Práctica 5

Gestión avanzada de interrupciones y teclado matricial

5.1. Objetivos de la práctica

- Profundizar en el sistema de E/S de la placa S3CEV40.
- Manejo del teclado.
- Profundizar en el sistema de interrupciones.
- Gestión de varias fuentes de interrupción.

5.2. Teclado matricial

5.2.1. Decripción

La placa S3CEV40 contiene un teclado matricial. En este tipo de teclados las teclas están dispuestas según un array bidimensional como el mostrado en el Figura 5.1: el teclado es un array de líneas horizontales y líneas verticales junto con 16 interruptores (SB1-SB16) que al ser pulsados conectarán una línea horizontal con una línea vertical.

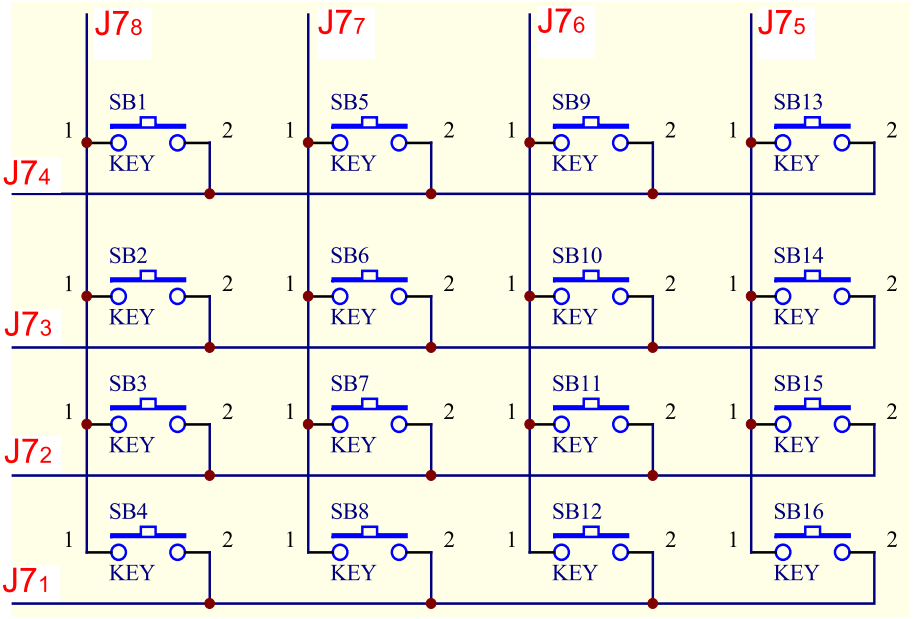

Figura 5.1: Circuito del teclado 4x4

5.2.2. Conexión del teclado

Este teclado, situado en una placa suplementaria, se conecta al resto de componentes a través del conector con nombre de instancia J7, situado en la parte inferior derecha de la placa. Dicho conector posee ocho pines y está conectado a los componentes que se muestran en la Figura 5.2.

Los pines 5-8 del conector J7 están asociados a las columnas del teclado (véase Figura 5.1) y los pines 1-4 están asociados a las filas. Por otro lado, los pines 5-8 están

