CAPÍTULO 8

Entrada/Salida: dispositivos

Índice

8.1. Entrada/salida de propósito general (GPIO - General Purpose Input Output)

La forma más sencilla de entrada/salida que podemos encontrar en un procesador son sus propios pines de conexión eléctrica con el exterior. Si la organización del procesador permite relacionar direcciones del mapa de memoria o de entrada salida con algunos pines, la escritura de un `1` o `0` lógicos por parte de un programa —arquitectura— en esas direcciones se reflejará en cierta tensión eléctrica en el pin, normalmente 0V para el nivel bajo y 5 o 3,3V para el alto, que puede ser utilizada para activar

o desactivar algún dispositivo externo. Por ejemplo, esto nos permitiría encender o apagar un LED mediante instrucciones de nuestro programa. De modo análogo, en el caso de las entradas, si el valor eléctrico presente en el pin se ve traducido por el diseño eléctrico del circuito en un `1` o `0` lógico que se puede leer en una dirección del sistema, podremos detectar cambios en el exterior de nuestro procesador. De esta manera, por ejemplo, nuestro programa podrá consultar si un pulsador está libre u oprimido, y tomar decisiones en función de su estado.

Veámoslo en un sencillo ejemplo:

ej-entrada-salida.s

```
1        ldr     r0, [r7, #PULSADOR]     @ Leemos el nivel
2        cmp     r0, #1                  @ Si no está pulsado
3        bne     sigue                   @ seguimos
4        mov     r0, #1                  @ Escribimos 1 para
5        str     r0, [r7, #LED]          @ encender el LED
```

El fragmento de código anterior supuestamente enciende un LED escribiendo un `1` en la dirección «`r7 + LED`» si el pulsador está presionado, es decir, cuando lee un `1` en la dirección «`r7 + PULSADOR`». Es un ejemplo figurado que simplifica el caso real. El apartado siguiente profundiza en la descripción de la *GPIO* (*General Purpose Input/Ouput* en inglés) y describe con más detalle sus características en los sistemas reales.

8.1.1. La GPIO en la E/S de los sistemas

La GPIO (*General Purpose Input Output*) es tan útil y necesaria que está presente en todos los sistemas informáticos. Los PC actuales la utilizan para leer pulsadores o encender algún LED del chasis. Por otra parte, en los microcontroladores, sistemas completos en un chip, la GPIO tiene más importancia y muestra su mayor complejidad y potencia. Vamos a analizar a continuación los aspectos e implicaciones de la GPIO y su uso en estos sistemas.

Aspectos lógicos y físicos de la GPIO o Programación y electrónica de la GPIO

En el ejemplo que hemos visto antes se trabajaba exclusivamente con un bit, que se corresponde con un pin del circuito integrado, tanto en entrada como en salida, utilizando instrucciones de acceso a una palabra de memoria, 32 bits en la arquitectura ARM. En la mayor parte de los sistemas, los diversos pines de entrada/salida se agrupan en *palabras*, de tal forma que cada acceso como los del ejemplo afectaría a todos los pines asociados a la palabra a cuya dirección se accede. De esta manera, se habla de *puertos* refiriéndose a cada una de las direcciones

asociadas a conjuntos de pines en el exterior del circuito, y cada pin individual es un bit del puerto. Así por ejemplo, si hablamos de *PB12* —en el microcontrolador ATSAM3X8E— nos estamos refiriendo al bit 12 del puerto de salida llamado PB —que físicamente se corresponde con el pin 86 del encapsulado LQFP del microcontrolador, algo que es necesario saber para diseñar el hardware del sistema—. En este caso, para actuar —modificar o comprobar su valor— sobre bits individuales o sobre conjuntos de bits es necesario utilizar máscaras y operaciones lógicas para no afectar a otros pines del mismo puerto. Suponiendo que el LED se encuentra en el bit 12 y el pulsador en el bit 20 del citado puerto PB, una versión más verosímil del ejemplo propuesto sería:

ej-acceso-es.s

```
1        ldr     r7, =PB         @ Dirección del puerto
2        ldr     r6, =0x00100000 @ Máscara para el bit 20
3        ldr     r0, [r7]        @ Leemos el puerto
4        ands    r0, r6          @ y verificamos el bit
5        beq     sigue           @ Seguimos si está a 0
6        ldr     r6, =0x00001000 @ Máscara para el bit 12
7        ldr     r0, [r7]        @ Leemos el puerto
8        orr     r0, r6          @ ponemos a 1 el bit
9        str     r0, [r7]        @ y lo escribimos en el puerto
```

En este caso, en primer lugar se accede a la dirección del puerto PB para leer el estado de todos los bits y, mediante una máscara y la operación lógica *AND*, se verifica si el bit correspondiente al pulsador —bit 20— está a `1`. En caso afirmativo —cuando el resultado de AND no es cero— se lee de nuevo PB y mediante una operación OR y la máscara correspondiente se pone a `1` el bit 12, correspondiente al LED para encenderlo. La operación OR permite, en este caso, poner a `1` un bit sin modificar los demás. Aunque este ejemplo es más cercano a la realidad y sería válido para muchos microcontroladores, el caso del ATSAM3X8E es algo más complejo, como se verá en su momento.

Obviando esta complejidad, el ejemplo que se acaba de presentar es válido para mostrar la gestión por programa de la entrada y salida tipo GPIO. Sin embargo, es necesario que nos surja alguna duda al considerar —no lo olvidemos— que los pines se relacionan realmente con el exterior mediante magnitudes eléctricas. Efectivamente, el comportamiento eléctrico de un pin que funciona como entrada es totalmente distinto al de otro que se utiliza como salida, como veremos más adelante. Para resolver esta paradoja volvemos a hacer hincapié en que el ejemplo que se ha comentado es de gestión de la entrada/salida durante el funcionamiento del sistema, pero no se ha querido comentar, hasta ahora, que previamente hace falta una configuración de los puertos de la GPIO en que se indique qué pines van a actuar como entrada y cuáles como salida. Así

pues, asociado a la dirección en la que se leen o escriben los datos y que hemos llamado *PB* en el ejemplo, habrá al menos otra que corresponda a un registro de control de la GPIO en la que se indique qué pines se comportan como entradas y cuáles como salidas, lo que se conoce como *dirección de los pines*. Consideremos de nuevo el hecho diferencial de utilizar un pin —y su correspondiente bit en un puerto— como entrada o como salida. En el primer caso son los circuitos exteriores al procesador los que determinan la tensión presente en el pin, y la variación de ésta no depende del programa, ni en valor ni en tiempo. Sin embargo, cuando el pin se usa como salida, es el procesador ejecutando instrucciones de un programa el que modifica la tensión presente en el pin al escribir en su bit asociado. Se espera además —pensemos en el LED encendido— que el valor se mantenga en el pin hasta que el programa decida cambiarlo escribiendo en él otro valor. Si analizamos ambos casos desde el punto de vista de necesidad de almacenamiento de información, veremos que en el caso de la entrada nos limitamos a leer un valor eléctrico en cierto instante, valor que además viene establecido desde fuera y no por el programa ni el procesador, mientras que en la salida es necesario asociar a cada pin un espacio de almacenamiento para que el `1` o `0` escrito por el programa se mantenga hasta que decidamos cambiarlo, de nuevo de acuerdo con el programa. Esto muestra por qué la GPIO a veces utiliza dos puertos, con direcciones distintas, para leer o escribir en los pines del sistema. El registro —o *latch*— que se asocia a las salidas suele tener una dirección y las entradas —que no requieren registro pues leen el valor lógico fijado externamente en el pin— otra. Así, en el caso más común, un puerto GPIO ocupa al menos tres direcciones en el mapa: una para el registro de control que configura la dirección de los pines, otra para el registro de datos de salida y otra para leer directamente los pines a través del registro de datos de entrada. En la Figura 8.1 (obtenida del manual [Atm11]) se muestra la estructura interna de un pin de E/S de un microcontrolador de la familia Atmel AVR.

En este caso, que como hemos dicho es muy común, ¿qué ocurre si escribimos en un pin configurado como entrada, o si leemos un pin configurado como salida? Esto depende en gran medida del diseño electrónico de los circuitos de E/S del microcontrolador, pero en muchos casos el comportamiento es el siguiente: si leemos un pin configurado como salida podemos leer bien el valor almacenado en el registro, bien el valor presente en el pin. Ambos deberían coincidir a nivel lógico, salvo que algún error en el diseño del circuito o alguna avería produjeran lo contrario. Por ejemplo, en un pin conectado a masa es imposible que se mantenga un `1` lógico. Por otra parte, si escribimos en un pin configurado como entrada es común que, en realidad, se escriba en el registro de salida, sin modificar el valor en el pin. Este comportamiento es útil, dado que permite fijar un valor lógico conocido en un pin, antes de configurarlo

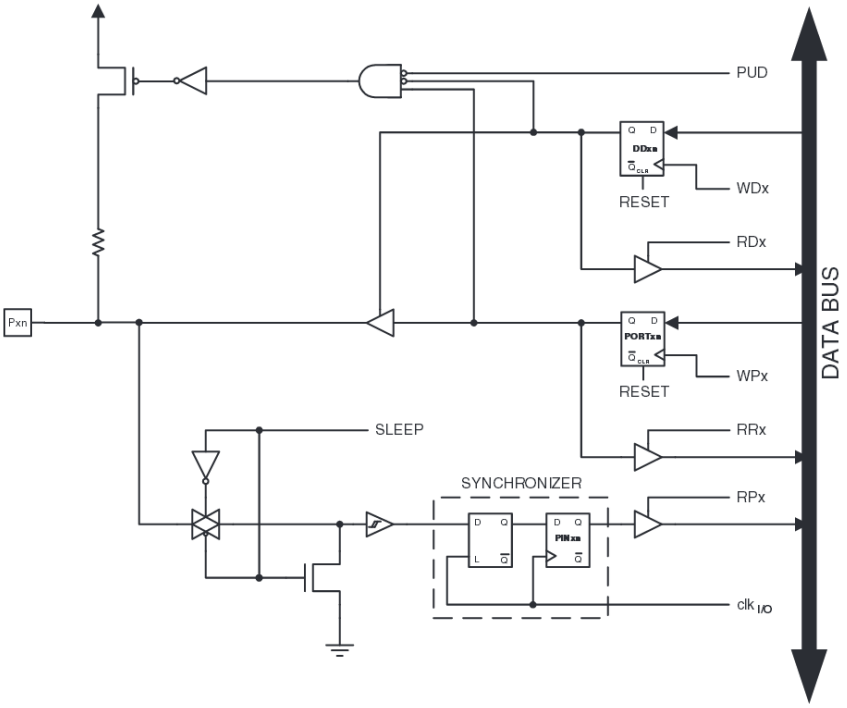

Figura 8.1: Estructura interna de un pin de E/S de un microcontrolador de la familia Atmel AVR

como salida. Dado que las entradas son eléctricamente más seguras, los pines suelen estar configurados como tales tras el reset del sistema. Así, el procedimiento normal para inicializar un pin de salida es escribir su valor mientras está configurado como entrada, y luego configurarlo como salida. Esto permite además fijar valores lógicos en el exterior mediante resistencias, que si son de valor elevado permitirán posteriormente el funcionamiento normal del pin como salida.

Para comprender adecuadamente esta última afirmación, vamos a estudiar brevemente las características eléctricas de los pines de entrada/salida. Como hemos dicho, la forma de interactuar con el exterior de un pin de E/S es típicamente mediante una tensión eléctrica presente en él. En el caso de las salidas, cuando escribimos un `1` lógico en el registro se tendrá un cierto voltaje en el pin correspondiente, y otro distinto cuando escribimos un `0`. En el de las entradas, al leer el pin obtendremos un `1` o un `0` según la tensión fijada en él por la circuitería externa.

Tratemos en primer lugar las salidas, y consideremos el caso más común hoy en día de lógica positiva —las tensiones de los `1` lógicos son mayores que las de los `0`—. Las especificaciones eléctricas de los

circuitos indican típicamente un valor mínimo, `VOHMIN`, que especifica la mínima tensión que vamos a tener en dicho pin cuando escribimos en él un `1` lógico. Se especifica solo el valor mínimo porque se supone que el máximo es el de alimentación del circuito. Por ejemplo 5V de alimentación y 4,2V como `VOHMIN` serían valores razonables. Estos valores nos garantizan que la tensión en el pin estará comprendida entre 4,2 y 5V cuando en él tenemos un `1` lógico. De manera análoga se especifica `VOLMAX` como la mayor tensión que podemos tener en un pin cuando en él escribimos un `0` lógico. En este caso, la tensión mínima es 0 voltios y el rango garantizado está entre `VOLMAX`, por ejemplo 0,8V, y 0V. En las especificaciones de valores anteriores, `V` indica voltaje, `O` salida (*output*), `H` y `L` se refieren a nivel alto (*high*) y bajo (*low*) respectivamente, mientras que `MAX` y `MIN` indican si se trata, como se ha dicho, de un valor máximo o mínimo. Inmediatamente veremos cómo estas siglas se combinan para especificar otros parámetros.

El mundo real tiene, sin embargo, sus límites, de tal modo que los niveles de tensión eléctrica especificados requieren, para ser válidos, que se cumpla una restricción adicional. Pensemos en el caso comentado antes en que una salida se conecta directamente a masa —es decir, 0V—. La especificación garantiza, según el ejemplo, una tensión mínima de 4,2V, pero sabemos que el pin está a un potencial de 0V por estar conectado a masa. Como la resistividad de las conexiones internas del circuito es despreciable, la intensidad suministrada por el pin, y con ella la potencia disipada, debería ser muy elevada para poder satisfacer ambas tensiones. Sabemos que esto no es posible, que un pin normal de un circuito integrado puede suministrar como mucho algunos centenares de miliamperios —y estos valores tan altos solo se alcanzan en circuitos especializados de potencia—. Por esta razón, la especificación de los niveles de tensión en las salidas viene acompañada de una segunda especificación, la de la intensidad máxima que se puede suministrar —en el nivel alto— o aceptar —en el nivel bajo— para que los citados valores de tensión se cumplan. Estas intensidades, `IOHMAX` e `IOLMAX` o simplemente `IOMAX` cuando es la misma en ambas direcciones, suelen ser del orden de pocas decenas de miliamperios —normalmente algo más de 20mA, lo requerido para encender con brillo suficiente un LED—. Así pues la especificación de los valores de tensión de las salidas se garantiza siempre y cuando la corriente que circule por el pin no supere el valor máximo correspondiente.

