 1 whenever receive FIFO is full during receive operation 0 = 0-byte ≤ Rx FIFO data ≤ 15-byte 1 = Full
Tx FIFO Count	[7:4]	Number of data in Tx FIFO
Rx FIFO Count	[3:0]	Number of data in Rx FIFO

Tabla 9.8: Registros UMSTAT para observar la línea CTS en el control de flujo software.

UMSTATn	Bit	Description
Delta CTS	[4]	This bit indicates that the nCTS input to S3C44B0X has changed state since the last time it was read by CPU. (Refer to Fig. 10-7) 0 = Has not changed 1 = Has changed
Reserved	[3:1]	Reserved
Clear to Send	[0]	0 = CTS signal is not activated(nCTS pin is high) 1 = CTS signal is activated(nCTS pin is low)

Tabla 9.9: Direcciones de los registros URXH.

Register	Address	R/W	Description	Reset Value
URXH0	0x01D00024(L) 0x01D00027(B)	R (by byte)	UART channel 0 receive buffer register	–
URXH1	0x01D04024(L) 0x01D04027(B)	R (by byte)	UART channel 1 receive buffer register	–

Tabla 9.10: Direcciones de los registros UTXH.

Register	Address	R/W	Description
UTXH0	0x01D00020(L) 0x01D00023(B)	W (by byte)	UART channel 0 transmit holding register
UTXH1	0x01D04020(L) 0x01D04023(B)	W (by byte)	UART channel 1 transmit holding register

9.1.7. Conexión de la UART a los puertos serie de la placa

La figura 9.4 muestra cómo se establece la conexión entre el S3C44BOX y los conectores DB9 de la placa S3CEV40. Como puede apreciarse en la figura, el conector UART0 dispone únicamente de dos líneas (RXD y TXD) y por lo tanto sólo puede emplearse para comunicaciones serie simples (sin control de flujo). El conector UART1 además de las líneas de transmisión y recepción, está conectado a 6 pines de E/S generales lo que permite que, con una gestión adecuada de los mismos, se pueda conectar a un modem RS-232.

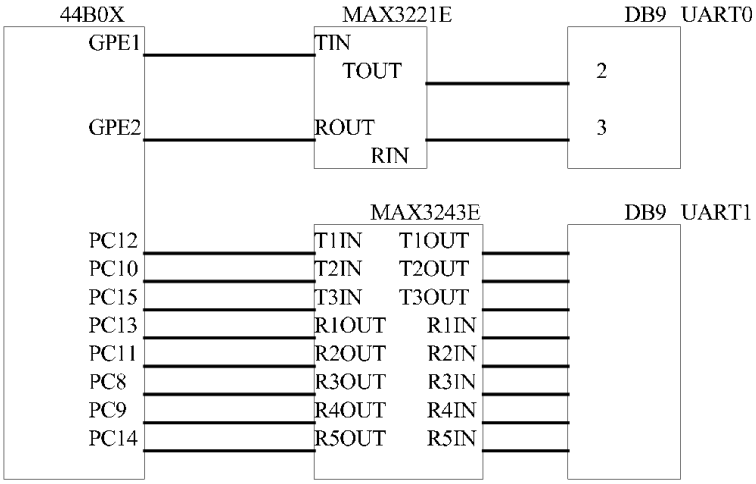

Figura 9.4: Conexión de la UART a los puertos serie de la placa (DB9).

Capítulo 10

Controlador de DMA

El S3C44B0X incorpora un controlador de *acceso directo a memoria* (DMA), que permite realizar transferencias memoria a memoria, memoria a periférico y periférico a memoria sin intervención del procesador. El controlador de DMA se encarga de solicitar el bus y luego controlarlo para que las transferencias se realicen de forma correcta.

El controlador de DMA del sistema tiene cuatro canales. Dos de ellos, ZDMA0 y ZDMA1, están conectados al bus de sistema. Los otros dos, BDMA0 y BDMA1, están localizados en el *bridge* que hace de interfaz con el bus de periféricos.