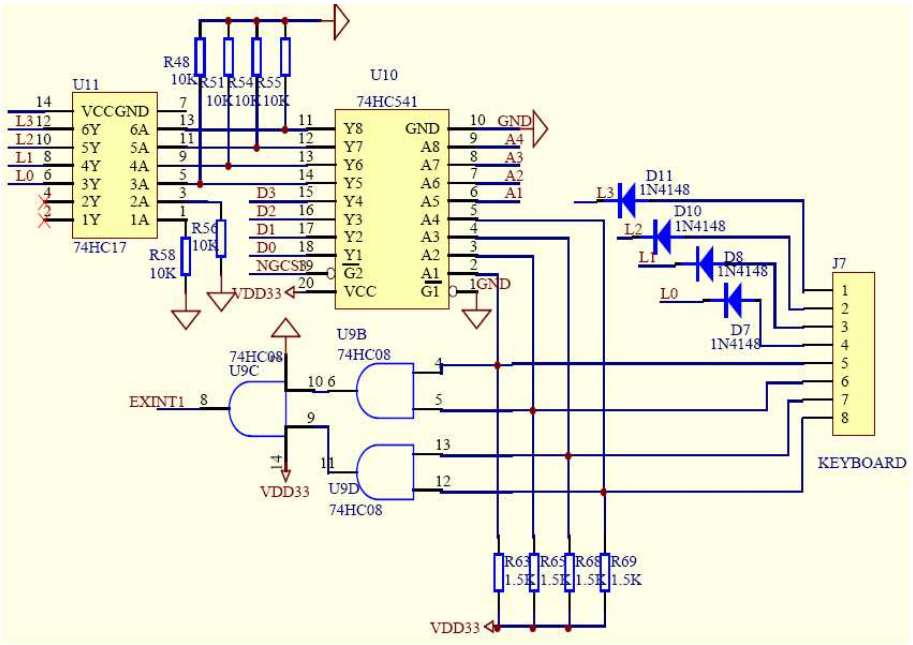

Figura 5.2: Conexión del teclado

conectados a resistencias de "*pull-up*" a fin de que, en ausencia de driver, la tensión en ellos sea una tensión de alta (VDD33). Estos pines, a través del circuito integrado U9[1], generan la señal EINT1 del controlador de interrupciones (dispositivo que ya presentamos en la práctica anterior). Asimismo, estos pines están conectados a U10[2]. Las salidas de este CI se conectan a U11[3].

Los pines 1-4 están conectados a resistencias de "*pull-down*" para que, en ausencia de driver, la tensión en ellos sea una tensión de baja. No es una conexión directa sino que se realiza a través de varios componentes. Así, los pines 1-4 están conectados al ánodo de cuatro diodos (DN7/8/10/11)[4] y los cátodos de estos dispositivos están conectado a la señales L0-L3. Obsérvese que estas señales provienen de las salidas 3Y-6Y de U11 y las entradas correspondientes (3A-6A) están conectadas a resistencias de "*pull-down*". De esta forma cuando U10 esté deshabilitado (es decir, cuando $NGCS3 = \overline{G2} = 1$) sus salidas, Y1-Y8, estarán a alta impedancia y por lo tanto las entradas 3A-6A de U11 estarán a una tensión de baja debido a las resistencias de "*pull-down*".

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

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

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

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

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

5.2.3. Funcionamiento del teclado

Como ya mencionamos en la sección anterior, mientras $NGCS3 = 1$ los pines Y1-Y8 de U10 estarán a alta impedancia y los pines 3A-6A de U11 estarán a una tensión de baja debido a las resistencias de "*pull-down*". En consecuencia, las líneas L0-L3 tendrán una tensión de baja, los diodos DN7/8/10/11 estarán en conducción y en los pines 1-4 de J7 habrá una tensión de baja. Cuando una tecla es pulsada, el interruptor asociado cortocircuita la columna con la fila correspondiente, de forma que la columna tendrá una tensión de baja. Luego uno de los pines 5-8 tendrá una tensión de baja que, al propagarse a través de U9, provocará que la señal EINT1 tome el valor de baja y active el sistema de interrupciones mediante el controlador de interrupciones. Para ello previamente es necesario configurar los pines de interrupciones externas mediante el registro EXINT. Este registro configura el método de señalización de una petición de interrupción externa y la polaridad de la señal. Su valor por defecto es 0x00000000, el cual establece que las interrupciones se disparan a nivel bajo.

Una vez la interrupción ha sido detectada debe identificarse la tecla pulsada. Hay dos formas de hacerlo: mediante "*scanning*" o mediante inversión. Veamos cada una de ellas con algo de detalle.