La naturaleza y el comportamiento de las entradas son radicalmente distintos, aunque se defina para ellas un conjunto similar de parámetros. Mediante un puerto de entrada queremos leer un valor lógico que se relacione con la tensión presente en el pin, fijada por algún sistema eléctrico exterior. Así pues, la misión de la circuitería del pin configurado como entrada es detectar niveles de tensión del exterior, con la menor

influencia en ellos que sea posible. De este modo, una entrada aparece para un circuito externo como una resistencia muy elevada, lo que se llama una *alta impedancia*. Dado que es un circuito activo y no una resistencia, el valor que se especifica es la intensidad máxima que circula entre el exterior y el circuito integrado, que suele ser despreciable en la mayor parte de los casos —del orden de pocos microamperios o menor—. Los valores especificados son `IIHMAX` e `IILMAX` o simplemente `IIMAX` si son iguales. En este caso la `I` significa entrada (*input*). Según esto, el circuito externo puede ser diseñado sabiendo la máxima corriente que va a disiparse hacia el pin, para generar las tensiones adecuadas para ser entendidas como `0` o `1` al leer el pin de entrada. Para ello se especifican `VIHMIN` y `VILMAX` como la mínima tensión de entrada que se lee como un `1` lógico y la máxima que se lee como un `0`, respectivamente. En ambos casos, por diseño del microcontrolador, se sabe que la corriente de entrada está limitada, independientemente del circuito externo.

¿Qué ocurre con una tensión en el pin comprendida entre `VIHMIN` y `VILMAX`? La lectura de un puerto de entrada siempre devuelve un valor lógico, por lo tanto cuando la tensión en el pin se encuentra fuera de los límites especificados, se lee también un valor lógico `1` o `0` que no se puede predecir según el diseño del circuito —una misma tensión podría ser leída como nivel alto en un pin y bajo en otro—. Visto de otra forma, un circuito —y un sistema en general— se debe diseñar para que fije una tensión superior a `VIHMIN` cuando queremos señalar un nivel alto, e inferior a `VILMAX` cuando queremos leer un nivel bajo. Otra precaución a tener en cuenta con las entradas es la de los valores máximos. En este caso el peligro no es que se lea un valor lógico distinto del esperado o impredecible, sino que se dañe el chip. Efectivamente, una tensión superior a la de alimentación o inferior a la de masa puede dañar definitivamente el pin de entrada e incluso todo el circuito integrado.

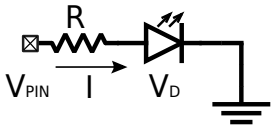

Figura 8.2: Conexión de un LED a un pin de E/S de un microcontrolador

Hagamos un pequeño estudio de los circuitos eléctricos relacionados con los dispositivos de nuestro ejemplo, el LED y el pulsador. Comencemos como viene siendo habitual por el circuito de salida. En la Figura 8.2 se muestra esquemáticamente la conexión de un LED a un pin de E/S de un microcontrolador. Un LED, por ser un diodo, tiene una tensión de conducción más o menos fija —en realidad en un LED depende más de la corriente que en un diodo de uso general— que en uno de color

rojo está entorno a los 1,2V. Por otra parte, a partir de 10mA el brillo del diodo es adecuado, pudiendo conducir sin deteriorarse hasta 30mA o más. Supongamos en nuestro microcontrolador los valores indicados más arriba para `VOHMIN` y `VOLMAX`, y una corriente de salida superior a los 20mA. Si queremos garantizar 10mA al escribir un `1` lógico en el pin, nos bastará con polarizar el LED con una resistencia que limite la corriente a este valor en el peor caso, es decir cuando la tensión de salida sea `VOHMIN`, es decir 4,2V. Mediante la ley de Ohm tenemos:

$$I = \frac{V}{R} \rightarrow 10mA = \frac{4{,}2V - 1{,}2V}{R} = \frac{3V}{R} \rightarrow R = 300\Omega$$

Una vez fijada esta resistencia, podemos calcular el brillo máximo del led, que se daría cuando la tensión de salida es de 5V, y entonces la corriente de 12,7mA aproximadamente.

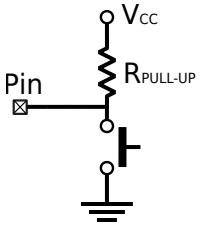

Figura 8.3: Conexión de un pulsador a un pin de E/S de un microcontrolador

Veamos ahora cómo conectar un pulsador a una entrada del circuito. En la Figura 8.3 se muestra el circuito esquemático de un pulsador conectado a un pin de E/S de un microcontrolador. Un pulsador no es más que una placa de metal que se apoya o se separa de dos conectores, permitiendo o no el contacto eléctrico entre ellos. Es, pues, un dispositivo electromecánico que no genera de por sí ninguna magnitud eléctrica. Para ello, hay que conectarlo en un circuito y, en nuestro caso, en uno que genere las tensiones adecuadas. Para seguir estrictamente los ejemplos, podemos pensar que el pulsador hace contacto entre los 5V de la alimentación y el pin. De este modo, al pulsarlo, conectaremos el pin a la alimentación y leeremos en él un `1`, tal y como se ha considerado en el código de ejemplo. Sin embargo, si no está pulsado, el pin no está conectado a nada por lo que el valor presente en él sería, en general, indefinido. Por ello el montaje correcto requiere que el pin se conecte a otro nivel de tensión, masa —0V— en este caso, a través de una resistencia para limitar la corriente al pulsar. Como la corriente de entrada en el pin es despreciable, el valor de la resistencia no es crítico, siendo lo habitual usar decenas o cientos de KΩ. Según este circuito y siguiendo

el ejemplo, al pulsar leeríamos un nivel alto y en caso de no pulsar, un `0` lógico.

La configuración más habitual es, sin embargo la contraria: conectar el pulsador a masa con uno de sus contactos y al pin y a la alimentación, a través de una resistencia, con el otro. De esta forma los niveles lógicos se invierten y se lee un `1` lógico en caso de no pulsar y un nivel bajo al hacerlo. Esto tiene implicaciones en el diseño de los microcontroladores y en la gestión de la GPIO, que nos ocupa. Es tan habitual el uso de resistencias conectadas a la alimentación —llamadas *resistencias de pull-up* o simplemente *pull-ups*— que muchos circuitos las llevan integradas en la circuitería del pin y no es necesario añadirlas externamente. Estas resistencias pueden activarse o no en las entradas, por lo que suele existir alguna forma de hacerlo, un nuevo registro de control del GPIO en la mayor parte de los casos.

8.1.2. Interrupciones asociadas a la GPIO

Como sabemos, las interrupciones son una forma de sincronizar el procesador con los dispositivos de entrada/salida para que éstos puedan avisar de forma asíncrona al procesador de que requieren su atención, sin necesidad de que aquél se preocupe periódicamente de atenderlos —lo que sería encuesta o prueba de estado. Los sistemas avanzados de GPIO incorporan la posibilidad de avisar al procesador de ciertos cambios mediante interrupciones, para poder realizar su gestión de forma más eficaz. Existen dos tipos de interrupciones que se pueden asociar a la GPIO, por supuesto siempre utilizada como entrada, como veremos a continuación.

En primer lugar se pueden utilizar los pines como líneas de interrupción, bien para señalar un cambio relativo al circuito conectado al pin, como oprimir un pulsador, bien para conectar una señal de un circuito externo y que así el circuito sea capaz de generar interrupciones. En este último caso el pin de la GPIO haría el papel de una línea de interrupción externa de un procesador. En ambos casos suele poder configurarse si la interrupción se señala por nivel o por flanco y su polaridad. En segundo lugar, y asociado a las características de bajo consumo de los microcontroladores, se tiene la interrupción por cambio de valor. Esta interrupción puede estar asociada a un pin o un conjunto de ellos, y se activa cada vez que alguno de los pines de entrada del grupo cambia su valor, desde la última vez que se leyó. Esta interrupción, además, suele usarse para sacar al procesador de un modo de bajo consumo y activarlo otra vez para reaccionar frente al cambio indicado.

El uso de interrupciones asociadas a la GPIO requiere añadir nuevos registros de control y estado, para configurar las interrupciones y sus

características —control— y para almacenar los indicadores —*flags*— que informen sobre las circunstancias de la interrupción.

8.1.3. Aspectos avanzados de la GPIO

Además de las interrupciones y la relación con los modos de bajo consumo, los microcontroladores avanzados añaden características y, por lo tanto, complejidad, a sus bloques de GPIO. Aunque estas características dependen bastante de la familia de microcontroladores, se pueden encontrar algunas tendencias generales que se comentan a continuación.

En primer lugar tenemos las modificaciones eléctricas de los bloques asociados a los pines. Estas modificaciones afectan solo a subconjuntos de estos pines y en algunos casos no son configurables, por lo que se deben tener en cuenta fundamentalmente en el diseño electrónico del sistema. Por una parte tenemos pines de entrada que soportan varios umbrales lógicos —lo más normal es 5V y 3,3V para el nivel alto—. También es frecuente encontrar entradas con disparador de Schmitt para generar flancos más rápidos en las señales eléctricas en el interior del circuito, por ejemplo en entradas que generen interrupciones, lo que produce que los valores `VIHMIN` y `VILMAX` en estos pines estén más próximos, reduciendo el rango de tensiones indeterminadas —a nivel lógico— entre ellos. Tenemos también salidas que pueden configurarse como colector abierto —*open drain*, en nomenclatura CMOS— lo que permite utilizarlas en sistemas AND cableados, muy utilizados en buses.

Otra tendencia actual, de la que participa el ATSAM3X8E, es utilizar un muestreo periódico de los pines de entrada, lo que requiere almacenar su valor en un registro, en lugar de utilizar el valor presente en el pin en el momento de la lectura. De esta manera es posible añadir filtros que permitan tratar ruido eléctrico en las entradas o eliminar los rebotes típicos en los pulsadores e interruptores. En este caso, se incorporan a la GPIO registros para activar o configurar estos métodos de filtrado. Esta forma de tratar las entradas requiere de un reloj para muestrearlas y almacenar su valor en el registro, lo que a su vez requiere poder parar este reloj para reducir el consumo eléctrico.

La última característica asociada a la GPIO que vamos a tratar surge de la necesidad de versatilidad de los microcontroladores. Los dispositivos actuales, además de gran número de pines en su GPIO, incorporan muchos otros dispositivos —conversores ADC y DAC, buses e interfaces estándar, etcétera— que también necesitan de pines específicos para relacionarse con el exterior. Para dejar libertad al diseñador de seleccionar la configuración del sistema adecuada para su aplicación, muchos pines pueden usarse como parte de la GPIO o con alguna de estas funciones específicas. Esto hace que exista un complejo subsistema de encaminado de señales entre los dispositivos internos y los pines, que afecta directa-

mente a la GPIO y cuyos registros de configuración suele considerarse como parte de aquélla.

8.1.4. GPIO en el Atmel ATSAM3X8E

El microcontrolador ATSAM3X8E dispone de bloques de GPIO muy versátiles y potentes. Dichos bloques, llamados *Parallel Input/Output Controller* (*PIO* en inglés) se describen en detalle a partir de la página 641 del manual. Vamos a resumir en este apartado los aspectos más importantes, centrándonos en la versión ATSAM3X8E del microcontrolador, presente en la tarjeta *Arduino Due*. En la Figura 8.4 (obtenida del manual [Atm12]) se muestra la estructura interna de un pin de E/S del microcontrolador ATSAM3X8E.

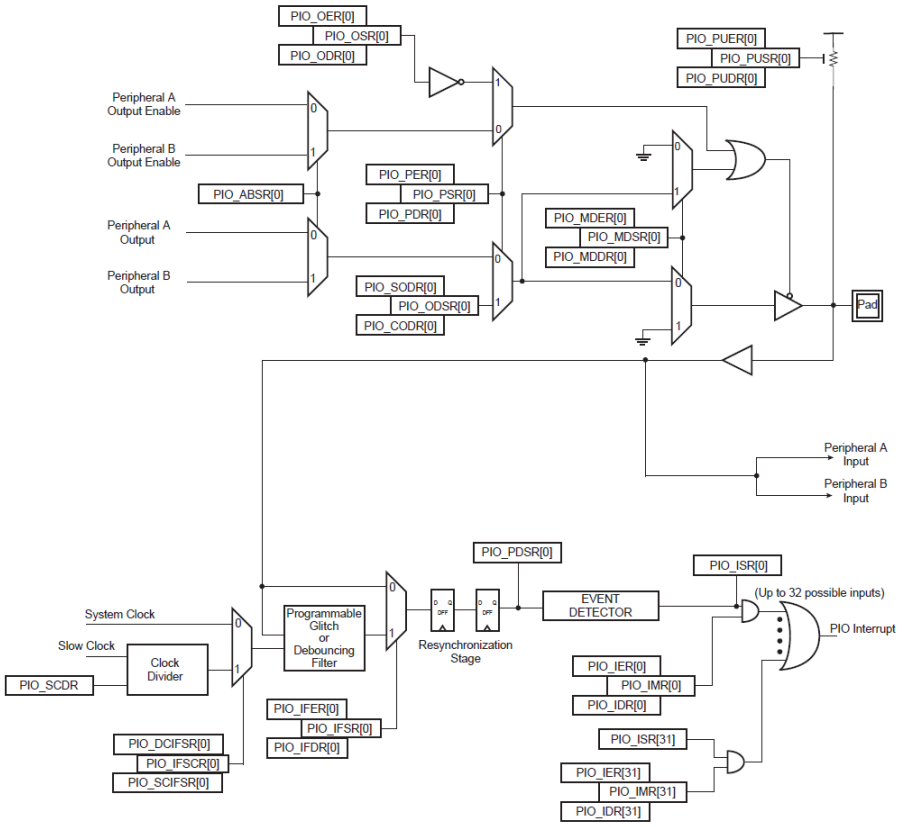

Figura 8.4: Estructura interna de un pin de E/S del microcontrolador ATSAM3X8E

Con un encapsulado LQFP de 144 pines, el microcontrolador dispone de 4 controladores PIO capaces de gestionar hasta 32 pines cada uno

de ellos, para un total de 103 líneas de GPIO. Cada una de estas líneas de entrada/salida es capaz de generar interrupciones por cambio de valor o como línea dedicada de interrupción; de configurarse para filtrar o eliminar rebotes de la entrada; de actuar con o sin *pull-up* o en colector abierto. Además, los pines de salida pueden ponerse a uno o a cero individualmente, mediante registros dedicados de *set* o *clear*, o conjuntamente a cualquier valor escribiendo en un tercer registro. Todas estas características hacen que cada bloque PIO tenga una gran complejidad, ocupando un espacio de 324 registros de 32 bits —1.296 direcciones— en el mapa de memoria. Veamos a continuación los registros más importantes y su uso.