Los canales ZDMAn se utilizan para realizar transferencias de datos desde memoria a memoria, o entre la memoria y dispositivos de E/S mapeados en direcciones fijas de memoria. Los canales BDMAn están orientados a transferencias entre la memoria y los dispositivos de entrada/salida conectados al bus de periféricos, como SIO, IIS, timers o UART.

10.1. Programación de transferencias DMA

Programar una transferencia DMA es sencillo. Para ello debemos configurar el controlador DMA correspondiente con la dirección origen, la dirección destino y el número de bytes a transferir. Después, si la señalización es software debemos activar la petición DMA escribiendo en el registro correspondiente. Si la señalización es hardware la transferencia se realizará cuando el periférico active la petición. Además, de forma similar a como sucedía con los temporizadores, el controlador de DMA tiene un sistema de doble buffer. Al inicio, los registros de inicialización de direcciones y el contador de bytes a transferir se copian en los registros de trabajo. Mientras se realiza una transferencia DMA podemos programar la siguiente modificando los registros de inicialización. Debemos tener en cuenta que no basta con escribir los valores correctos en estos registros, el controlador no realizará la transferencia hasta que el periférico la solicite (asumiendo señalización hardware).

10.2. Funcionamiento del BDMA

En esta sección describiremos en detalle el controlador BDMA, cuyo diagrama aparece en la Figura 10.1. para ampliar la información sobre los controladores de DMA se aconseja consultar el manual del S3C44B0X [um-].

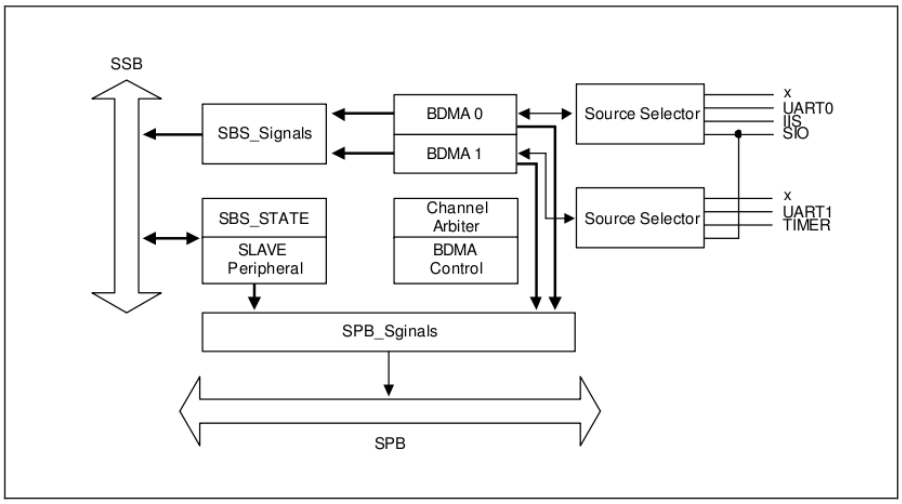

Figura 10.1: Diagrama de bloques del controlador BDMA.

El controlador BDMA sólo admite un protocolo de comunicación, *handshake* y un modo de transferencia, *Unit transfer*. El primero define cómo se activan/desactivan las señales de petición, *request*, y reconocimiento, *acknowledge*, en el bus. El segundo define la granularidad con la que se realizan las operaciones de lectura-escritura. Concretamente en el modo *Unit transfer* el controlador solicita el bus para realizar un par lectura-escritura y luego lo libera, como se indica en la figura 10.2. Cada lectura-escritura puede ser de tamaño byte, media palabra o palabra. El par lectura-escritura no puede ser interrumpido. Si el controlador de DMA tiene que hacer más transferencias vuelve a solicitar el bus.