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

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

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

El envío secuencial de un "0" por cada uno de los pines 1-4 de J7 se realizará escribiendo secuencialmente los patrones 4'b0111, 4'b1011, 4'b1101, 4'b1110 sobre las líneas A4-A1 del bus de direcciones. Para cada uno de estos patrones se procederá a la lectura de los pines 5-8, cuyos valores estarán en las líneas D0-D3 del bus de datos. Para que ello sea posible, los valores escritos en el bus de direcciones deberán atravesar U10. U10 habilita sus salidas cuando sus dos entradas $\overline{G1} = \overline{G2} = 0$, luego se debe habilitar U10 haciendo $nGCS3 = 0$. Tal y como se mencionó en la práctica 3, la señal $nGCS3 = 0$ es generada por el controlador de memoria del S3C44B0X cuando se escribe en el bus de direcciones

alguna dirección pertenceciente al rango asociado al teclado, 0x0600_0000-0x07FF_FFFF (véase Tabla 3.4). En otras palabras, para escribir el patrón 4b'0111 en las líneas A4-A1 y que se propague por U10 es necesario escribir en el bus de direcciones la dirección 0x0600_00EF; para escribir el patrón 4'b1011 se debe escribir la dirección 0x0600_00F7, y así sucesivamente.

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

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

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

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

La Tabla 5.1 muestra los valores que habrá que escribir en el bus de direcciones y leer del bus de datos cuando la tecla correspondiente haya sido pulsada. Mientras no haya identificación, en el bus de datos estará el dato 0xF.

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

A continuación se va a ilustrar el método de reconocimiento de tecla mediante un ejemplo. Supóngase que ha sido pulsada la tecla que conecta los pines 1 y 5 del conector

	A4	A3	A2	A1	A0	Address	D3	D2	D1	D0	Data
SB1	1	1	1	0	1	0xFDH	0	1	1	1	0x7H
SB2	1	1	0	1	1	0xFBH	0	1	1	1	0x7H
SB3	1	0	1	1	1	0xF7H	0	1	1	1	0x7H
SB4	0	1	1	1	1	0xEFH	0	1	1	1	0x7H
SB5	1	1	1	0	1	0xFDH	1	0	1	1	0xBH
SB6	1	1	0	1	1	0xFBH	1	0	1	1	0xBH
SB7	1	0	1	1	1	0xF7H	1	0	1	1	0xBH
SB8	0	1	1	1	1	0xEFH	1	0	1	1	0xBH
SB9	1	1	1	0	1	0xFDH	1	1	0	1	0xDH
SB10	1	1	0	1	1	0xFBH	1	1	0	1	0xDH
SB11	1	0	1	1	1	0xF7H	1	1	0	1	0xDH
SB12	0	1	1	1	1	0xEFH	1	1	0	1	0xDH
SB13	1	1	1	0	1	0xFDH	1	1	1	0	0xEH
SB14	1	1	0	1	1	0xFBH	1	1	1	0	0xEH
SB15	1	0	1	1	1	0xF7H	1	1	1	0	0xEH
SB16	0	1	1	1	1	0xEFH	1	1	1	0	0xEH
-	1	1	1	1	1	Initial	1	1	1	1	Initial

Tabla 5.1: Valores del bus de direcciones y del bus de datos para cada una de las teclas del teclado matricial.

J7 y que se ha dado servicio a la rutina de tratamiento de la interrupción. La RTI deberá escribir valores en el bus de direcciones de acuerdo a la Tabla 5.1 y se identificará la tecla cuando en los cuatro bits menos significativos del bus de datos se lea el valor indicado en dicha tabla:

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

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

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

- Lectura del contenido de la posición de memoria 0x0600_00EF. Con este valor el pin 1 de J7 está a "0" (el diodo D11 está en conducción al tener el ánodo a una tensión de baja). Ahora el pin 5 tendrá un "0", la señal $EINT1 = 0$ y la salida en el bus de

datos es 0xE. Este valor coincide con el indicado en la Tabla 5.1. Luego se ha pulsado la tecla SB16.