Configuración como GPIO o E/S específica de otro periférico

La mayor parte de los pines del encapsulado pueden utilizarse como parte de la GPIO o con una función específica, seleccionable de entre dos dispositivos de E/S del microcontrolador. Así pues, para destinar un pin a la GPIO deberemos habilitarlo para tal fin. Si posteriormente queremos utilizar alguna de las funciones específicas, podremos volver a deshabilitarlo como E/S genérica. Como en muchos otros casos, y por motivos de seguridad y velocidad —ahorra tener que leer los registros para preservar sus valores— el controlador PIO dedica tres registros a este fin: uno para habilitar los pines, otro para deshabilitarlos y un tercero para leer el estado de los pines en un momento dado. De esta manera, dado que al habilitar y deshabilitar se escriben `1` en los bits afectados, sin modificar el resto, no es necesario preservar ningún estado al escribir. Veamos los registros asociados a esta funcionalidad:

- *PIO Enable Register* (`PIO_PER`): escribiendo un `1` en cualquier bit de este registro habilita el pin correspondiente para uso como GPIO, inhibiendo su uso asociado a otro dispositivo de E/S.
- *PIO Disable Register* (`PIO_PDR`): al revés que el anterior, escribiendo un `1` se deshabilita el uso del pin como parte de la GPIO y se asocia a uno de los dispositivos periféricos asociados.
- *PIO Status Register* (`PIO_PSR`): este registro de solo lectura permite conocer en cualquier momento el estado de los pines asociados al PIO. Un `1` en el bit correspondiente indica que son parte de la GPIO mientras que un `0` significa que están dedicados a la función del dispositivo asociado.
- *PIO Peripheral AB Select Register* (`PIO_ABSR`): permite seleccionar a cuál de los dos posibles dispositivos periféricos está asociado el pin en caso de no estarlo a la GPIO. Un `0` selecciona el `A` y un `1` el

`B`. Los dispositivos identificados como `A` y `B` dependen de cada pin en particular.

Configuración y uso genéricos como GPIO

Una vez los pines se han asignado a la GPIO, es necesario realizar su configuración específica. Esto incluye indicar si su dirección es entrada o salida, y activar o no las resistencias de *pull-up* o la configuración en colector abierto. Veamos los registros del PIO que se utilizan:

- *Output Enable Register* (`PIO_OER`): escribiendo un `1` en cualquier bit de este registro configura el pin correspondiente como salida.
- *Output Disable Register* (`PIO_ODR`): escribiendo un `1` en cualquier bit de este registro deshabilita el pin correspondiente como salida, quedando entonces como pin de entrada.
- *Output Status Register* (`PIO_OSR`): este registro de solo lectura permite conocer en cualquier momento la dirección de los pines. Un `1` en el bit correspondiente indica que el pin está configurado como salida mientras que un `0` significa que el pin es una entrada.

Se dispone además del trío de registros *Pull-up Enable*, *Pull-up Disable* y *Pull-up Status* que permiten respectivamente activar, desactivar y leer el estado de configuración de las resistencias de *pull-up*. En este último caso, un `1` indica deshabilitada y un `0` habilitada. Por último, los tres registros *Multi-driver Enable*, *Multi-driver Disable* y *Multi-driver Status* permiten configurar eléctricamente el pin en colector abierto —o deshacer esta configuración— y leer el estado de los pines a este respecto —de la forma habitual, no invertida como en el caso anterior—.

Una vez configurada la GPIO, nuestro programa debe únicamente trabajar con los pines, escribiendo y leyendo valores según la tarea a realizar. De nuevo el bloque GPIO ofrece una gran versatilidad, a costa de cierta complejidad, como veremos a continuación. Comencemos con la lectura de los valores de entrada, algo sencillo dado que basta con leer el registro *Pin Data Status Register* (`PIO_PDSR`) para obtener los valores lógicos presentes en ellos. Es conveniente indicar que para poder leer los pines de entrada —igual que para muchas otras funciones del PIO— el reloj que lo sincroniza debe estar activado. La gestión de las salidas presenta algo más de complejidad dado que existen dos modos de actuar sobre cada una de ellas. Por una parte, tenemos la forma común en este microcontrolador, disponiendo de un registro para escribir unos y otro para escribir ceros. Este modo, que en muchas ocasiones simplifica la escritura en los pines, presenta el problema de que no se pueden escribir de forma simultánea unos y ceros. Por ello existe un segundo modo, en

que se escribe en una sola escritura el valor, con los unos y ceros deseado, sobre el registro de salida. Para que este modo no afecte a todos los pines gestionados por el PIO —hasta 32— existe un registro adicional para seleccionar aquéllos a los que va a afectar esta escritura. Para poder implementar todos estos modos, el conjunto de registros relacionados con la escritura de valores en las salidas, es el siguiente:

- *Set Output Data Register* (`PIO_SODR`): escribiendo un `1` en cualquier bit de este registro se escribe un uno en la salida correspondiente.
- *Clear Output Data Register* (`PIO_CODR`): escribiendo un `1` en cualquier bit de este registro se escribe un cero en la salida correspondiente.
- *Output Data Status Register* (`PIO_ODSR`): al leer este registro obtenemos en cualquier momento el valor lógico que hay en las salidas cuando se lee. Al escribir en él, modificamos con el valor escrito los valores de aquellas salidas habilitadas para escritura directa en `PIO_OWSR`.
- *Output Write Enable Register* (`PIO_OWER`): escribiendo un `1` en cualquier bit de este registro habilita tal pin para escritura directa.
- *Output Write Disable Register* (`PIO_OWDR`): escribiendo un `1` en cualquier bit de este registro deshabilita tal pin para escritura directa.
- *Output Write Status Register* (`PIO_OWSR`): este registro de solo lectura permite conocer en cualquier momento las salidas habilitadas para escritura directa.

Gestión de interrupciones asociadas a la GPIO

El controlador PIO es capaz de generar diversas interrupciones asociadas a los pines de la GPIO a él asociados. Para que dichas interrupciones se puedan propagar al sistema, la interrupción generada por el PIO debe estar convenientemente programada en el controlador de interrupciones del sistema, llamado *NVIC*. Además el reloj de sincronización del PIO debe estar activado. Dándose estas circunstancias, el PIO gestiona diferentes fuentes de interrupción que disponen de un registro de señalización y otro de máscara. La activación de cualquier causa de interrupción se reflejará siempre en el registro de señalización y se generará o no la interrupción en función del valor de la máscara correspondiente, que estará a `1` si la interrupción está habilitada. La causa básica de interrupción es el cambio de valor en un pin. Sin embargo, esta causa

puede modificarse para que la interrupción se genere cuando se detecta un flanco de subida o de bajada, o un nivel determinado en el pin. Veamos el conjunto de registros que permiten esta funcionalidad, y su uso:

- *Interrupt Enable Register* (`PIO_IER`): escribiendo un `1` en cualquier bit de este registro se habilita la interrupción correspondiente.
- *Interrupt Disable Register* (`PIO_IDR`): escribiendo un `1` en cualquier bit de este registro se deshabilita la interrupción correspondiente.
- *Interrupt Mask Register* (`PIO_IMR`): al leer este registro obtenemos el valor de la máscara de interrupciones, que se corresponde con las interrupciones habilitadas.
- *Interrupt Status Register* (`PIO_ISR`): en este registro se señalan con un `1` las causas de interrupción pendientes, es decir aquéllas que se han dado, sea cual sea su tipo, desde la última vez que se leyó este registro. Se pone a `0` automáticamente al ser leído.
- *Additional Interrupt Modes Enable Register* (`PIO_AIMER`): escribiendo un `1` en cualquier bit de este registro se selecciona la causa de interrupción adicional, por flanco o por nivel.
- *Additional Interrupt Modes Disable Register* (`PIO_AIMDR`): escribiendo un `1` en cualquier bit de este registro se selecciona la causa básica de interrupción, cambio de valor en el pin.
- *Additional Interrupt Modes Mask Register* (`PIO_AIMMR`): este registro de solo lectura permite saber si la causa de interrupción configurada es cambio de valor, lo que se indica con un `0`, o modo adicional, con un `1`.
- *Edge Select Register* (`PIO_ESR`): escribiendo un `1` en cualquier bit de este registro se selecciona el flanco como la causa de interrupción adicional.
- *Level Select Register* (`PIO_LSR`): escribiendo un `1` en cualquier bit de este registro se selecciona el nivel como la causa de interrupción adicional.
- *Edge/Level Status Register* (`PIO_ELSR`): este registro de solo lectura permite saber si la causa de interrupción adicional configurada es flanco, lo que se indica con un `0`, o nivel, con un `1`.
- *Falling Edge/Low Level Select Register* (`PIO_FELLSR`): escribiendo un `1` en cualquier bit de este registro se selecciona el flanco de

bajada o el nivel bajo, según `PIO_ELSR`, como la polaridad de interrupción.

- *Rising Edge/High Level Select Register* (`PIO_REHLSR`): escribiendo un `1` en cualquier bit de este registro se selecciona el flanco de subida o el nivel alto, según `PIO_ELSR`, como la polaridad de interrupción.
- *Fall/Rise Low/High Status Register* (`PIO_FRLHSR`): este registro de solo lectura permite saber la polaridad de la interrupción.

Registros adicionales y funciones avanzadas del PIO

Una de las funciones avanzadas del controlador PIO es la eliminación de ruidos en las entradas. Aunque no se va a ver en detalle —recordemos que siempre se puede acceder a la especificación completa en el manual— conviene comentar que se tiene la posibilidad de activar filtros para las señales de entrada, configurables como filtros de ruido —traducción aproximada de *glitches*— o para la eliminación de rebotes si a la entrada se conecta un pulsador —*debouncing*—. Como estos filtros están basados en el sobremuestreo de la señal presente en el pin —recordemos que las entradas no leen directamente el pin sino que muestrean su valor y lo almacenan en un registro— es posible además variar el reloj asociado a este sobremuestreo.

La última posibilidad que ofrece el controlador PIO es la de bloquear o proteger contra escritura parte de los registros de configuración que se han descrito, para prevenir que errores en la ejecución del programa produzcan cambios indeseados en la configuración.

8.1.5. Controladores PIO en el ATSAM3X8E

Conocida la información que aparece en el texto anterior, para hacer programas que interactúen con la GPIO del microcontrolador solo falta conocer las direcciones del mapa de memoria en que se sitúan los registros de los controladores PIO del ATSAM3X8E y la dirección —más bien desplazamiento u *offset*— de cada registro dentro del bloque. El Cuadro 8.1 muestra las direcciones base de los controladores PIO de que dispone el sistema.

Así mismo, en los Cuadross 8.2, 8.3 y 8.4 se muestran los desplazamientos (*offsets*) de los registros de E/S de cada uno de los controladores PIO, de forma que para obtener la dirección de memoria efectiva de uno de los registros hay que sumar a la dirección base del controlador PIO al que pertenece, el desplazamiento indicado para el propio registro:

PIO	Pines de E/S disponibles	Dirección base
PIOA	30	0x400E 0E00
PIOB	32	0x400E 1000
PIOC	31	0x400E 1200
PIOD	10	0x400E 1400

Cuadro 8.1: Direcciones base de los controladores PIO del ATSAM3X8E

Registro	Alias	Desplazamiento
PIO Enable Register	PIO_PER	0x0000
PIO Disable Register	PIO_PDR	0x0004
PIO Status Register	PIO_PSR	0x0008
Output Enable Register	PIO_OER	0x0010
Output Disable Register	PIO_ODR	0x0014
Output Status Register	PIO_OSR	0x0018
Glitch Input Filter Enable Register	PIO_IFER	0x0020
Glitch Input Filter Disable Register	PIO_IFDR	0x0024
Glitch Input Filter Status Register	PIO_PIFSR	0x0028
Set Output Data Register	PIO_SODR	0x0030
Clear Output Data Register	PIO_CODR	0x0034
Output Data Status Register	PIO_ODSR	0x0038
Pin Data Status Register	PIO_PDSR	0x003C

Cuadro 8.2: Registros de E/S de cada controlador PIO y sus desplazamientos. Parte I

8.1.6. La tarjeta de entrada/salida

Para poder practicar con la GPIO se ha diseñado una pequeña tarjeta que se inserta en los conectores de expansión de la Arduino Due. La tarjeta dispone de un LED RGB —rojo *Red* verde *Green* azul *Blue*— conectado a tres pines que se usarán como salidas, y un pulsador conectado a un pin de entrada. Los tres diodos del LED están configurados

Registro	Alias	Desplazamiento
Interrupt Enable Register	PIO_IER	0x0040
Interrupt Disable Register	PIO_IDR	0x0044
Interrupt Mask Register	PIO_IMR	0x0048
Interrupt Status Register	PIO_ISR	0x004C
Multi-driver Enable Register	PIO_MDER	0x0050
Multi-driver Disable Register	PIO_MDDR	0x0054
Multi-driver Status Register	PIO_MDSR	0x0058
Pull-up Disable Register	PIO_PUDR	0x0060
Pull-up Enable Register	PIO_PUER	0x0064
Pad Pull-up Status Register	PIO_PUSR	0x0068
Peripheral AB Select Register	PIO_ABSR	0x0070
System Clock Glitch Input Filter Select Register	PIO_SCIFSR	0x0080
Debouncing Input Filter Select Register	PIO_DIFSR	0x0084
Glitch or Debouncing Input Filter Clock Selection Status Register	PIO_IFDGSR	0x0088
Slow Clock Divider Debouncing Register	PIO_SCDR	0x008C

Cuadro 8.3: Registros de E/S de cada controlador PIO y sus desplazamientos. Parte II

en ánodo común, por lo que se encienden al escribir un `0` en la salida correspondiente. Cada canal del LED lleva su correspondiente resistencia para limitar la corriente; el terminal común se debe conectar, a través del cable soldado a la tarjeta, a la salida de `3.3V` de la Arduino Due. El pulsador se conecta a un pin de entrada a través de una resistencia de protección, y a masa. Activando la resistencia de *pull-up* asociada al pin, se leerá un `1` lógico si el interruptor no está pulsado, y un `0` al pulsarlo.