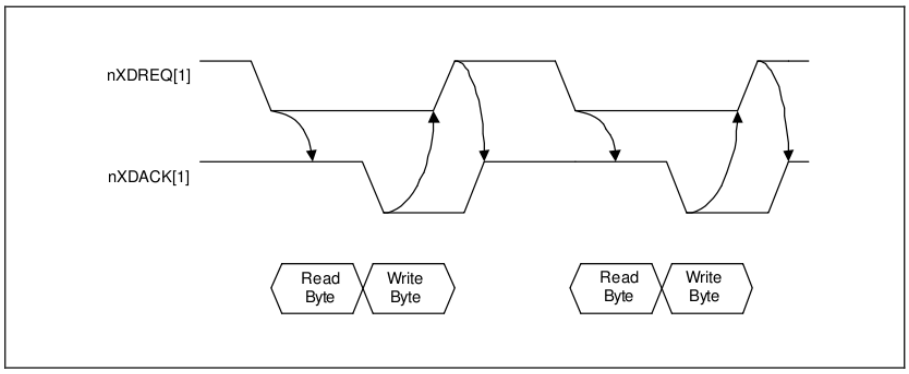

Figura 10.2: Cronograma del modo de *handshake-Unit transfer* utilizado por el controlador BDMA.

10.3. Configuración del controlador de BDMA

Como el resto de los controladores que hemos visto, el controlador de BDMA se configura a través de una serie de registros mapeados en memoria, que son:

- *BDCONn (0x01F80000, 0x01F80020)* Registro que nos permite comprobar el estado de la transferencia en curso, si ha terminado o no, habilitar o no las peticiones y realizar la señalización software. La descripción de estos registros se recoge en la figura 10.3.

BDCONn	Bit	Description	Initial State
INT	[7:6]	Reserved	00
STE	[5:4]	Status of DMA channel (Read only) 00 = Ready 01 = Not TC yet 10 = Terminal Count 11 = N/A Before the DMA counter decreases from a initial counter value, STE is still the ready state.	00
QDS	[3:2]	Disable/Enable External/Internal DMA request (UARTn, SIO, IIS, Timer) 00 = Enable Other = Disable	00
CMD	[1:0]	Software commands 00: No command. After writing 01, 10, 11, CMD bits are cleared automatically. 01: Reserved 10: Reserved 11: Cancels DMA operation.	00

Figura 10.3: Registros de control del controlador BDMA.

- *BDISRCn (0x01F80004, 0x01F80024) y BDCSRCn (0x01F80010,0x01F80030)* Sirven para especificar la dirección origen de la transferencia, si hay que incrementar o decrementar la dirección para completar la transferencia y el tamaño (byte, media palabra o palabra). El de inicialización (BDISRC) se copia en el actual (BDCSRC) al comienzo. La descripción de estos registros se da en la figura 10.4.

BDISRCn/BDCSRCn	Bit	Description	Initial State
DST	[31:30]	Data size for transfer 00 = Byte 01 = Half word 10 = Word 11 = Not used	00
DAL	[29:28]	Direction of address for load 00 = N/A 01 = Increment 10 = Decrement 11 = Internal peripheral (fixed address)	00
ISADDR/CSADDR	[27:0]	Initial/current source address for BDMAn. If the destination is the internal peripherals, the SFR address has to be used. For example, if the source is the UART0 Rx buffer, the UART0 Rx buffer address will be used.	0x0000000

Figura 10.4: Registros de configuración del origen de la transferencia.

- *BDIDESn (0x01F80008, 0x01F80028) y BDCDESn (0x01F80014, 0x01F80034)* Registros para configurar la dirección destino de la transferencia, la dirección (memoria a periférico, periférico a memoria o periférico a periférico) y modo en que hay que cambiar la dirección para completar la transferencia (incrementar o decrementar). La descripción de estos registros se da en la figura 10.5.

BDIDESn/BDCDESn	Bit	Description	Initial State
TDM	[31:30]	Transfer direction mode 00 = Reserved 01 = M2IO (from external memory to internal peripheral) 10 = IO2M (from internal peripheral to external memory) 11 = IO2IO (from internal peripheral to internal peripheral) **NOTE:** The initial value is '00', but you must change TDM value as another though the BDMA channel is unused.	00
DAS	[29:28]	Direction of address for store 00 = N/A 01 = Increment 10 = Decrement 11 = Internal peripheral (fixed address)	00
IDADDR/CDADDR	[27:0]	Initial/current destination address for BDMAn If the destination is the internal peripherals, the SFR address has to be used. For example, if the destination is UART0 Tx buffer, the UART0 Tx buffer address will be used.	0x0000000

Figura 10.5: Registros de configuración del destino de la transferencia.