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

5.2.4. El problema de los rebotes

Cuando se pulsa una tecla, no se origina una transición instantánea y estable (podríamos decir "limpia") de niveles de tensión entre los bornes del pulsador, como correspondería a una conmutación ideal. Esto es debido a los rebotes. En la Figura 5.3 aparece un posible diagrama temporal de un proceso de pulsación: los flancos de presión y depresión de la tecla van seguidos de una serie de rebotes.

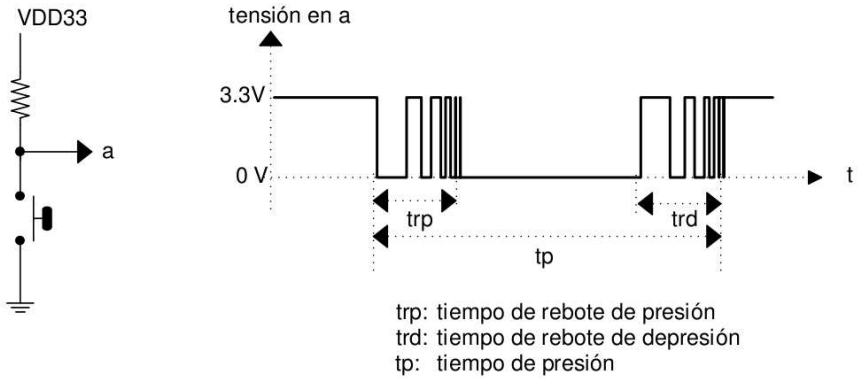

Figura 5.3: Fenómeno de rebotes al pulsar o liberar una tecla.

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

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

5.2.5. Teclado, interrupciones y pines E/S

Como podemos ver en la figura 5.2, cuando se pulsa una tecla con el teclado en estado de reposo se activa la línea EINT1. Como vimos en la práctica anterior esta línea es accesible externamente a través del puerto G, que es como se conecta al circuito externo U9B. Para que la interrupción se genere correctamente el pin 1 del puerto G debe configurarse para activar la línea EINT1 cuando se genere un flanco de bajada por dicha línea.

5.3. Múltiples fuentes de interrupción

La mayor parte de los conceptos generales relacionados con el manejo de interrupciones y la programación de los puertos de E/S ya fueron presentados en la práctica anterior. Sin embargo el tratamiento que se dio a las interrupciones en dicha práctica impedía que una ISR fuese interrumpida por otra fuente de interrupción, incluso aunque ésta fuese más prioritaria. Si repasamos lo que sucede cuando se produce una interrupción veremos cuál es el motivo. Si recordamos, cuando se produce la interrupción en el procesador, antes de saltar a la ISR, enmascara la línea IRQ (o FIQ) en el registro de estado. Así, pase lo que pase en el controlador de interrupciones, las interrupciones por esa misma línea (IRQ o FIQ) quedan inhabilitadas durante la ejecución de la ISR. Para permitir anidamiento, es decir que una nueva interrupción sea atendida mientras se ejecuta la ISR en curso, es preciso habilitar explícitamente la línea (IRQ o FIQ) escribiendo en los bits de control del registro de estado.

Recordemos también que la ISR de una interrupción se ejecuta en modo específico, modo IRQ o modo FIQ según el caso. Al hacer el salto a la ISR se salva el registro de estado en el registro de sombra del modo (SPSR_irq o SPRS_fiq). Si queremos que la ISR pueda ser interrumpida, éste registro debe ser salvado en la pila, ya que de lo contrario al saltar a la nueva ISR este registro se sobrescribiría. Lo mismo sucede con el registro de enlace, que contiene la dirección de retorno de la ISR. Posteriormente, antes de finalizar la ISR, es preciso restaurar el contenido de estos registros a partir de la información almacenada en la pila.