El Cuadro 8.5 y las Figuras 9.3 y 9.4 completan la información técnica acerca de la tarjeta.

Registro	Alias	Desplazamiento
Output Write Enable	PIO_OWER	0x00A0
Output Write Disable	PIO_OWDR	0x00A4
Output Write Status Register	PIO_OWSR	0x00A8
Additional Interrupt Modes Enable Register	PIO_AIMER	0x00B0
Additional Interrupt Modes Disable Register	PIO_AIMDR	0x00B4
Additional Interrupt Modes Mask Register	PIO_AIMMR	0x00B8
Edge Select Register	PIO_ESR	0x00C0
Level Select Register	PIO_LSR	0x00C4
Edge/Level Status Register	PIO_ELSR	0x00C8
Falling Edge/Low Level Select Register	PIO_FELLSR	0x00D0
Rising Edge/ High Level Select Register	PIO_REHLSR	0x00D4
Fall/Rise - Low/High Status Register	PIO_FRLHSR	0x00D8
Lock Status	PIO_LOCKSR	0x00E0
Write Protect Mode Register	PIO_WPMR	0x00E4
Write Protect Status Register	PIO_WPSR	0x00E8

Cuadro 8.4: Registros de E/S de cada controlador PIO y sus desplazamientos. Parte III

8.2. Gestión del tiempo

La medida del tiempo es fundamental en la mayoría de las actividades humanas y por ello, lógicamente, se incluye entre las características principales de los ordenadores, en los que se implementa habitualmente mediante dispositivos de entrada/salida. Anotar correctamente la fecha y hora de modificación de un archivo, arrancar automáticamente tareas con cierta periodicidad, determinar si una tecla se ha pulsado duran-

PIN	Función	Puerto	Bit
6	LED azul	PIOC	24
7	LED verde	PIOC	23
8	LED rojo	PIOC	22
13	Pulsador	PIOB	27

Cuadro 8.5: Pines y bits de los dispositivos de la tarjeta de E/S en la tarjeta Arduino Due

te más de medio segundo, son actividades comunes en los ordenadores que requieren de una correcta medida y gestión del tiempo. En estos ejemplos se puede ver además las distintas escalas y formas de tratar el tiempo. Desde expresar una fecha y hora de la forma habitual para las personas —donde además se deben tener en cuenta las diferencias horarias entre distintos países— hasta medir lapsos de varias horas o pocos milisegundos, los ordenadores son capaces de realizar una determinación adecuada de tiempos absolutos o retardos entre sucesos. Esto se consigue mediante un completo y elaborado sistema de tratamiento del tiempo, que tiene gran importancia dentro del conjunto de dispositivos y procedimientos relacionados con la entrada/salida de los ordenadores.

8.2.1. El tiempo en la E/S de los sistemas

Un sistema de tiempo real se define como aquél capaz de generar resultados correctos y a tiempo. Los ordenadores de propósito general pueden ejecutar aplicaciones de tiempo real, como reproducir una película o ejecutar un videojuego, de la misma forma en que mantienen la fecha y la hora del sistema, disparan alarmas periódicas, etcétera. Para ser capaces de ello, además de contar con la velocidad de proceso suficiente, disponen de un conjunto de dispositivos asociados a la entrada/salida que facilitan tal gestión del tiempo liberando al procesador de buena parte de ella. En los microcontroladores, dispositivos especialmente diseñados para interactuar con el entorno y adaptarse temporalmente a él, normalmente mediante proceso de tiempo real, el conjunto de dispositivos y mecanismos relacionados con el tiempo es mucho más variado e importante.

En todos los ordenadores se encuentra, al menos, un dispositivo tipo contador que se incrementa de forma periódica y permite medir intervalos de tiempo de corta duración —milisegundos o menos—. A partir de esta base de tiempos se puede organizar toda la gestión temporal del sistema, sin más que incluir los programas necesarios. Sin embargo se suele disponer de otro dispositivo que gestiona el tiempo en formato

humano —formato de fecha y hora— que permite liberar de esta tarea al software del sistema y además, utilizando alimentación adicional, mantener esta información aún con el sistema apagado. Por último, para medir eventos externos muy cortos, para generar señales eléctricas con temporización precisa y elevadas frecuencias, se suelen añadir otros dispositivos que permiten generar pulsos periódicos o aislados o medir por hardware cambios eléctricos en los pines de entrada/salida.

Todos estos dispositivos asociados a la medida de tiempo pueden avisar al sistema de eventos temporales tales como desbordamiento en los contadores o coincidencias de valores de tiempo —alarmas— mediante los correspondientes bits de estado y generación de interrupciones. Este variado conjunto de dispositivos se puede clasificar en ciertos grupos que se encuentran, de forma más o menos similar, en la mayoría de los sistemas. En los siguientes apartados se describen estos grupos y se indican sus características más comunes.

El temporizador del sistema

El temporizador —*timer*— del sistema es el dispositivo más común y sencillo. Constituye la base de medida y gestión de tiempos del sistema. Se trata de un registro contador que se incrementa de forma periódica a partir de cierta señal de reloj generada por el hardware del sistema. Para que su resolución y tiempo máximo puedan configurarse según las necesidades, es habitual encontrar un divisor de frecuencia o *prescaler* que permite disminuir con un margen bastante amplio la frecuencia de incremento del contador. De esta manera, si la frecuencia final de incremento es f, se tiene que el tiempo mínimo que se puede medir es el periodo, $T = 1/f$, y el tiempo que transcurre hasta que se desborde el contador $2^n \cdot T$, siendo n el número de bits del registro temporizador. Para una frecuencia de 10KHz y un contador de 32 bits, el tiempo mínimo sería 100μs y transcurrirían unos 429.496s —casi cinco días— hasta que se desbordara el temporizador. El temporizador se utiliza, en su forma más simple, para medir tiempos entre dos eventos —aunque uno de ellos pueda ser el inicio del programa—. Para ello, se guarda el valor del contador al producirse el primer evento y se resta del valor que tiene al producirse el segundo. Esta diferencia multiplicada por el periodo nos da el tiempo transcurrido entre ambos eventos.

El ejemplo comentado daría un valor incorrecto si entre las dos lecturas se ha producido más de un desbordamientos del reloj. Por ello, el temporizador activa una señal de estado que generalmente puede causar una interrupción cada vez que se desborda, volviendo a `0`. El sistema puede tratar esta información, sobre todo la interrupción generada, de distintas formas. Por una parte se puede extender el contador con variables en memoria para tener mayor rango en la cuenta de tiempos. Es

habitual también utilizarla como interrupción periódica para gestión del sistema —por ejemplo, medir los tiempos de ejecución de los procesos en sistemas multitarea—. En este último caso es habitual poder recargar el contador con un valor distinto de `0` para tener un control más fino de la periodicidad de la interrupción, por ello suele ser posible escribir sobre el registro que hace de contador.

Además de este funcionamiento genérico del contador, existen algunas características adicionales bastante extendidas en muchos sistemas. Por una parte, no es extraño que la recarga del temporizador después de un desbordamiento se realice de forma automática, utilizando un valor almacenado en otro registro del dispositivo. De esta forma, el software de gestión se libera de esta tarea. En sistemas cuyo temporizador ofrece una medida de tiempo de larga duración, a costa de una resolución poco fina, de centenares de ms, se suele generar una interrupción con cada incremento del contador. La capacidad de configuración de la frecuencia de tal interrupción es a costa del *prescaler*. Es conveniente comentar que, en arquitecturas de pocos bits que requieren contadores con más resolución, la lectura de la cuenta de tiempo requiere varios accesos —por ejemplo, un contador de 16 bits requeriría dos en una arquitectura de 8 bits—. En este caso pueden leerse valores erróneos si el temporizador se incrementa entre ambos accesos, de forma que la parte baja se desborde. Por ejemplo, si el contador almacena el valor `0x3AFF` al leer la parte baja y se incrementa a `0x3B00` antes de leer la alta, el valor leído será `0x3BFF`, mucho mayor que el real. En estos sistemas el registro suele constar de una copia de respaldo que se bloquea al leer una de las dos partes, con el valor de todo el temporizador en ese instante. De esta manera, aunque el temporizador real siga funcionando, las lecturas se harán de esta copia bloqueada, evitando los errores.

Más adelante se describen las particularidades del temporizador *RTT* (*Real-time Timer*) del ATSAM3X8E.

Otros dispositivos temporales

Si solo se dispone de un dispositivo temporizador se debe elegir entre tener una medida de tiempos de larga duración —hasta de varios años en muchos casos— para hacer una buena gestión del tiempo a lo largo de la vida del sistema o tener una buena resolución —pocos ms o menos— para medir tiempos con precisión. Por eso es común que los sistemas dispongan de varios temporizadores que, compartan o no la misma base de tiempos, pueden configurar sus periodos mediante *prescaleres* individuales. Estos sistemas, con varios temporizadores, añaden otras características que permiten una gestión mucho más completa del tiempo. Las características más comunes de estas extensiones del temporizador básico se analizan a continuación.

Algún registro temporizador puede utilizar como base de tiempos una entrada externa —un pin del microcontrolador, normalmente—. Esto permite por una parte tener una fuente de tiempo con las características que se deseen o utilizar el registro como contador de eventos, no de tiempos, dado que las señales en el pin no tienen por qué cambiar de forma periódica.

Se utilizan registros de comparación, con el mismo número de bits del temporizador, que desencadenan un evento cuando el valor del temporizador coincide con el de alguno de aquéllos. Estos eventos pueden ser internos, normalmente la generación de alguna interrupción, o externos, cambiando el nivel eléctrico de algún pin y pudiendo generar salidas dependientes del tiempo.

Se añaden registros de copia que guardan el valor del temporizador cuando se produce algún evento externo, además de poder generar una interrupción. Esto permite medir con precisión el tiempo en que ocurre algo en el exterior, con poca carga para el software del sistema.

Están muy extendidas como salidas analógicas aquéllas que permiten modulación de anchura de pulsos, *PWM —Pulse Width Modulation—*. Se dispone de una base de tiempos asociada a un temporizador que marca la frecuencia del canal PWM, y de un registro que indica el porcentaje de nivel alto o bajo de la señal de salida. De esa forma se genera una señal periódica que se mantiene a nivel alto durante un cierto tiempo y a nivel bajo el resto del ciclo. Como el ciclo de trabajo depende del valor almacenado en el registro, la cantidad de potencia —nivel alto— entregada de forma analógica en cada ciclo se relaciona directamente con su valor —magnitud digital—. Si la salida PWM ataca un dispositivo que se comporta como un filtro pasa-baja, lo que es muy frecuente en dispositivos reales —bombillas y LEDs, calefactores, motores, etcétera— se tiene una conversión digital-analógica muy efectiva, basada en el tiempo.

El reloj en tiempo real

En un computador, el reloj en tiempo real o *RTC* (*Real-time Clock*) es un circuito específico encargado de mantener la fecha y hora actuales incluso cuando el computador está desconectado de la alimentación eléctrica. Es por este motivo que suele estar asociado a una batería o a un condensador que le proporciona la energía necesaria para seguir funcionando cuando se interrumpe la alimentación.

Habitualmente este periférico emplea como frecuencia base una señal de 32.768 Hz, es decir, una señal cuadrada que completa 32.768 veces un ciclo OFF-ON cada segundo. Esta frecuencia es la empleada habitualmente por los relojes de cuarzo, dado que coincide con 2^{15} ciclos por segundo, con lo cual el bit de peso 15 del contador de ciclos cambia de valor exactamente una vez por segundo y puede usarse como señal de

activación del segundero en el caso de un reloj analógico o del contador de segundos en uno digital.

El módulo RTC se suele presentar como un dispositivo independiente conteniendo el circuito oscilador, el contador, la batería y una pequeña cantidad de memoria RAM que se usa para almacenar la configuración de la *BIOS* del computador. Este módulo se incorpora en la placa base del computador presentando, respecto de la opción de implementarlo por software, las siguientes ventajas:

- El procesador queda liberado de la tarea de contabilizar el tiempo. El RTC dispone de algunos registros de E/S mediante los cuales se pueden configurar y consultar la fecha y hora actuales,
- Suele presentar mayor precisión, dado que está diseñado específicamente.
- La presencia de la batería permite mantener el reloj en funcionamiento cuando el computador se apaga.

8.2.2. El temporizador del Atmel ATSAM3X8E y del sistema Arduino

La arquitectura ARM especifica un temporizador llamado *System Timer* como la base principal de tiempos del sistema, con una frecuencia de incremento similar a la del procesador lo que le permite medir intervalos de tiempo muy pequeños —del orden de microsegundos—. Para este temporizador, que no es otra cosa que un dispositivo de entrada/salida, aunque especial en el sistema, reserva sin embargo una excepción del sistema, llamada *SysTick*, con un número de interrupción fijo en el sistema, a diferencia del resto de dispositivos, cuyos números de interrupción no están fijados por la arquitectura.