- *BDICNT (0x01F8000C, 0x01F8002C) y BDCCNT (0x01F80018, 0x01F80038)* Permiten configurar el número de bytes a transmitir, habilitar o deshabilitar el propio DMA, activar el modo auto-reload, seleccionar la fuente y el modo de interrupciones. La descripción de estos registros aparece en la figura 10.6.

10.4. Funcionamiento del ZDMA

En esta sección describiremos en detalle el controlador ZDMA, cuyo diagrama aparece en la Figura 10.7. para ampliar la información sobre los controladores de DMA se aconseja consultar el manual del S3C44B0X [um-].

Aunque el BDMA puede ser utilizado para realizar transferencias memoria a memoria, es preferible utilizar para ello el ZDMA que tiene un buffer temporal para almacenar bloques de 16 bytes pudiendo realizar transferencias en modo *burst*. El controlador ZDMA permite cuatro protocolos de comunicación y 3 modos de transferencia. Los protocolos de comunicación son:

- *Modo handshake:* El DMA genera una señal de reconocimiento para cada petición de DMA individual. Hay una petición de DMA para cada transferencia de una palabra, media palabra o byte. Durante la operación de DMA (ciclo de lectura y escritura) el controlador del bus no deja a ningún otro bus master usar el bus, como en la figura 10.8.

- *Modo single step:* Hay dos ciclos de reconocimiento de DMA que indican el ciclo de lectura y el de escritura. Entre ellos se le puede conceder el uso del bus a otro master, como en la figura 10.9.

- *Modo whole service:* Se usa una única activación de la petición de DMA para que se ejecuten todas las operaciones de DMA necesarias hasta completar el número de transferencias especificadas en la cuenta. Para evitar que las operaciones de transferencia muy largas causen problemas a otros usuarios del bus, el DMA libera la propiedad del bus cada vez que termina una transferencia unitaria (byte, media palabra o palabra). Si hay otro master que se adueña del bus la parte de la operación

BDICNT0/BDCCNT0	Bit	Description	Initial State
QSC	[31:30]	DMA request source selection 00 = N/A 01 = IIS 10 = UART0 11 = SIO	00
Reserved	[29:28]	00: handshake mode	00
Reserved	[27:26]	01: unit transfer mode	01
Reserved	[25:24]	00: on-the-fly mode is not supported in BDMAn	00
INTS	[23:22]	Interrupt mode set 00 = Polling mode 01 = N/A 10 = Int. whenever transferred 11 = Int. whenever terminated count	00
AR	[21]	Auto-reload and Auto-start after DMA count are 0. 0= Disable 1= Enable. Even after DMA count is 0, the DMA H/W enable bit (EN bit) is still 1. But, DMA will start to operate only if the start command or DMA request is activated	0
EN	[20]	DMA H/W enable/disable 0 = Disable DMA 1 = Enable DMA. If the QDS bit is 00b, DMA request can be serviced. Also if the S/W command is started, the DMA operation will occur. If the EN bit is 0, DMA will not operate even though S/W command is started. If the S/W command is canceled, the DMA operation will be canceled and EN bit will be cleared to 0. At the terminal count, the EN bit will be cleared to 0. **NOTE:** Do not set the EN bit and the other bits of BDICNT register at the same time. User have to set EN bit after setting the other bits of BDICNT register as following steps, 1. Set BDICNT register with disabled En bit. 2. Set EN bit enable.	0
ICNT/CCNT	[19:0]	Transfer count for BDMA0. The transfer count must be right value. For example, if DST is word, ICNT must be 4n. If 1 byte is transferred, the ICNT will be decreased by 1. If 1 half-word is transferred, the ICNT will be decreased by 2. If 1 word is transferred, the ICNT will be decreased by 4.	0x00000

Figura 10.6: Registros de configuración de la cuenta.

