Documentación de prácticas:
Circuitos i Sistemas Electrónicos IV

Pedro Bibiloni Serrano
Adrià Recasens Continente

Introducción

En este documento recogemos la realización práctica y experimental de la asignatura de CiSE IV. Ésta se basa en la programación de un microcontrolador, el LPC2292 de Philips, en una placa LPCEB2000. Utilizaremos el entorno de desarrollo Eclipse, programaremos en lenguaje **C** y compilaremos con el compilador cruzado *arm-elf-gcc*. En total presentamos las memorias de las 6 prácticas siguientes:

- Práctica 3: Depuración y programación con el entorno Eclipse.
- Práctica 4: Control de los displays de 7 segmentos.
- Práctica 5: Timer0, Timer1 y Vectored Interrupt Controller.
- Práctica 6: Ciclos de bus externos.
- Práctica 7: Teclado.
- Práctica 8: Altavoz.

Practica 3: Depuración y programación con el entorno Eclipse

En esta práctica, principalmente, hemos abierto un proyecto existente con el entorno Eclipse, y hemos configurado este entorno para programar en el microcontrolador con todas las herramientas necesarias (*Eclipse*, *OnChipDebug*, Placa de pruebas). También hemos compilado correctamente con este montaje el archivo de prueba. Gracias a esta práctica hemos podido trabajar por primera vez con un sistema embebido en un entorno de desarrollo.

En el primer intento, tuvimos ciertos problemas con el conexionado: el sistema no funcionaba porque nos fallaba la comunicación entre la placa de pruebas y el *OnChipDebug*. El fallo estaba en que aparecían errores en la pantalla MS-DOS del *OnChipDebug*, probablemente debidos a problemas de conexionado. Esta práctica nos ha servido, además, como referencia para futuras prácticas ya que sintetiza todos los pasos que tenemos que hacer para compilar y comprobar nuestros propios progamas.

Pregunta 1: Encontrar el fichero donde se indica a partir de que dirección se cargará el programa.

Respuesta 1: El fichero en cuestión es: *olimex_lpce2294_ram.ld*. Concretamente, la indicación se encuentra en las siguientes sentencias:

```
1 MEMORY
2 {
3 ram : org = 0x81000000, len = 16k
4 }
```

Practica 4: Control de los displays de 7 segmentos

En esta práctica hemos aprendido a utilizar los registros más básicos del microcontrolador, es decir, los encargados de la entrada i la salida. Estos registros son, concretamente, *PINSEL∗* para configurar los puertos como GPIO o como puertos reservados, *IODIR* para configurar los puertos GPIO como entradas o salidas, y *IOCLR∗, IOSET∗* y *IOPIN∗* para escribir o leer los pines de los diferentes puertos.

En esta práctica, hemos creado el siguiente código en **C** para manejar los displays, el cual enseña por pantalla los numeros del 0 al 7, uno en cada display. Para ello, no utilizamos ningún timer, sino una función que se ocupa de 'retardar' un poco para darle tiempo al circuito encargado de manejar los displays y que pueda desempeñar su función (linea 33 del código).

Los principales errores que hemos tenido en esta práctica han sido no identificar bien los pines, y no activar bien los pines (no tener en cuenta que hay algunos pines que se activan a nivel bajo). Esto nos ha causado que se enciendan displays que no tocan, o que se enciendan LEDs que no deberían hacerlo dentro de un mismo display. También hemos notado que, si la función que se encargaba de retardar era muy rápida (es decir, prácticamente no retardaba), se encendían débilmente casi todos los LEDs de los displays. Hemos atribuido este fallo a los retardos en los buffers que seleccionan el display activo, ya que en cada display se podía intuir el número que iba en el display siguiente.

```
1 #define __MAIN_C__
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <LPC_Base.h>
5 #include <LPC_SysControl.h>
6
7 void initdisplay(void)
8 {
9 *PINSEL0&=0x000FFFFF; // Pins P0.10-15 son GPIO
10 *PINSEL1&=0xF3FFFF00; // Pins P0.16-19 i P0.29 son GPIO
11 *IODIR0|=P10+P11+P12+P13+P14+P15+P16+P17+P18+P19+P29; // Pins P0.10:19 i P0
       .29 son sortides
12 *IOCLR0=P29; // Posem a 0 bit P0.29.
```

```
13 }
14
15
16 int main (void)
17 {
18 int k,i; //Definimos variables para los bucles
19 int v[8]={P17,P16,P15,P14,P13,P12,P11,P10}; //Definimos un vector donde v[i]
     representa el puerto del display i.
20 int numero[10]={P10+P11+P13+P15+P16+P17,P13+P15,P10+P11+P14+P15+P17,P11+P13+
    P14+P15+P17,P13+P14+P15+P16,P11+P13+P14+P16+P17,P10+P11+P13+P14+P16+P17,
    P13+P15+P17,P10+P11+P13+P14+P15+P16+P17,P13+P14+P15+P16+P17}; //
    Codificamos los 10 numeros en codigo del seven-segments.
21 int posicio[8]={numero[0],numero[1],numero[2],numero[3],numero[4],numero[5],
    numero[6],numero[7]}; // En este vector hay que numero aparece en cada
    display
22 initdisplay(); //Iniciamos los puertos de los displays
23 i=0;
24 while(1){
25   *IOCLR0=P18+P19+P29; //Ponemos un 0 a los dos enables i al buffer para
       activar-lo
26   *IOCLR0=P10+P11+P12+P13+P14+P15+P16+P17; //Ponemos todos los puertos a 0
27   *IOSET0=posicio[i]; //Activamos los que tocan para poner la figura poisico
       [i]
28   *IOSET0=P18; //Activamos el primer latch
29   *IOCLR0=P18; //Cuando ya hemos cargado los valores, desactivamos el primer
       latch
30   *IOSET0=P10+P11+P12+P13+P14+P15+P16+P17; //Podemos todos los puertos a uno
31   *IOCLR0=v[i]; //Activamos el display que queremos
32   *IOSET0=P19; //Activamos el segundo Latch
33   for(k=0;k<250;k++); //Dejamos tiempo de espera para poder cargar bien el
       valor
34   *IOCLR0=P19; //Desactivamos el latch que escoge que seven-segments esta
       abierto
35   i=(i+1)%8;
36   }
37
38 }
```