El microcontrolador ATSAM3X8E implementa el *System Timer* especificado en la arquitectura. Se trata de un dispositivo de entrada/salida que se comporta como un temporizador convencional, que se decrementa con cada pulso de su reloj. Dispone de cuatro registros, que se describen a continuación:

- *Control and Status Register* (`CTRL`): de los 32 bits que contiene este registro solo 4 son útiles, tres de control y uno de estado. De los primeros, el bit 2 —`CLKSOURCE`— indica la frecuencia del temporizador, que puede ser la misma del sistema o un octavo de ésta; el bit 1 —`TICKINT`— es la habilitación de interrupción y el bit 0 —`ENABLE`— la habilitación del funcionamiento del temporizador. El bit 16 —`COUNTFLAG`— es el único de estado, e indica si el contador ha llegado a `0` desde la última vez que se leyó el registro.

- *Reload Value Register* (`LOAD`): cuando el temporizador llega a `0` recarga automáticamente el valor de 24 bits —los 8 más altos no se usan— presente en este registro, comenzando a decrementarse desde tal valor. De esta manera se puede ajustar con más precisión el tiempo transcurrido hasta que se llega a cero y con ello, si están habilitadas, el tiempo entre interrupciones.
- *Current Value Register* (`VAL`): este registro guarda el valor actual del contador decreciente, de 24 bits —los 8 más altos no se usan—.
- *Calibration Value Register* (`CALIB`): contiene valores relacionados con la calibración de la frecuencia de actualización.

En el Cuadro 8.6 aparecen las direcciones de los registros citados.

Registro	Alias	Dirección
Control and Status Register	`CTRL`	`0xE000E010`
Reload Value Register	`LOAD`	`0xE000E014`
Current Value Register	`VAL`	`0xE000E018`
Calibration Value Register	`CALIB`	`0xE000E01C`

Cuadro 8.6: Registros del temporizador del ATSAM3X8E y sus direcciones de E/S

El entorno Arduino añade a cada programa el código necesario para la configuración del sistema y las rutinas de soporte necesarias. Entre ellas se tiene la configuración del *System Timer* y la rutina de tratamiento de la excepción *SysTick*. Este código configura el reloj del sistema, de 84MHz, como frecuencia de actualización del temporizador, y escribe el valor `0x01481F`, 83.999 en decimal, en el registro de recarga. De esta manera se tiene un cambio en el contador cada 12 nanosegundos más o menos y una interrupción cada milisegundo, lo que sirve de base para las funciones «`delay()`» y «`millis()`» del entorno. Ambas utilizan el contador de milisegundos del sistema «`_dwTickCount`», que es una variable en memoria que se incrementa en la rutina de tratamiento de *SysTick*. Las funciones de mayor precisión «`delayMicroseconds()`» y «`micros()`» se implementan leyendo directamente el valor del registro `VAL`.

8.2.3. El reloj en tiempo real del Atmel ATSAM3X8E

Algunos microcontroladores incorporan un RTC entre sus periféricos integrados, lo cual les permite disponer de fecha y hora actualizadas. En este caso, sin embargo, no suele existir alimentación específica para el

módulo RTC, con lo cual, al desaparecer la alimentación externa, se pierde la información de fecha y hora actuales.

El microcontrolador ATSAM3X8E posee un RTC cuya estructura se muestra en la Figura 8.5. Como puede apreciarse, recibe una señal de reloj `SCLK` (*Slow Clock*) generada internamente por el microcontrolador que presenta la ya mencionada frecuencia de 32.768 Hz. Esta señal se hace pasar por un divisor por 32768 para obtener una señal de reloj de exactamente 1 Hz, que se encargará de activar las actualizaciones de los contenidos de los registros que mantienen la hora y la fecha actuales, en ese orden.

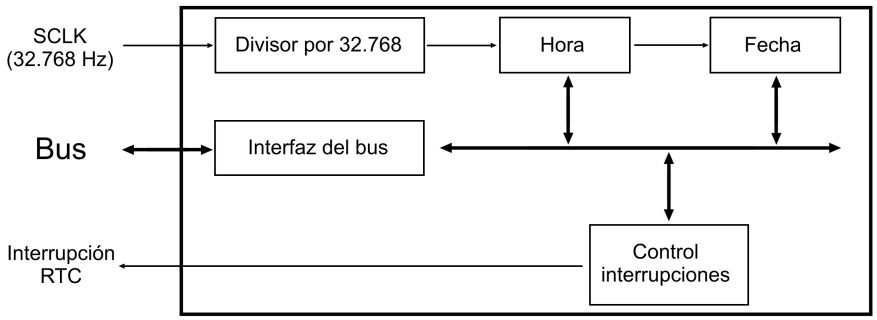

Figura 8.5: Estructura interna del RTC del ATSAM3X8E

Por otro lado, el RTC está conectado al bus interno del ATSAM3X8E para que se pueda acceder a los contenidos de sus registros. De esta forma es posible tanto leer la fecha y hora actuales, modificarlas y configurar las alarmas. Para ello, el RTC dispone de algunos registros de control encargados de gestionar las funciones de consulta, modificación y configuración del módulo.

Hora actual

La hora actual se almacena en un registro de 32 bits denominado `RTC_TIMR` (*RTC Time Register*), cuyo contenido se muestra en la Figura 8.6, donde la hora puede estar expresada en formato de 12 horas más indicador AM/PM —bit 22— o en formato de 24 horas. Todos los valores numéricos están codificados en *BCD* (*Binary Coded Decimal*, decimal codificado en binario), cuya equivalencia se muestra en el Cuadro 8.7.

Los segundos —`SEC`— se almacenan en los bits 0 al 6, conteniendo los bits del 0 al 3 el valor de las unidades. Dado que las decenas adoptan como máximo el valor 5, para este dígito solamente son necesarios tres

Decimal	BCD	Decimal	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Cuadro 8.7: Equivalencia entre decimal y BCD

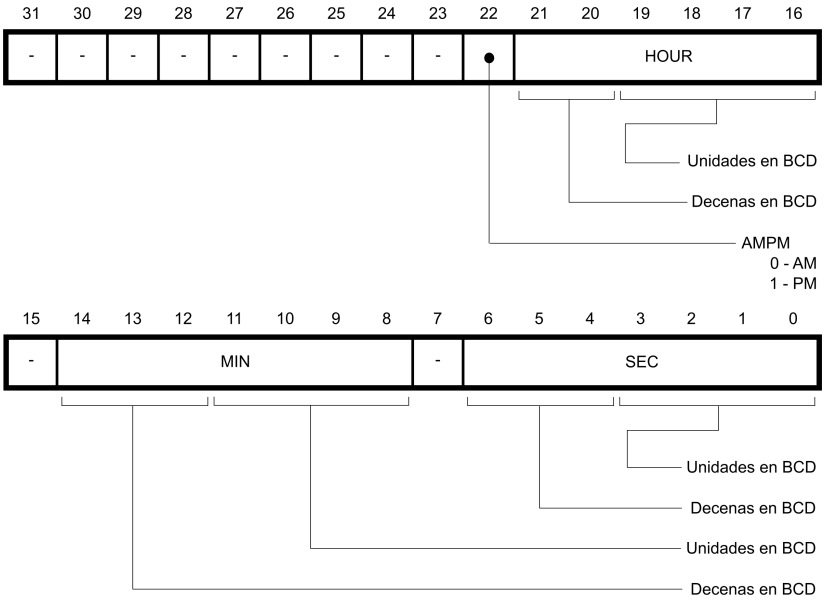

Figura 8.6: Contenido del registro RTC Time Register

bits —del 4 al 6—, por lo cual el bit 7 no se usa nunca y siempre debe valer cero.

Los minutos —`MIN`— se almacenan en los bits del 8 al 14. Las unidades se almacenan en los bits del 8 al 11 y con las decenas ocurre lo mismo que con los segundos: solamente hacen falta tres bits, del 12 al 14.

En cuanto a las horas —`HOUR`—, las decenas solamente pueden tomar los valores 0, 1 y 2, con lo cual es suficiente con dos bits —el 20 y el 21— y así el bit 22 queda para expresar la mañana y la tarde en el formato de 12 horas y el 23 no se usa.

El resto de bits —del 24 al 31— no se usan. El valor de la hora actual se lee y se escribe como un valor de 32 bits accediendo a este registro

con una sola operación de lectura o escritura.

Fecha actual

La fecha actual se almacena en el registro `RTC_CALR` (*RTC Calendar Register*) organizado como se muestra en la Figura 8.7:

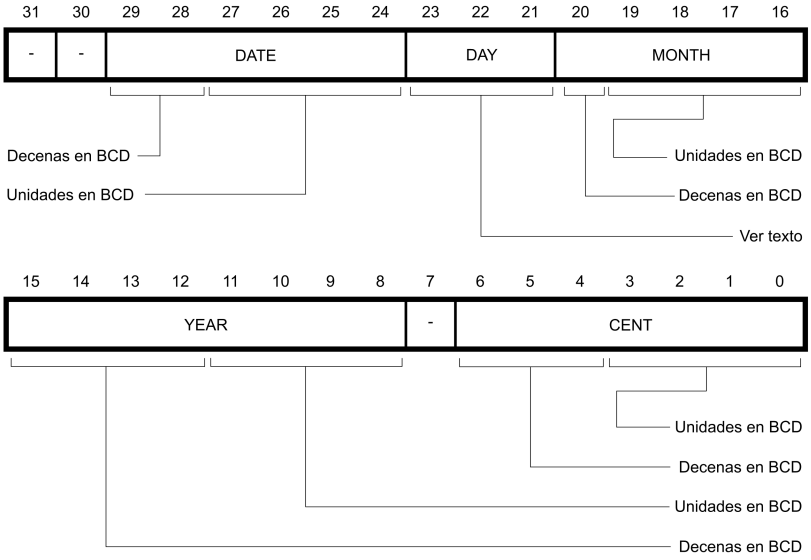

Figura 8.7: Contenido del registro RTC Calendar Register

Los bits del 0 al 6 contienen el valor del siglo —`CENTURY`—, pudiendo tomar solamente los valores 19 y 20 —refiriéndose, respectivamente, a los siglos 20 y 21—. Los bits del 0 al 3 almacenan las unidades de este valor (9 ó 0) y los bits del 4 al 6 las decenas (0 ó 2).

El año actual —`YEAR`— se almacena en los bits del 8 al 15, conteniendo los bits del 8 al 11 el valor BCD correspondiente a las unidades y los bits del 12 al 15 el valor BCD correspondiente a las decenas.

Esto confiere al ATSAM3X8E la capacidad de expresar la fecha actual en un rango de 200 años, desde el 1 de enero de 1900 hasta el 31 de diciembre de 2099.

El mes del año —`MONTH`— se almacena en los bits del 16 al 20, conteniendo el valor BCD de las unidades los bits del 16 al 19 y el de las decenas —`0` ó `1`— el bit 20.

El día de la semana —`DAY`— es almacenado en los bits del 21 al 24, pudiendo tomar valores comprendidos entre `0` y `7` cuyo significado es asignado por el usuario.

La fecha del mes —DATE— se almacena en los bits del 24 al 29, de forma que los bits del 24 al 27 contienen el valor en BCD de las unidades y los bits 28 y 29 el valor en BCD de las decenas (0, 1, 2 ó 3).

Lectura de la fecha y hora actuales

Para poder acceder a los registros del RTC se debe conocer tanto la dirección base que ocupa el periférico en el mapa de memoria como el desplazamiento del registro al que se desea acceder. En este caso, el RTC del ATSAM3X8E abarca 256 direcciones —desplazamientos comprendidos entre 0x00 y 0xFF— a partir de la dirección 0x400E1A60. Los desplazamientos de los diferentes registros del RTC pueden consultarse en el Cuadro 8.8, donde puede verse que el correspondiente al registro RTC_TIMR es 0x08 y el del registro RTC_CALR es 0x0C. Así pues, para leer la fecha actual será necesario realizar una operación de lectura sobre la dirección 0x400E1A6C y para obtener la hora actual será necesario leer el contenido de la dirección 0x400E1A68.

Registro	Alias	Desplazamiento
Control Register	RTC_CR	0x00
Mode Register	RTC_MR	0x04
Time Registert	RTC_TIMR	0x08
Calendar Register	RTC_CALR	0x0C
Time Alarm Register	RTC_TIMALR	0x10
Calendar Alarm Register	RTC_CALALR	0x14
Status Register	RTC_SR	0x18
Status Clear Command Register	RTC_SCCR	0x1C
Interrupt Enable Register	RTC_IER	0x20
Interrupt Disable Register	RTC_IDR	0x24
Interrupt Mask Register	RTC_IMR	0x28
Valid Entry Register	RTC_VER	0x2C
Reserved Register	—	0x30–0xE0
Write Protect Mode Register	RTC_WPMR	0xE4
Reserved Register	—	0xE8–0xFC

Cuadro 8.8: Desplazamientos de los registros del RTC

Debido a que el RTC es independiente del resto del sistema y funciona de forma asíncrona respecto del mismo, para asegurar que la lectura de sus contenidos es correcta, es necesario realizarla por duplicado y comparar ambos resultados. Si son idénticos, es correcto. De lo contrario hay que repetir el proceso, requiriéndose un mínimo de dos lecturas y un máximo de tres para obtener el valor correcto.

Actualización de la fecha y hora actuales

La configuración de la fecha y hora actuales en el RTC requiere de un procedimiento a que se describe a continuación.