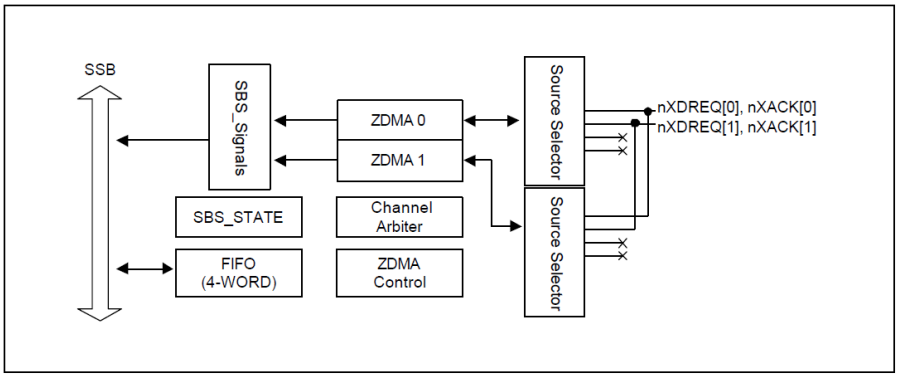

Figura 10.7: Diagrama de bloques del controlador ZDMA.

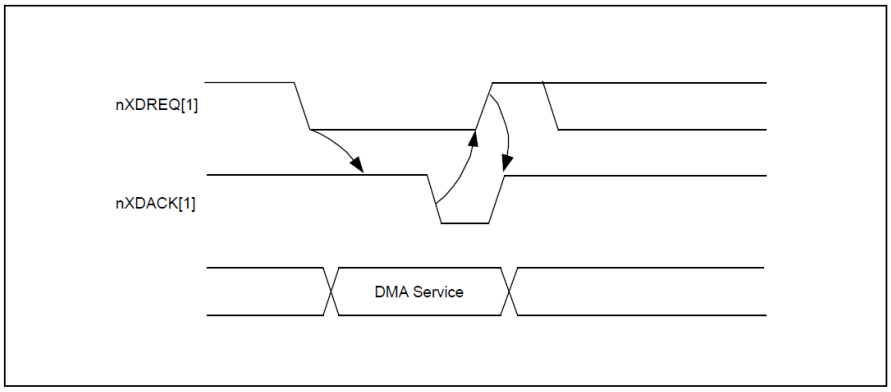

Figura 10.8: Diagrama del modo handshake .

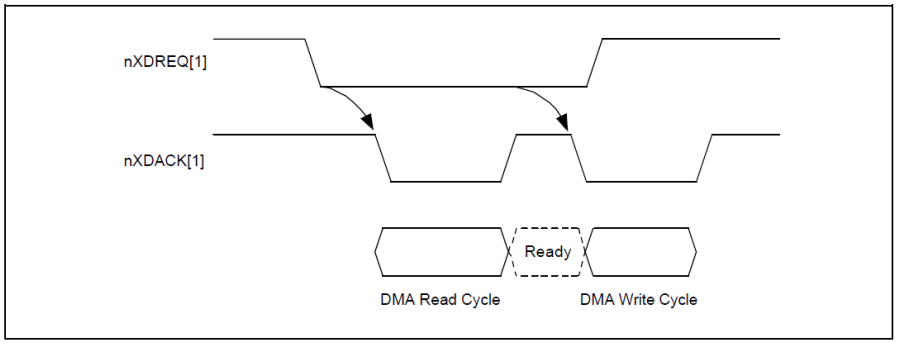

Figura 10.9: Diagrama del modo single step.

de DMA sin ejecutar prosigue cuando el otro master libera el bus sin necesitad de una re-activación de la petición de DMA, como en la figura 10.10.