Además de todo esto, si queremos realizar alguna llamada a subrutina desde la ISR mientras las interrupciones están habilitadas, es necesario pasar al modo *System* previamente. De lo contrario, al realizar la llamada a la subrutina el registro de enlace sería escrito con la dirección de retorno de esta subrutina y si durante la ejecución de esta se produce una nueva interrupción, este registro sería sobreescrito con la dirección de retorno de la nueva ISR, perdiendo de este modo el punto de retorno de la subrutina. Si la subrutina se ejecuta en un modo de procesador distinto, utiliza una versión distinta del registro (LR_system) y se evita este problema.

Lamentablemente, esta labor no puede realizarse desde C y es necesario emplear lenguaje ensamblador (o al menos "inlining" de ensamblador). La secuencia completa de pasos que hay que realizar para una ISR que emplee la línea IRQ pueda ser interrumpida por otra que use la misma línea es la siguiente:

1. Guardar en la pila los registros cuyo valor se necesite en la ISR así como aquellos otros que no hayan sido previamente guardados (SPSR_irq y LR_irq incluidos).

2. Borrar la fuente de interrupción de INTPND y enmascarar las líneas adecuadas en INTMSK.
3. Pasar al modo *System* habilitando simultáneamente las interrupciones (modificar el bit I del registro CPSR).
4. En modo *System* guardar en la pila el registro de enlace (LR_sys) así como el resto de registros que puedan ser modificados. En caso de duda, es preferible guardarlos todos.
5. Llamar a la función (C o ensamblador) que realiza el cuerpo de la ISR. Si empleamos código en C, no se debe emplear el atributo `interrupt`.
6. Tras volver de esta función, es preciso restaurar los registros.
7. Después volvemos al modo IRQ deshabilitando también las interrupciones (poniendo a 1 el bit I del registro de estado).
8. Una vez en IRQ debemos restaurar los registros guardados en el primer paso.
9. Desenmascarar las líneas del controlador de interrupciones adecuadas.
10. Finalizar la ISR.

En esta práctica deberemos permitir que la ISR que se encarga de gestionar los pulsadores pueda ser interrumpida cuando el usuario pulse una tecla del teclado. Es decir, en esa situación la tecla debe ser inmediatamente reconocida y la ISR del pulsador sólo continuará cuando termine la ISR del teclado.

5.4. Watchdog timer

El Watchdog timer es un temporizador que se utiliza habitualmente para generar un reset y así reinicializar el sistema tras un fallo debido al ruido u otros errores del sistema. La figura 5.4 muestra un esquema de este temporizador. Como los temporizadores estudiados en la práctica anterior, el Watchdog timer dispone de un módulo de preescalado y un módulo divisor que permiten ajustar la frecuencia de reloj del contador. Estos valores se configuran en el registro WTCON (0x01D30000), cuya descripción viene en la tabla 5.2. Tras un reset el Watchdog timer queda configurado para generar un reset cada 128 MCLK ciclos. Para calcular el tiempo con el que se decrementa el watchdog timer podemos utilizar la siguiente expresión:

$$T_{watchdog} = \frac{1}{\frac{MCLK}{(PrescalerValue+1)(DivisionFactor)}}$$

El valor de cuenta se establece en el registro WTDAT(0x01D30004).

En la biblioteca de sistema suministrada con el proyecto el watchdog timer se utiliza como contador normal en la función Delay. La primera vez que se invoca esta rutina utiliza el watchdog para medir el tiempo transcurrido en ejecutar un par de bucles vacíos. Este tiempo se utiliza luego para reajustar el valor de los límites del bucle para que la rutina suponga un retardo fijo.

Para más información sobre el watchdog timer consultar el manual del S3C44B0X [um-].

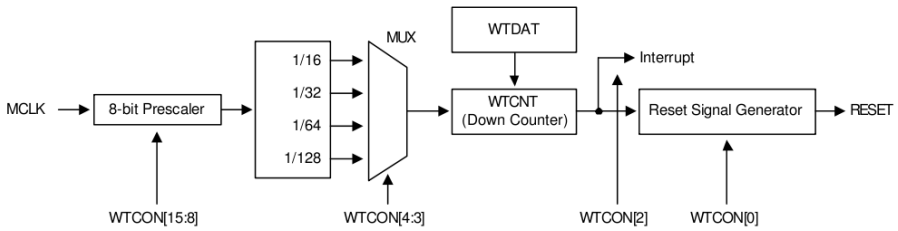

Figura 5.4: Esquema del Watchdog timer.