1. Inhibir la actualización del RTC. Esto se consigue mediante los bits `UPDCAL` para la fecha y `UPDTIM` para la hora. Ambos se encuentran, como muestra la Figura 8.8, en el registro de control `RTC_CR`. Cada uno de estos bits detiene la actualización del contador correspondiente cuando toma al valor `1`, permitiendo el funcionamiento normal del RTC cuando vale `0`. Así pues, si deseamos establecer la fecha actual, deberemos escribir un `1` en el bit de peso 1 del registro `RTC_CR` antes de modificar el registro `RTC_CALR`. Este registro, junto con los que sirven para configurar las alarmas, dispone de una protección contra escritura que se puede habilitar en el registro `RTC_WPMR` (*RTC Write Protect Mode Register*) introduciendo la clave correcta en el registro, cuyo contenido se muestra en la Figura 8.9. Para que el cambio de modo de protección contra escritura de los registros protegidos —`RTC_CR`, `RTC_CALALR` y `RTC_TIMALR`— se produzca, la clave introducida en el campo `WPKEY` debe ser `0x525443` —'RTC' en ASCII—, mientras el byte de menor peso de la palabra de 32 bits debe tomar el valor `0x00` para permitir la escritura y el valor `0x01` para impedirla. Por defecto, la protección contra escritura está deshabilitada.

2. Esperar la activación de `ACKUPD`, que es el bit de peso 0 del registro de estado `RTC_SR` —mostrado en la Figura 8.10—. En caso de que la generación de interrupciones esté activada, no será necesario consultar el registro, dado que se producirá una interrupción.

3. Una vez se haya detectado que el bit `ACKUPD` ha tomado el valor `1`, es necesario restablecer este indicador escribiendo un `1` en el bit de peso 0, denominado `ACKCLR`, del registro `RTC_SCCR` (*RTC Status Clear Command Register*) cuyo contenido se muestra en la Figura 8.11.

4. Ahora se puede escribir el nuevo valor de la hora y/o fecha actuales en sus correspondientes registros —`RTC_TIMR` y `RTC_CALR`

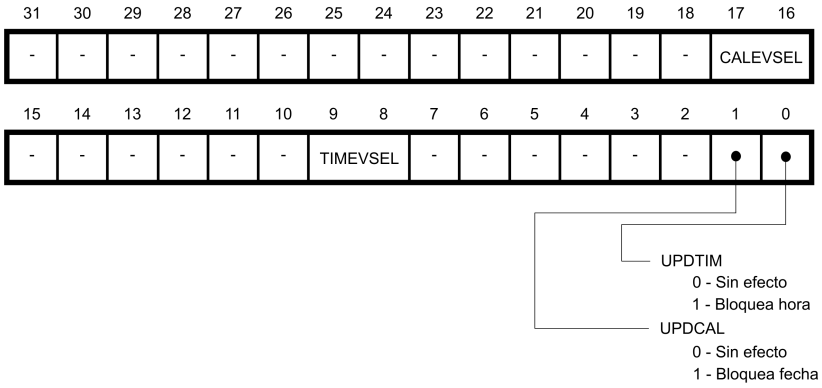

Figura 8.8: Contenido del registro RTC Control Register

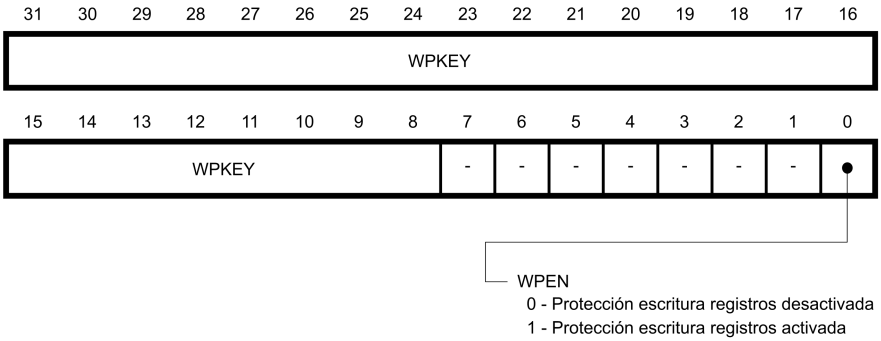

Figura 8.9: Contenido del registro RTC Write Protect Mode Register

respectivamente—. El RTC comprueba que los valores que se escriben sean correctos. De no ser así, se activa el indicador correspondiente en el `RTC_VER` (*RTC Valid Entry Register*) cuyo contenido se muestra en la Figura 8.12.

De esta forma, si alguno de los campos de la hora indicada no es correcto, se activará (presentando un valor `1`) el indicador `NVTIM` (*Non-valid Time*) y si uno o más de los campos de la fecha no es correcto, se activará el indicador `NVCAL` (*Non-valid Calendar*). El RTC quedará bloqueado mientras se mantenga esta situación y los indicadores solamente volverán a la normalidad cuando se introduzca un valor correcto.

5. Restablecer el valor de los bits de inhibición de la actualización

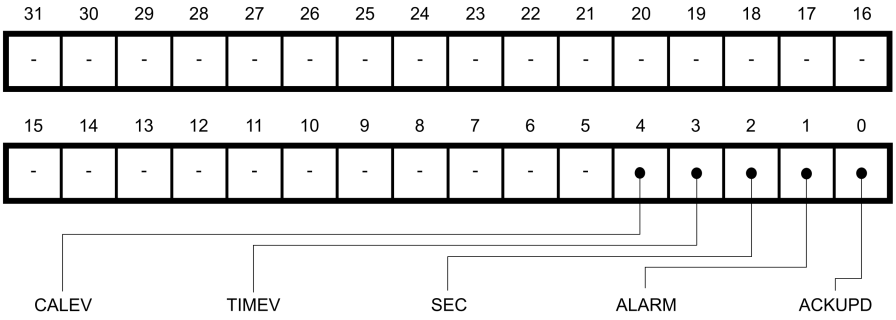

Figura 8.10: Contenido del registro RTC Status Register

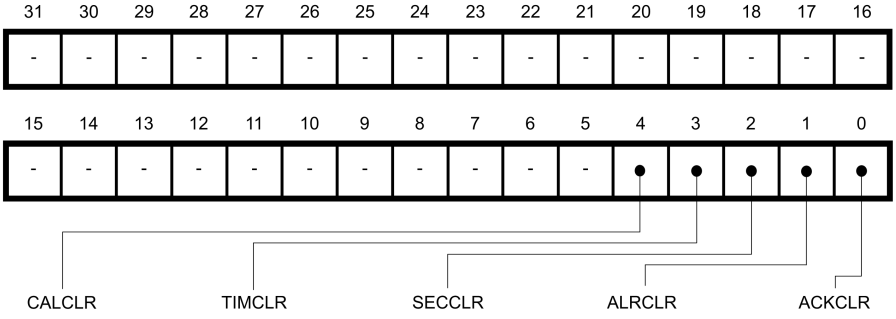

Figura 8.11: Contenido del registro RTC Status Clear Command Register

—`UPDCAL` y/o `UPDTIM`— para permitir la reanudación del funcionamiento del RTC. Si solamente se modifica el valor de la fecha actual, la porción del RTC dedicada al cálculo de la hora actual sigue en funcionamiento, mientras que si solo se modifica la hora, el calendario también es detenido. La modalidad de 12/24 horas se puede seleccionar mediante `HRMOD`, bit de peso 0 del registro `RTC_MR` (*Mode Register*) cuyo contenido se muestra en la Figura 8.13. Escribiendo en `HRMOD` el valor `1` se configura el RTC en modo 24 horas, mientras que el valor `0` establece la configuración en el modo de 12 horas.

Alarmas

El RTC posee la capacidad de establecer valores de alarma para cinco campos: mes, día del mes, hora, minuto, segundo. Estos valores están re-

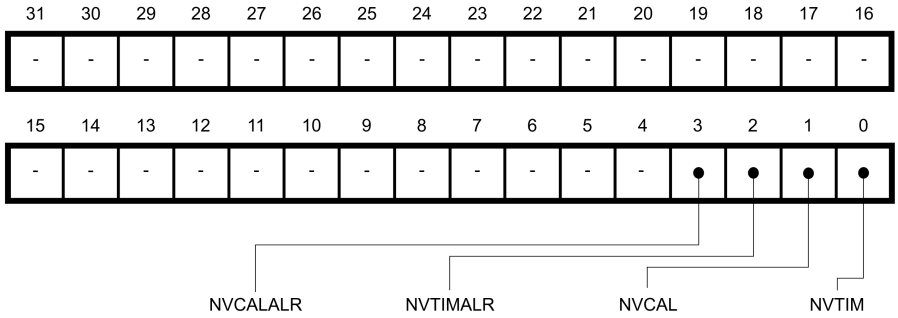

Figura 8.12: Contenido del registro RTC Valid Entry Register

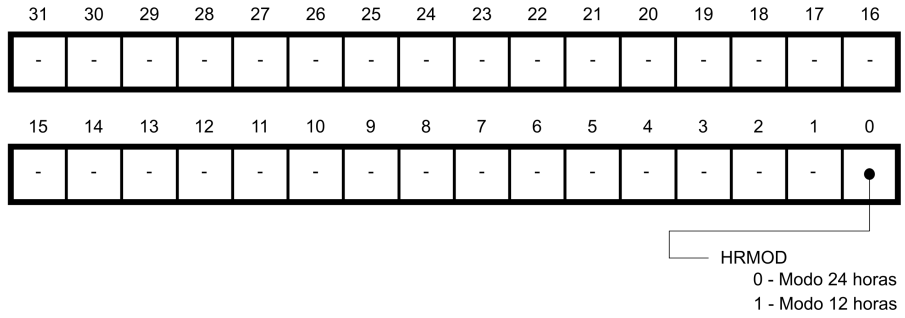

Figura 8.13: Contenido del registro RTC Mode Register

partidos en dos registros: `RTC_TIMALR` (*RTC Time Alarm Register*) cuyo contenido es mostrado en la Figura 8.14, y `RTC_CALALR` (*RTC Calendar Alarm Register*) cuyo contenido es mostrado en la figura 8.15.

Cada uno de los campos configurables posee un bit de activación asociado, de forma que su valor puede ser considerado o ignorado en la activación de la alarma. Así pues, si por ejemplo escribimos un `1` en `DATEEN` —bit 23 del registro `RTC_CALALR`— y el valor '18' en BCD en `DATE` —bits 16 a 20 del mismo registro— generaremos una alarma el día 18 de cada mes.

Los valores introducidos en los campos configurables se comprueban al igual que los de fecha y hora anteriormente comentados y, si se detecta un error, se activan los indicadores correspondientes del registro `RTC_VER` (*RTC Valid Entry Register*) mostrado en la Figura 8.12. Si se activan todos los campos configurables y se establece un valor válido para cada uno de ellos, se configura una alarma para un instante determinado, llegado el cual se activará el bit `ALARM` (bit 1 del registro `RTC_SR` (*RTC Status Register*) cuyo contenido se muestra en la Figura 8.10) y, en ca-

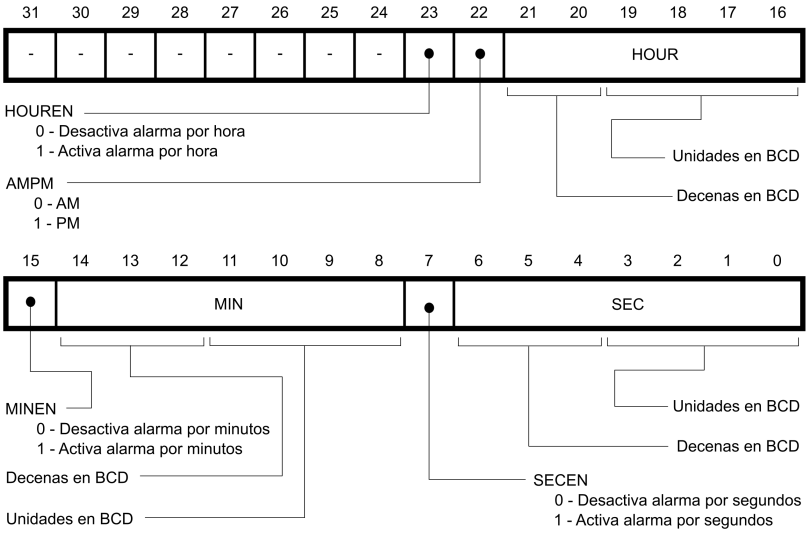

Figura 8.14: Contenido del registro RTC Time Alarm Register

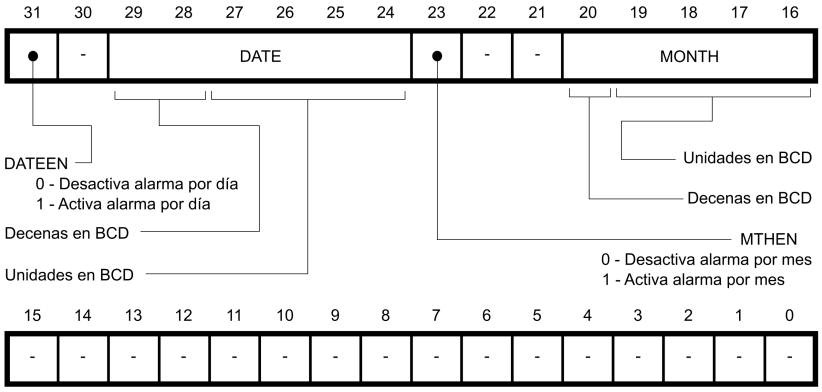

Figura 8.15: Contenido del registro RTC Calendar Alarm Register

so de estar activada la generación de interrupciones, se producirá una interrupción. Para restablecer los indicadores del registro `RTC_SR` (*RTC Status Register*) hay que escribir un `1` en cada uno de los bits correspondientes del registro `RTC_SCCR` (*RTC Status Clear Command Register*).

Si se produce una segunda alarma antes de que se haya leído el registro `RTC_SR` (*RTC Status Register*), mostrado en la Figura 8.10, tras una alarma, se activará `SEC` —bit de peso 2 del registro `RTC_SR`— que indica

que al menos dos alarmas se han producido desde que se restableció el valor del indicador por última vez.

Eventos periódicos

Además de las alarmas en instantes programados, como se ha visto en el apartado anterior, el RTC también posee la capacidad de producir alarmas periódicas con diferentes cadencias configurables a través del registro `RTC_CR` (*RTC Control Register*), cuyo contenido se muestra en la Figura 8.8. En sus bits 8 y 9 se encuentra el valor del campo `TIMEVSEL` que activa/desactiva la generación de eventos periódicos de hora y en los bits 16 y 17 el campo `CALEVSEL` que activa/desactiva la generación de eventos periódicos de calendario.