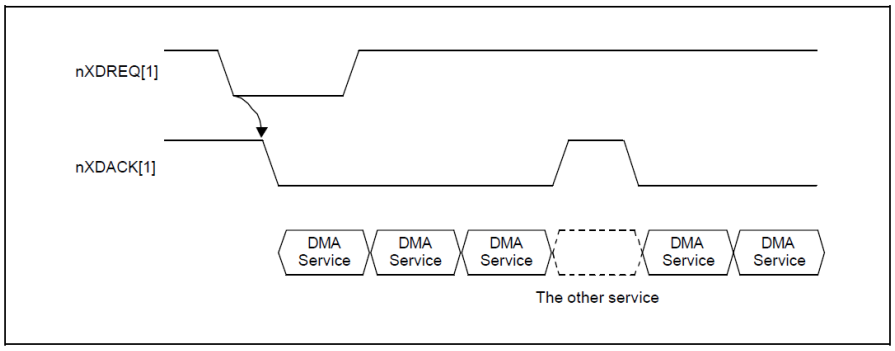

Figura 10.10: Diagrama del modo whole service cuando otro master se adueña del bus.

- *Modo demand:* Implica que hay ciclos de transferencia DMA continuos tanto tiempo como la señal de petición de DMA esté activa. Ningún otro master puede tener control del bus durante ese tiempo.

Los modos de transferencia definen el número de lecturas y escrituras durante una transferencia unidad, tal como se indica a continuación:

- *Transferencia unidad:* hay un ciclo de lectura y su ciclo de escritura correspondiente para cada petición de DMA. Por ejemplo, para el modo handshake, la figura 10.11.

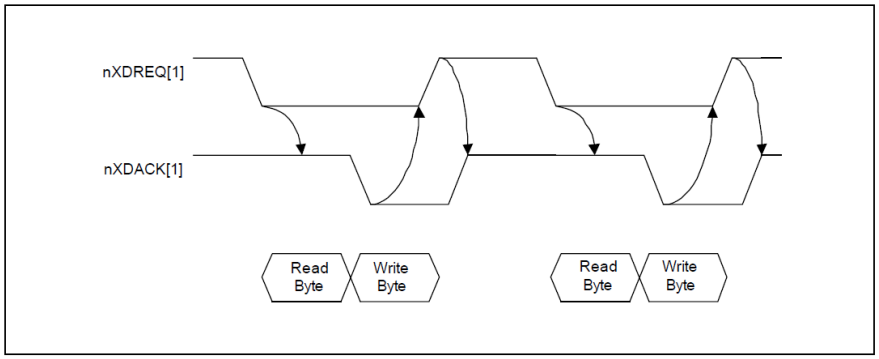

Figura 10.11: Modo de transferencia unidad con protocolo handshake.

- *Transferencia de bloque:* hay un ciclo de lectura de 4 palabras consecutivas seguido de un ciclo de escritura de 4 palabras. Por ejemplo, para el modo single step, la figura 10.12. El tamaño total de los datos a transferir debe ser múltiplo de 16 bytes, que es la mínima unidad de transferencia en este modo. Los cuatro bits menos significativos de la dirección deben ser cero para respetar el alineamiento a 16 bytes.

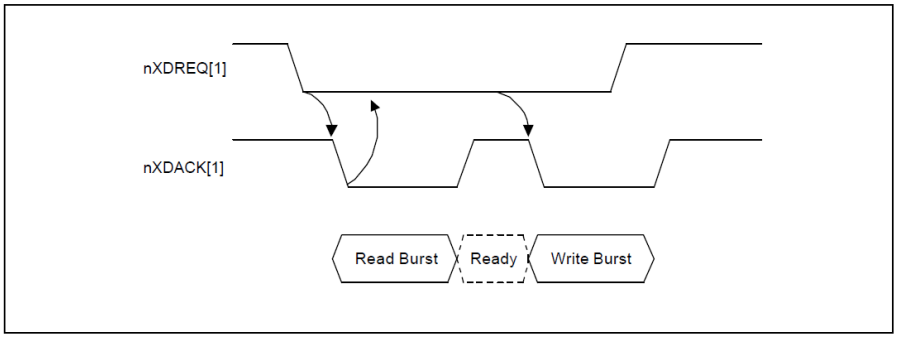

Figura 10.12: Modo de transferencia de bloque con protocolo single step.