Tabla 5.2: Watchdog timer control Register

WTCON	Bit	Descripción
Valor de Preescalado	[15:8]	El valor de preescalado. El rango válido es: 0 a $(2^8 - 1)$
Reservado	[7:6]	Reservado. Estos dos bits deben ser cero en funcionamiento normal.
watchdog timer enable/disable	[5]	Habilita o desabilita el watchdog timer. 0 = Deshabilita el watchdog timer 1 = Habilita watchdog timer
Clock select	[4:3]	Determinan el factor de división 00: 1/16 01: 1/32 10: 1/64 11: 1/128
Interrupt enable/disable	[2]	Habilita o deshabilita la generación de interrupción. 0 = Deshabilita la generación de interrupción 1 = Habilita la generación de interrupción
Reservado	[1]	Reservado. Debe ser 0 en funcionamiento normal.
Reset enable/disable	[0]	Habilita o deshabilita la generación de reset por el watchdog timer. 1: Genera un reset con el time-out del watchdog 0: Deshabilita la generación de reset por time-out del watchdog

5.5. Desarrollo de la Práctica

En esta práctica vamos a dividir el trabajo en dos partes. En la primera, nos familiarizaremos con el uso del teclado de manera guiada. En la segunda, abordaremos el problema del anidamiento de interrupciones mediante un ejercicio que mezcla varios de los dispositivos vistos en las últimas prácticas: teclado, pulsadores, leds y display 8-segmentos.

5.5.1. Primera parte

En esta primera parte completaremos un programa que identifica las teclas pulsadas en el teclado matricial y visualiza su valor hexadecimal en el display 8-segmentos.

1. Creamos un workspace nuevo y le añadimos la carpeta *common* de manera análoga a la práctica anterior.

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

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

4. Creamos un fichero `main.c` en el que añadimos la función `Main()`. El código completo viene a continuación y su diagrama de flujo se muestra en la figura 5.5:

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

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

/*--- funciones externas ---*/
extern void keyboard_init();

/*--- Codigo de la funcion ---*/
int Main(void)
{
    sys_init();        // Inicializacion de la placa
    keyboard_init();   // Inicialización del teclado

    while(1);

    return;
}
```

5. Creamos un fichero `keyboard.c` en el que añadimos las funciones para manejo del teclado. El código parcialmente completo viene a continuación y el diagrama de flujo de la rutina de teclado se muestra en la figura 5.5:

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

/*--- Definición de macros ---*/
#define KEY_VALUE_MASK 0x0f

/*--- Variables globales ---*/
volatile UCHAR *keyboard_base = (UCHAR *)0x06000000;

/*--- Funciones externas ---*/
void D8Led_symbol(int value);

/*--- Declaracion de funciones ---*/
void keyboard_init();
void KeyboardInt(void) __attribute__ ((interrupt ("IRQ")));

/*--- Codigo de las funciones ---*/
void keyboard_init()
{
    /* Configurar controlador de interrupciones */

    /* Habilitar linea EINT1 */

    /* Establece la rutina de servicio para EINT1 */

    /* Configurar el puerto G (si no lo estuviese ya) */

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

void KeyboardInt(void)
{
    int value;

    /* Esperar trp mediante la funcion Delay() */

    /* Identificar la tecla */
    value = key_read();

    /* Si la tecla se ha identificado, visualizarla */
    if(value > -1)
       {
            D8Led_symbol(value);
       }
```

```
    /* Esperar a se libere la tecla */
    // consultar bit 1 del registro de datos del puerto G

   /* Esperar trd mediante la funcion Delay() */

   /* Borrar interrupciones */
}

inline int key_read()
{
   int value;
   char temp;

   /* Identificar la tecla mediante ''scanning''  */
   temp = *(keyboard_base+0xfd);

   /* Si la identificación falla la función debe devolver -1 */
}
```