Un evento de hora puede ser a su vez de cuatro tipos diferentes, mostrados en el Cuadro 8.9, dependiendo del valor que tome el campo `TIMEVSEL`.

Valor	Nombre	Evento
0	`MINUTE`	Cada cambio de minuto
1	`HOUR`	Cada cambio de hora
2	`MIDNIGHT`	Cada día a medianoche
3	`NOON`	Cada día a mediodía

Cuadro 8.9: Tipos de eventos periódicos de hora

De la misma forma, un evento de fecha puede ser a su vez de tres tipos diferentes, mostrados en el Cuadro 8.10, dependiendo del valor que tome el campo `CALEVSEL`.

Valor	Nombre	Evento
0	`WEEK`	Cada lunes a las 0:00:00
1	`MONTH`	El día 1 de cada mes a las 0:00:00
2	`YEAR`	Cada 1 de enero a las 0:00:00
3	—	Valor no permitido

Cuadro 8.10: Tipos de eventos periódicos de fecha

Al igual que ocurre con las alarmas de tiempo concreto, la notificación de que se ha producido un evento periódico se produce a través del registro `RTC_SR` (*RTC Status Register*) mostrado en la Figura 8.10, donde, en caso de que se haya producido un evento periódico de hora, se activará `TIMEV` —bit de peso 3— y en caso de que se haya detectado

un evento periódico de fecha de acuerdo con lo configurado, se activará `CALEV` —bit de peso 4—.

Al leer este registro, el hecho de que uno o más de estos bits estén activos, es decir, que presenten el valor `1`, nos indicará que la condición de evento periódico se ha producido al menos en una ocasión desde la última vez que se leyó el contenido del registro. La lectura del registro restablece el valor de todos sus indicadores a `0`.

Interrupciones en el RTC

El RTC posee la capacidad de generar interrupciones cuando se producen una serie de circunstancias:

- Actualización de fecha/hora.
- Evento de tiempo.
- Evento de calendario.
- Alarma.
- Segundo evento de alarma periódica.

Para gestionar la generación de estas interrupciones y la atención de las mismas, existen los registros de interrupción del RTC, que a continuación se describen.

La activación de la generación de interrupciones se consigue a través del registro `RTC_IER` (*RTC Interrupt Enable Register*), cuyo contenido se muestra en la Figura 8.16, donde puede apreciarse que se dispone de cinco bits de configuración para activar la generación de interrupciones:

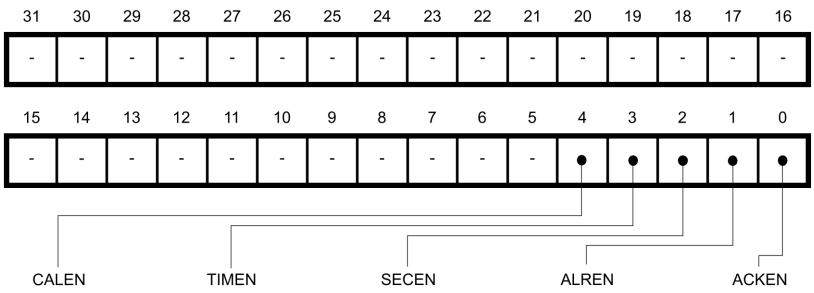

Figura 8.16: Contenido del registro RTC Interrupt Enable Register

- *Inhibición de la actualización del RTC*: escribiendo un `1` en `ACKEN` —bit de peso 0— activamos la generación de una interrupción cuando se active el bit `ACKUPD` del registro `RTC_SR` (*RTC Status Register*), mostrado en la Figura 8.10, como consecuencia de haber inhibido la actualización del RTC mediante uno de los bits `UPDCAL` o `UPDTIM` del registro `RTC_CR` —véase la Figura 8.8— o ambos.

- *Condición de alarma*: escribiendo un `1` en `ALREN` —bit de peso 1— activamos la generación de una interrupción al activarse el bit `ALARM` del registro `RTC_SR` (*RTC Status Register*). Esto ocurre cuando se cumple la condición de generación de alarma especificada en uno de los registros `RTC_TIMALR` (*RTC Time Alarm Register*) o `RTC_CALALR` (*RTC Calendar Alarm Register*).

- *Segunda alarma*: escribiendo un `1` en `SECEN` —bit de peso 2—, activamos la generación de una interrupción al activarse el bit `SEC` del registro `RTC_SR` (*RTC Status Register*), lo cual indica que se ha cumplido la condición de alarma en una segunda ocasión sin que el registro *RTC Status Register* haya sido leído.

- *Evento periódico de hora*: escribiendo un `1` en `TIMEN` —bit de peso 3— se activa la generación de una interrupción cuando se activa el bit `TIMEV` del registro `RTC_SR`, lo cual ocurrirá cuando se cumpla la condición periódica configurada en el campo `TIMEVSEL` del registro `RTC_CR` ((*RTC Control Register*) —véanse la Figura 8.8 y el Cuadro 8.9—.

- *Evento periódico de fecha*: escribiendo un `1` en `CALEN` —bit de peso 4— se activa la generación de una interrupción cuando se activa el bit `CALEV` del registro `RTC_SR` lo cual ocurrirá cuando se cumpla la condición periódica configurada en el campo `CALEVSEL` del registro `RTC_CR` (*RTC Control Register*) —véanse la Figura 8.8 y el Cuadro 8.10—.

Cada vez que se produzca una interrupción, en la correspondiente rutina de servicio se deberá acceder al registro `RTC_SR` (*RTC Status Register*) para averiguar cuál es la causa de la misma. Es posible que varias circunstancias hayan concurrido para la generación de la interrupción, con lo cual es aconsejable comprobar todos y cada uno de los bits del registro de estado. Una vez averiguadas las causas de la interrupción y tomadas las acciones pertinentes, antes de regresar de la rutina de servicio, se deben restablecer los indicadores escribiendo ceros en el registro `RTC_SCCR` (*RTC Status Clear Command Register*), cuyo contenido se muestra en la Figura 8.11, para dejar el sistema en disposición de que se produzcan nuevas interrupciones.

8.2.4. El Real-Time Timer (RTT) del Atmel ATSAM3X8E

El *Real-Time Timer* (RTT) es un temporizador del ATSAM3X8E simple y versátil, por lo que se puede utilizar de forma sencilla —y con plena libertad dado que no es utilizado por el sistema Arduino—. Se trata básicamente de un registro contador de 32 bits, que tiene una frecuencia base de actualización de 32.768Hz, como el RTC. Esta frecuencia se puede dividir gracias a un prescaler de 16 bits. El dispositivo dispone además de un registro de alarma para generar interrupciones cuando la cuenta del temporizador alcanza el valor almacenado en él, y es capaz además de generar interrupciones periódicas cada vez que se incrementa el valor del temporizador. Utiliza los cuatro registros que se describen a continuación:

- *Mode Register* (`RTT_MR`): es el registro de control del dispositivo. Los 16 bits más bajos almacenan el prescaler. Con un valor de `0x8000` se tiene una frecuencia de actualización de un segundo, lo que indica que está pensado para temporizaciones relacionadas con tiempos de usuario más que de sistema. No obstante, se puede poner en estos bits cualquier valor superior a 2. El bit 18 —`RTTRST`— sirve para reiniciar el sistema —escribiendo un `1`—, poniendo el contador a `0` y actualizando el valor del prescaler. El bit 17 —`RTTINCIEN`— es la habilitación de interrupción por incremento, y el bit 16 —`ALMIEN`— la de interrupción por alarma. Ambas se habilitan con un `1`.

- *Alarm Register* (`RTT_AR`): almacena el valor de la alarma, de 32 bits. Cuando el contador alcance este valor, se producirá una interrupción en caso de estar habilitada.

- *Value Register* (`RTT_VR`): guarda el valor del contador, que se va incrementando con cada pulso —según la frecuencia base dividida por el prescaler—, de forma cíclica.

- *Status Register* (`RTT_SR`): es el registro de estado. Su bit 1 —`RTTINC`— indica que se ha producido un incremento del valor, y su bit 0 —`ALMS`— que ha ocurrido una alarma. Ambas circunstancias se señalan con un `1`, que se pone a `0` al leer el registro.

En el Cuadro 8.11 aparecen las direcciones de los registros citados.

Es interesante realizar un comentario acerca del uso del RTT y de la forma en que se generan las interrupciones. La interrupción por incremento está pensada para generar una interrupción periódica; según el valor del prescaler, el periodo puede ser desde inferior a una décima de milisegundo hasta casi dos segundos. Eléctricamente la interrupción

Registro	Alias	Dirección
Mode Register	RTT_MR	0x400E 1A30
Alarm Register	RTT_AR	0x400E 1A34
Value Register	RTT_VR	0x400E 1A38
Status Register	RTT_SR	0x400E 1A3C

Cuadro 8.11: Registros del temporizador en tiempo real del ATSAM3X8E y sus direcciones de E/S

se genera por flanco, por lo que al producirse basta con leer el RTT_SR para evitar que se genere hasta el próximo incremento, comportamiento típico de una interrupción periódica.

La interrupción de alarma, sin embargo, se produce por nivel mientras el valor del contador sea igual al de la alarma. Leer en este caso el RTT_SR no hace que deje de señalarse la interrupción, que se seguirá disparando hasta que llegue otro pulso de la frecuencia de actualización. Este comportamiento no se evita cambiando el valor del RTT_AR pues la condición de interrupción se actualiza con el mismo reloj que incrementa el temporizador. Esto quiere decir que la interrupción por alarma está pensada para producirse una vez y deshabilitarse, hasta que el programa decida configurar una nueva alarma por algún motivo. Su uso como interrupción periódica no es, por tanto, recomendable.

8.3. Gestión de excepciones e interrupciones en el ATSAM3X8E

La arquitectura ARM especifica un modelo de excepciones que lógicamente incluye el tratamiento de interrupciones. Es un modelo elaborado y versátil, pero a la vez sencillo de usar, dado que la mayor parte de los mecanismos son llevados a cabo de forma automática por el hardware del procesador.

Se trata de un modelo vectorizado, con expulsión —*preemption*— en base a prioridades. Cada causa de excepción, de las que las interrupciones son un subconjunto, tiene asignado un número de orden que identifica un vector de excepción en una zona determinada de la memoria. Cuando se produce una excepción, el procesador carga el valor almacenado en ese vector y lo utiliza como dirección de inicio de la rutina de tratamiento. Al mismo tiempo, cada excepción tiene una prioridad que determina cuál de ellas será atendida en caso de detectarse varias a la vez, y permite que una rutina de tratamiento sea interrumpida —expulsada— si se detecta una excepción con mayor prioridad, volviendo

a aquélla al terminar de tratar la más prioritaria. El orden de la prioridad es inverso a su valor numérico, así una prioridad de 0 es mayor que una de 7, por ejemplo.

De acuerdo con este modelo, una excepción es marcada como *pendiente —pending—* cuando ha sido detectada, pero no ha sido tratada todavía, y como *activa —active—* cuando su rutina de tratamiento ya ha comenzado a ejecutarse. Debido a la posibilidad de expulsión, en un momento puede haber más de una excepción activa en el sistema. Cuando una excepción es a la vez pendiente y activa, ello significa que se ha vuelto a detectar la excepción mientras se trataba la anterior.

Cuando se detecta una excepción, si no se está atendiendo a otra de mayor prioridad, el procesador guarda automáticamente en la pila los registros `r0` a `r3` y `r12`; la dirección de retorno, el registro de estado y el registro `lr`. Entonces realiza el salto a la dirección guardada en el vector de interrupción y pasa la excepción de pendiente a activa. En el registro `lr` se escribe entonces un valor especial, llamado `EXC_RETURN`, que indica que se está tratando una excepción, y que será utilizado para volver de la rutina de tratamiento. La rutina de tratamiento tiene la estructura de una rutina normal, dado que los registros ya han sido salvados adecuadamente. Cuando se termina el tratamiento se retorna a la ejecución normal con una instrucción «`pop`» que incluya el registro `pc`. De esta manera, el procesador recupera de forma automática los valores de los registros que había guardado previamente, continuando con la ejecución de forma normal.

En el Cuadro 8.12 aparecen las excepciones del sistema, con su número, su prioridad y el número de interrupción asociado. Por conveniencia, la arquitectura asigna a las excepciones un número de interrupción negativo, quedando el resto para las interrupciones de dispositivos.

8.3.1. El Nested Vectored Interrupt Controller NVIC

Como hemos visto en el apartado anterior, la arquitectura ARM especifica el modelo de tratamiento de las excepciones. Del mismo modo, incluye entre los dispositivos periféricos del núcleo —*Core Peripherals*— de la versión *Cortex-M3* de dicha arquitectura, un controlador de interrupciones llamado *Nested Vectored Interrupt Controller*, *NVIC*. Cada implementación distinta de la arquitectura conecta al NVIC las interrupciones generadas por sus distintos dispositivos periféricos. En el caso del ATSAM3X8E, dichas conexiones son fijas, de manera que cada dispositivo tiene un número de interrupción —y por tanto, un vector— fijo y conocido de antemano.

El NVIC, además de implementar el protocolo adecuado para señalar las interrupciones al núcleo, contiene una serie de registros que permiten al software del sistema configurar y tratar las interrupciones según las

Excepción	IRQ	Tipo	Prioridad	Vector (offset)
1	—	Reset	−3	`0x0000 0004`
2	−14	NMI	−2	`0x0000 0008`
3	−13	Hard fault	−1	`0x0000 000C`
4	−12	Memory management fault	Configurable	`0x0000 0010`
5	−11	Bus fault	Configurable	`0x0000 0014`
6	−10	Usage fault	Configurable	`0x0000 0018`
11	−5	SVCall	Configurable	`0x0000 002C`
14	−2	PendSV	Configurable	`0x0000 0038`
15	−1	SysTick	Configurable	`0x0000 003C`
16	0	IRQ0	Configurable	`0x0000 0040`
17	1	IRQ1	Configurable	`0x0000 0044`
...	...	...	...	...