- *Transferencia on-the-fly:* cuando el DMA lee (escribe) un dato, un dispositivo externo de dirección fija escribe (lee). La operación de lectura y la de escritura ocurren simultáneamente. Por ejemplo, para whole service mode ver figura 10.13.

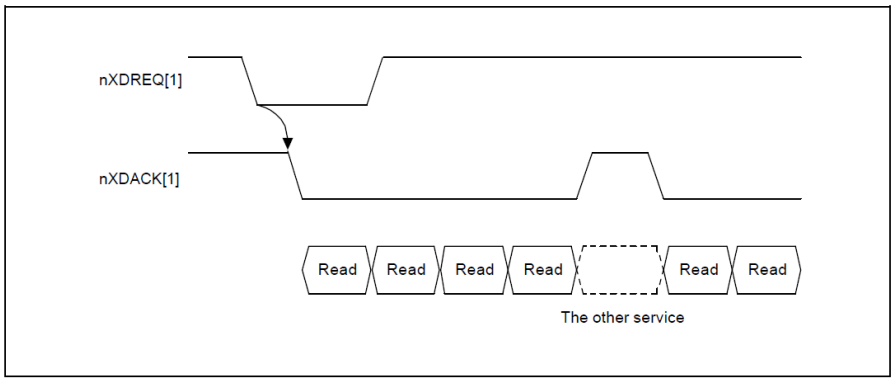

Figura 10.13: Modo de transferencia on-the-fly con protocolo whole service.

10.5. Configuración del controlador de ZDMA

Como el resto de los controladores que hemos visto, el controlador de ZDMA se configura a través de una serie de registros mapeados en memoria, que son:

- *ZDCONn (0x01E80000, 0x01E80020)* Registro que nos permite comprobar el estado de la transferencia en curso, si ha terminado o no, habilitar o no las peticiones y realizar la señalización software. La descripción de estos registros se recoge en la figura 10.14.

ZDCONn	Bit	Description	Initial State
INT	[7:6]	Reserved	00
STE	[5:4]	Status of DMA channel (Read only) 00 = Ready 01 = Not TC yet 10 = Terminal Count 11 = N/A Before the DMA counter decreases from the initial counter value, STE is still in the ready state.	00
QDS	[3:2]	Disable/Enable External DMA request (nXDREQ) 00 = Enable other = Disable	00
CMD	[1:0]	Software commands 00: No command. After writing 01,10,11, CMD bit is cleared automatically. nXDREQ is available. 01: Starts DMA operation by S/W without nXDREQ. S/W start function can be used only in the whole mode. As DMA is in the whole mode, the DMA will operate until the counter is 0. If nXDREQ is used , this command must not be issued. 10: Pauses DMA operation. But nXDREQ is still available. 11: Cancels DMA operation.	00

NOTE: If users start the ZDMA operation by CMD=01b, the DREQ protocol must be whole service mode.

Figura 10.14: Registros de control del controlador ZDMA.

- *ZDISRCn (0x01E80004, 0x01E80024) y ZDCSRCn (0x01E80010,0x01E80030)* Sirven para especificar la dirección origen de la transferencia, si hay que incrementar o decrementar la dirección para completar la transferencia y el tamaño (byte, media palabra o palabra). El de inicialización (ZDISRC) se copia en el actual (ZDCSRC) al comienzo. La descripción de estos registros se da en la figura 10.15.

- *ZDIDESn (0x01E80008, 0x01E80028) y ZDCDESn (0x01E80014, 0x01E80034)* Registros para configurar la dirección destino de la transferencia, opciones internas del DMA y el modo en que hay que cambiar la dirección para completar la transferencia (incrementar o decrementar). La descripción de estos registros se da en la figura 10.16.

- *ZDICNT (0x01E8000C, 0x01E8002C) y ZDCCNT (0x01E80018, 0x01E80038)* Permiten configurar el número de bytes a transmitir, seleccionar el protocolo y modo de transferencia, habilitar o deshabilitar el propio DMA, activar el modo auto-reload, seleccionar la fuente y el modo de interrupciones. La descripción de estos registros aparece en la figura 10.17.