La rutina de identificación de tecla, `int key_read()` devolverá el valor de la tecla pulsada. La identificación seguirá el procedimiento de "*scanning*":

- Se lee, en el bus de datos, el contenido de la dirección asociada a las teclas de la primera fila. Esta dirección se obtiene como la suma de la dirección base asociada al teclado ,`0x0600_0000`, más la dirección asociada a esa primera fila, `0xFD` (véase Tabla 5.1).

  ```
  temp = *(keyboard_base + 0xfd);
  ```

- Se comprueba si alguno de los cuatro bits menos significativos está a "0". En caso afirmativo, ya ha sido identificada la tecla pulsada.
- En caso contrario el anterior procedimiento se repite para cada nueva fila.

6. Para visualizar un código hexadecimal en el display 8-segmentos es necesario realizar las siguientes modificaciones al código de la práctica anterior:

   - Añadir el mapa de bits de los digitos hexadecimales:

     ```
     #define DIGIT_F (SEGMENT_A|SEGMENT_G|SEGMENT_E|SEGMENT_F)
     #define DIGIT_E (SEGMENT_A|SEGMENT_G|SEGMENT_E|SEGMENT_F|SEGMENT_D)
     #define DIGIT_D (SEGMENT_B|SEGMENT_C|SEGMENT_D|SEGMENT_F|SEGMENT_E)
     #define DIGIT_C (SEGMENT_A|SEGMENT_D|SEGMENT_E|SEGMENT_G)
     #define DIGIT_B (SEGMENT_C|SEGMENT_D|SEGMENT_F|SEGMENT_E|SEGMENT_G)
     #define DIGIT_A (SEGMENT_A|SEGMENT_B|SEGMENT_C|SEGMENT_F|SEGMENT_E|SEGMENT_G)
     #define DIGIT_9 (SEGMENT_A|SEGMENT_B|SEGMENT_C|SEGMENT_F|SEGMENT_G)
     ...
     #define DIGIT_0 (SEGMENT_A|SEGMENT_B|SEGMENT_C|SEGMENT_D|SEGMENT_E|SEGMENT_G)
     ```

   - Modificar la tabla de símbolos:

```
int Symbol[] = { DIGIT_0, DIGIT_1, DIGIT_2, DIGIT_3, DIGIT_4, DIGIT_5,
                 DIGIT_6, DIGIT_7, DIGIT_8, DIGIT_9, DIGIT_A, DIGIT_B,
                 DIGIT_C, DIGIT_D, DIGIT_E, DIGIT_F};
```

- Modificar la rutina `D8Led_symbol()` de acuerdo con el nuevo tamaño de la tabla de símbolos.

7. Configurar el proyecto como en la práctica anterior.
8. Completar el código, compilar el proyecto y subirlo a la placa. Comprobar que el programa funciona correctamente.
9. **Ejercicio:** Implementar la rutina `key_read()` en ensamblador.
10. Examinar la función `Delay()` y contestar a las siguientes preguntas:
    - ¿Cuál es intervalo de tiempo (en μs) con el se configura inicialmente el Watchdog?
    - ¿Cuál es el valor al que se ajusta `DelayLoopCount`?

5.5.2. Segunda parte

Partiendo del código que acabamos de elaborar y sirviéndose de los códigos desarrollados en la práctica anterior, es necesario implementar un programa con el siguiente comportamiento:

- Al presionar un pulsador se iluminará el LED correspondiente (SB2: LED1 y SB3:LED2) durante 2 segundos y después se volverá a apagar. Durante este intervalo de tiempo, si se presiona cualquiera de los pulsadores la petición de interrupción será ignorada.
- Al pulsar una tecla del teclado se visualizará su valor hexadecimal en el display 8-segmentos y se apagarán ambos LEDs. La pulsación de tecla debe poder interrumpir a la rutina de tratamiento de interrupción de los pulsadores.