Cuadro 8.12: Algunas de las excepciones del ATSAM3X8E y sus vectores de interrupción

necesidades de la aplicación. Para ello, dispone de una serie de registros especificados en la arquitectura, que en el caso del ATSAM3X8E permiten gestionar las interrupciones de los periféricos del microcontrolador. Veamos cuáles son esos registros y su función.

Para la habilitación individual de las interrupciones se dispone de los conjuntos de registros:

- *Interrupt Set Enable Registers* (`ISERx`): escribiendo un `1` en cualquier bit de uno de estos registros se habilita la interrupción asociada.
- *Interrupt Clear Enable Registers* (`ICERx`): escribiendo un `1` en cualquier bit de uno de estos registros se deshabilita la interrupción asociada.

Al leer cualquiera de los registros anteriores se obtiene la máscara de interrupciones habilitadas, indicadas con `1` en los bits correspondientes.

Para gestionar las interrupciones pendientes se tienen:

- *Interrupt Set Pending Registers* (`ISPRx`): escribiendo un `1` en cualquier bit de uno de estos registros se marca la interrupción aso-

ciada como pendiente. Esto permite forzar el tratamiento de una interrupción aunque no se haya señalado físicamente.

- *Interrupt Clear Pending Registers* (`ICPRx`): escribiendo un `1` en cualquier bit de uno de estos registros se elimina la interrupción asociada del estado pendiente. Esto permite evitar que se produzca el salto por hardware a la rutina de tratamiento.

Al leer cualquiera de los registros anteriores se obtiene la lista de interrupciones pendientes, indicadas con `1` en los bits correspondientes. Una interrupción puede estar en estado pendiente aunque no esté habilitada en la máscara vista más arriba.

La lista de interrupciones activas se gestiona por hardware, pero se puede consultar en los registros de solo lectura *Interrupt Active Bit Registers* (`ICPRx`). En ellos, un `1` indica que la interrupción correspondiente está siendo tratada en el momento de la lectura. Las prioridades asociadas a las interrupciones se configuran en los registros *Interrupt Priority Registers* (`IPRx`). Cada interrupción tiene asociado un campo de 8 bits en estos registros, aunque en la implementación actual solo los 4 de mayor peso almacenan el valor de la prioridad, entre 0 y 15.

Por último, existe un registro que permite generar interrupciones por software, llamado *Software Trigger Interrupt Register* (`STIR`). Escribiendo un valor en sus 9 bits menos significativos se genera la interrupción con dicho número.

Para dar cabida a los bits asociados a todas las posibles interrupciones —que pueden llegar a ser hasta 64 en la serie de procesadores SAM3X—, el NVIC implementado en el ATSAM3X8E dispone de 2 registros para cada uno de los conjuntos, excepto para el de prioridades que comprende 8 registros. Dado un número de IRQ es sencillo calcular cómo encontrar los registros y bits que almacenan la información que se desea leer o modificar, sin embargo ARM recomienda utilizar ciertas funciones en lenguaje C que suministra —en el modelo de código del sistema llamado *CMSIS*— y que son las que utilizaremos en nuestros programas. A continuación se describen las primitivas más usadas.

- «`void NVIC_EnableIRQ(IRQn_t IRQn)`». Habilita la interrupción cuyo número se le pasa como parámetro.
- «`void NVIC_DisableIRQ(IRQn_t IRQn)`». Deshabilita la interrupción cuyo número se le pasa como parámetro.
- «`uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)`». Devuelve un valor distinto de `0` si la interrupción cuyo número se pasa está pendiente; `0` en caso contrario.

- «`void NVIC_SetPendingIRQ (IRQn_t IRQn)`». Marca como pendiente la interrupción cuyo número se pasa como parámetro.
- «`void NVIC_ClearPendingIRQ (IRQn_t IRQn)`». Elimina la marca de pendiente de la interrupción cuyo número se pasa como parámetro.
- «`void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)`». Asigna la prioridad indicada a la interrupción cuyo número se pasa como parámetro.
- «`uint32_t NVIC_GetPriority (IRQn_t IRQn)`». Devuelve la prioridad de la interrupción cuyo número se pasa como parámetro.

8.3.2. Interrupciones del ATSAM3X8E en el entorno Arduino

Como se ha visto, buena parte de las tareas de salvaguarda y recuperación del estado en la gestión de interrupciones son realizadas por el hardware en las arquitecturas ARM. Gracias a esto el diseño de rutinas de tratamiento queda muy simplificado.

La estructura de una RTI es idéntica a la de una rutina normal, con la única restricción de que no admite ni devuelve parámetros. El hecho de que los registros que normalmente no se guardan —`r0 ... r3`— sean preservados automáticamente hace que el código de entrada y salida de una RTI no difiera del de una rutina, y por ello, las únicas diferencias entre ambas estriban en su propio código.

Teniendo en cuenta esta estructura, y teniendo en cuenta las funciones de gestión del NVIC que se han descrito más arriba, lo único que se necesita saber para implementar una RTI es el número de interrupción. Además, para configurarla habría que modificar el vector correspondiente para que apunte a nuestra función. De nuevo, en el estándar CMSIS se dan las soluciones para estos requisitos. Todos los dispositivos tienen definido su número de IRQ —que recordemos es fijo— de forma regular. Del mismo modo, los nombres de las funciones de tratamiento están igualmente predefinidos, de manera que para crear una RTI para un cierto dispositivo simplemente hemos de programar una función con el nombre adecuado. El proceso de compilación de nuestro programa se encarga de configurar el vector de manera transparente para el programador.

Los Cuadros 8.13, 8.14 y 8.15 muestran los símbolos que se deben usar para referirse a las distintas IRQ según el dispositivo, la descripción del dispositivo y el nombre de la rutina de tratamiento.

Una vez creada la RTI solo es necesario configurar el NVIC adecuadamente. El proceso suele consistir en deshabilitar primero la interrupción

Símbolo	Núm	Dispositivo	RTI
`SUPC_IRQn`	0	Supply Controller (SUPC)	«`void SUPC_Handler()`»
`RSTC_IRQn`	1	Reset Controller (RSTC)	«`void RSTC_Handler()`»
`RTC_IRQn`	2	Real Time Clock (RTC)	«`void RTC_Handler()`»
`RTT_IRQn`	3	Real Time Timer (RTT)	«`void RTT_Handler()`»
`WDT_IRQn`	4	Watchdog Timer (WDT)	«`void WDT_Handler()`»
`PMC_IRQn`	5	Power Management Controller (PMC)	«`void PMC_Handler()`»
`EFC0_IRQn`	6	Enhanced Flash Controller 0 (EFC0)	«`void EFC0_Handler()`»
`EFC1_IRQn`	7	Enhanced Flash Controller 1 (EFC1)	«`void EFC1_Handler()`»
`UART_IRQn`	8	Universal Asynchronous Receiver Transmitter	«`void UART_Handler()`»
`SMC_IRQn`	9	Static Memory Controller (SMC)	«`void SMC_Handler()`»
`PIOA_IRQn`	11	Parallel I/O Controller A, (PIOA)	«`void PIOA_Handler()`»
`PIOB_IRQn`	12	Parallel I/O Controller B (PIOB)	«`void PIOB_Handler()`»
`PIOC_IRQn`	13	Parallel I/O Controller C (PIOC)	«`void PIOC_Handler()`»
`PIOD_IRQn`	14	Parallel I/O Controller D (PIOD)	«`void PIOD_Handler()`»

Cuadro 8.13: IRQs del ATSAM3X8E y sus rutinas de tratamiento asociadas (Parte I)

correspondiente, limpiar el estado pendiente por si se produjo alguna falsa interrupción durante el arranque del sistema y establecer la prioridad —que por defecto suele ser `0` en el entorno Arduino—. Una vez hecho esto, hemos de configurar el dispositivo de la manera que se desee y habilitar la interrupción correspondiente.

A continuación se muestran unos fragmentos de código que permitirían establecer una RTI para el dispositivo `PIOB`.

RTI-PIOB.c

```
1 void setup() {
2         /* Otra configuración del sistema*/
```

Símbolo	Núm	Dispositivo	RTI
USART0_IRQn	17	USART 0 (USART0)	«void USART0_Handler()»
USART1_IRQn	18	USART 1 (USART1)	«void USART1_Handler()»
USART2_IRQn	19	USART 2 (USART2)	«void USART2_Handler()»
USART3_IRQn	20	USART 3 (USART3)	«void USART3_Handler()»
HSMCI_IRQn	21	Multimedia Card Interface (HSMCI)	«void HSMCI_Handler()»
TWI0_IRQn	22	Two-Wire Interface 0 (TWI0)	«void TWI0_Handler()»
TWI1_IRQn	23	Two-Wire Interface 1 (TWI1)	«void TWI1_Handler()»
SPI0_IRQn	24	Serial Peripheral Interface (SPI0)	«void SPI0_Handler()»
SSC_IRQn	26	Synchronous Serial Controller (SSC)	«void SSC_Handler()»
TC0_IRQn	27	Timer Counter 0 (TC0)	«void TC0_Handler()»
TC1_IRQn	28	Timer Counter 1 (TC1)	«void TC1_Handler()»
TC2_IRQn	29	Timer Counter 2 (TC2)	«void TC2_Handler()»
TC3_IRQn	30	Timer Counter 3 (TC3)	«void TC3_Handler()»
TC4_IRQn	31	Timer Counter 4 (TC4)	«void TC4_Handler()»
TC5_IRQn	32	Timer Counter 5 (TC5)	«void TC5_Handler()»
TC6_IRQn	33	Timer Counter 6 (TC6)	«void TC6_Handler()»
TC7_IRQn	34	Timer Counter 7 (TC7)	«void TC7_Handler()»
TC8_IRQn	35	Timer Counter 8 (TC8)	«void TC8_Handler()»

Cuadro 8.14: IRQs del ATSAM3X8E y sus rutinas de tratamiento asociadas(Parte II)

```
3         NVIC_DisableIRQ(PIOB_IRQn);
4         NVIC_ClearPendingIRQ(PIOB_IRQn);
5         NVIC_SetPriority(PIOB_IRQn, 0);
6         PIOsetupInt(EDGE, 1);
7         NVIC_EnableIRQ(PIOB_IRQn);
8 }
9
10 // RTI en C
11 void PIOB_Handler() {
12         /* Código específico del tratamiento */
```

Símbolo	Núm	Dispositivo	RTI
PWM_IRQn	36	Pulse Width Modulation Controller (PWM)	«void PWM_Handler()»
ADC_IRQn	37	ADC Controller (ADC)	«void ADC_Handler()»
DACC_IRQn	38	DAC Controller (DACC)	«void DACC_Handler()»
DMAC_IRQn	39	DMA Controller (DMAC)	«void DMAC_Handler()»
UOTGHS_IRQn	40	USB OTG High Speed (UOTGHS)	«void UOTGHS_Handler()»
TRNG_IRQn	41	True Random Number Generator (TRNG)	«void TRNG_Handler()»
EMAC_IRQn	42	Ethernet MAC (EMAC)	«void EMAC_Handler()»
CAN0_IRQn	43	CAN Controller 0 (CAN0)	«void CAN0_Handler()»
CAN1_IRQn	44	CAN Controller 1 (CAN1)	«void CAN1_Handler()»

Cuadro 8.15: IRQs del ATSAM3X8E y sus rutinas de tratamiento asociadas (Parte III)

```
13 }
```

RTI-PIOB.s

```
1 @ RTI en ensamblador
2 PIOB_Handler:
3         push {lr}
4         /* Código específico del tratamiento */
5         pop {pc}
```

8.4. El controlador de DMA del ATSAM3X8E

El *AHB DMA Controller* (*DMAC*) es el dispositivo controlador de acceso directo a memoria en el ATSAM3X8E. Se trata de un dispositivo con seis canales, con capacidad para almacenamiento intermedio de 8 o 32 bytes —en los canales 3 y 5— y que permite transferencias entre dispositivos y memoria, en cualquier configuración. Cada movimiento de información requiere de la lectura de datos de la fuente a través de los buses correspondientes, su almacenamiento intermedio y su posterior escritura en el destino, lo que requiere siempre dos accesos de transferencia de datos.

Además de un conjunto de registros de configuración globales, cada canal dispone de seis registros asociados que caracterizan la transacción.

Mediante estos registros, además de indicar el dispositivo fuente y destino y las direcciones de datos correspondientes, se pueden programar transacciones múltiples de bloques de datos contiguos o dispersos, tanto en la fuente como en el destino.

El dispositivo, además de gestionar adecuadamente los accesos a los distintos buses y dispositivos, es capaz de generar interrupciones para indicar posibles errores o la finalización de las transacciones de DMA programadas.

Para una información más detallada, fuera del objetivo de esta breve introducción, se puede consultar el *apartado 24. AHB DMA Controller (DMAC)* en la página 349 del manual de Atmel.

Bibliografía

[Adv95] Advanced RISC Machines Ltd (ARM) (1995). *ARM 7TDMI Data Sheet*.
URL http://www.ndsretro.com/download/ARM7TDMI.pdf

[Atm11] Atmel Corporation (2011). *ATmega 128: 8-bit Atmel Microcontroller with 128 Kbytes in-System Programmable Flash*.
URL http://www.atmel.com/Images/doc2467.pdf

[Atm12] Atmel Corporation (2012). *AT91SAM ARM-based Flash MCU datasheet*.
URL http://www.atmel.com/Images/doc11057.pdf

[Bar14] S. Barrachina Mir, G. León Navarro y J. V. Martí Avilés (2014). *Conceptos elementales de computadores*.
URL http://lorca.act.uji.es/docs/conceptos_elementales_de_computadores.pdf

[Cle14] A. Clements (2014). *Computer Organization and Architecture. Themes and Variations. International edition.* Editorial Cengage Learning. ISBN 978-1-111-98708-4.

[Shi13] S. Shiva (2013). *Computer Organization, Design, and Architecture, Fifth Edition*. Taylor & Francis. ISBN 9781466585546.
URL http://books.google.es/books?id=m5KlAgAAQBAJ