ZDISRCn/ZDCSRCn	Bit	Description	Initial State
DST	[31:30]	Data size for transfer 00 = Byte, 01 = Half word 10 = Word, 11 = Not used If the block transfer mode is used, the DST must be 10.	00
DAL	[29:28]	Direction of address for load 00 = N/A, 01 = Increment 10 = Decrement, 11 = Fixed	00
ISADDR/CSADDR	[27:0]	Initial/current source address for ZDMAn	0x0000000

Figura 10.15: Registros de configuración del origen de la transferencia.

ZDIDESn/ZDCDESn	Bit	Description	Initial State
OPT	[31:30]	DMA internal options. OPT = 10 is recommended. bit 31: Indicates how nXDREQ is sampled in the single step mode. 1 is recommended. bit 30: If the DST is half-word or word and if the DMA mode is not the block transfer mode, this bit takes a role. 1: DMA does word-swap or half-word swap Before transfer: B0,B1,B2,B3,B4,B5,B6,B7... word-swapped data: B3,B2,B1,B0,B7,B6,B5,B4,... half-word-swapped data: B1,B0,B3,B2,B5,B4,B7,B6,... 0: normal	00
DAS	[29:28]	Direction of address for store 00 = N/A 01 = Increment 10 = Decrement 11 = Fixed	00
IDADDR/CDADDR	[27:0]	Initial/current destination address for ZDMAn	0x0000000

Figura 10.16: Registros de configuración del destino de la transferencia.

ZDICNTn/ZDCCNTn	Bit	Description	Initial State
QSC	[31:30]	DREQ(DMA request) source selection 00 = nXDREQ[0] 01 = nXDREQ[1] 10 = N/A 11 = N/A	00
QTY	[29:28]	DREQ protocol 00 = Handshake 01 = Single step 10 = Whole Service 11 = Demand	00
TMD	[27:26]	Transfer mode 00 = Not used 01 = Unit transfer mode 10 = Block(4-word) transfer mode 11 = On the fly If block transfer mode is selected, the ADDR[3:0] should be '0' to meet 16-byte align condition.	00
OTF	[25:24]	On the fly mode 00 = N/A 01 = N/A 10 = Read time on the fly 11 = Write time on the fly	00
INTS	[23:22]	Interrupt mode set 00 = Polling mode 01 = N/A 10 = Int. whenever transferred 11 = Int. whenever terminated count	00
AR	[21]	Auto-reload and Auto-start after DMA count are 0. 0 = Disable 1 = Enable. Even after DMA count is 0, the DMA H/W enable bit (EN bit) is still 1. But, DMA will start to operate only if the start command or nXDREQ is activated.	0
EN	[20]	DMA H/W enable/disable 0 = Disable DMA 1 = Enable DMA. If the QDS bit is 00b, DMA request can be serviced. Also if the S/W command is started, the DMA operation will occur. If the EN bit is 0, DMA will not operate even though S/W command is started. If the S/W command is canceled, the DMA operation will be canceled and EN bit will be cleared to 0. At the terminal count, the EN bit will be cleared to 0. **NOTE:** Do not set the EN bit and the other bits of ZDICNT register at the same time. User have to set EN bit after setting the other bits of ZDICNT register as following steps, 1. Set ZDICNT register with disabled En bit. 2. Set EN bit enable.	0
ICNT/CCNT	[19:0]	Initial/current transfer count for ZDMAn. If 1 byte is transferred, the ICNT will be decreased by 1. If 1 half-word is transferred, the ICNT will be decreased by 2. If 1 word is transferred, the ICNT will be decreased by 4. For example, if the data size of a transfer is word and the count is 4n+3, the last 3 bytes will not be transferred.	0x00000

Figura 10.17: Registros de configuración de la cuenta.

Bibliografía

[arm] Arm architecture reference manual. Accesible en http://www.arm.com/miscPDFs/14128.pdf. Hay una copia en el campus virtual.

[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.