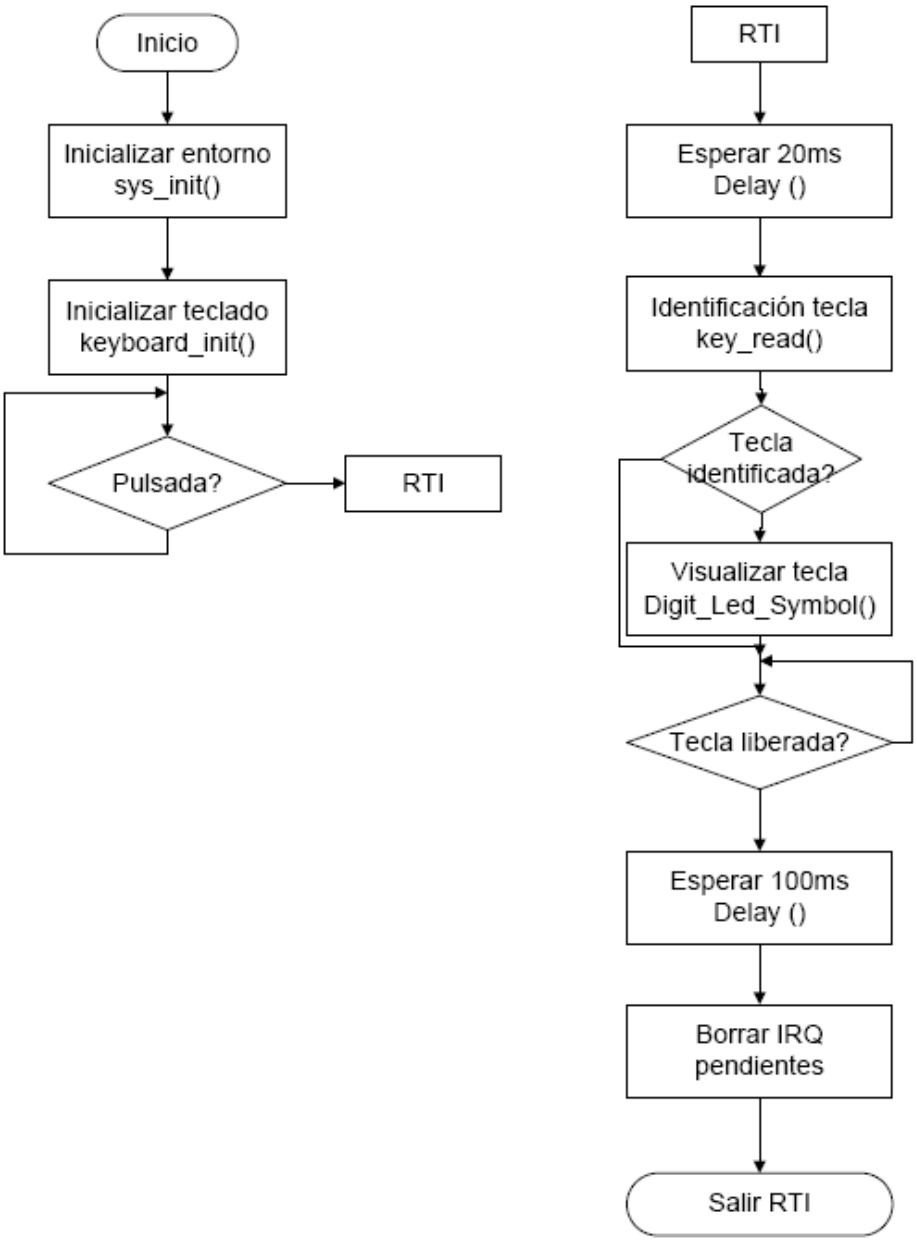

Figura 5.5: Diagrama de flujo de la rutina de identificación de tecla.

Bibliografía

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