sevensegments.c

Practica 5: Timer0, Timer1 y Vectored Interrupt Controller

En la práctica dedicada a estudiar el funcionamiento y la configuración de los timers del microcontrolador, hemos mejorado el programa anterior (sevensegments.c). Con ello, hemos obtenido otro programa que utiliza los dos timers: uno para multiplexar los displays de 7 segmentos, y otro para hacer rotar los numeros hacia la derecha (del 0 al 9, van apareciendo y desapareciendo a través de los 8 displays).

En esta práctica hemos aprendido a utilizar los temporizadores mediante:

- Los registros encargados de los temporizadores. A saber, *TIMER∗_TCR, TIMER∗_PR, TIMER∗_MR∗, TIMER∗_MCR*. También hemos utilizado el registro de control de

interrupción de los timers, *TIMER∗_IR*

- Haciendo que el temporizador tenga la interrupción vectorizada, es decir, que reproduzca una función programada por nosotros cada vez que se encuentre con una interrupción. Esta característica la hemos configurado mediante los registros *VICIntEnable, VICIntSelect, VICVectCntl∗, VICVectAddr∗*.

Estudio previo 1: ¿Cual es la frecuencia de interrupción del *Timer0*? ¿Cuánto tiempo tarda en ejecutase *retard(20)* ?

Respuesta: Dado que el reloj está programado a 10MHz, y que tenemos que tanto el Preescaler como el Match0 (el valor limite del timer0) dividen a la frecuencia, tenemos:

$$f_{Timer0} = \frac{f_{clk}}{Preescaler \cdot Match0}, \qquad t = \frac{n}{f_{Timer0}}$$

Donde n es el numero de veces que se repite. Los datos son:

$$f_{clk} = 10 \cdot 10^6, \qquad Match0 = 240, \qquad Preescaler = 47, \qquad n = 20$$

Con lo que obtenemos:

$$f_{Timer0} = 886,52Hz \Rightarrow t = \frac{n}{f_{Timer0}} = 22,56ms$$

En el laboratorio, para configurar $f_{timer1} = 100Hz$, hemos obtenido los valores:

$$Preescaler1 = 0xFFFFFFFF, \qquad Match1 = 0x8ACF2305$$

Más tarde, para que fuesen desplazándose los números, hemos impuesto $f_{timer1} = 1Hz$ (para que se desplace una cifra cada segundo), obtiendo los valores:

$$Preescaler1 = 0x0FFFFF, \qquad Match1 = 0x0A$$

Las principales dificultades con las que nos hemos encontrado han sido:

- El tiempo que tardaban los timers en llegar a la interrupción no cuadraban. Lo hemos arreglado revisando los registros de Match∗ y Preescaler∗
- El timer1 no llegaba a entrar en su rutina de servicio de interrupción, y parecía que no llegaba a solicitar la interrupción. El problema estaba en que no habíamos indicado a la interrupción vectorizada en qué rutina tenía que entrar.

Finalmente, hemos obtenido el siguiente código:

```
1 #define __MAIN_C__
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <LPC_Base.h>
6 #include <LPC_SysControl.h>
```

```
7  extern void TIMER1_VectISR(void);
8  extern void TIMER0_VectISR(void);
9  unsigned int tics=0,tics1=0, j=0;
10
11 void initimer0(void)
12 {
13 *TIMER0_TCR = B1; // Timer0 Reset
14 *TIMER0_PR = 0x02F; // Prescaler
15 *TIMER0_MR0 = 0xF0; // Match0
16 *TIMER0_MCR = B0 + B1;
17 *TIMER0_TCR = B0; // Timer0 ON
18 *VICIntSelect&=~B4; // Timer0 Int. es IRQ
19 *VICVectAddr0 = (int)TIMER0_VectISR;
20 *VICVectCntl0 = B5 + 0x4;
21 *VICIntEnable|=B4;
22 }
23
24
25 void TIMER0_ISR(void)
26 {
27 tics++;
28 *TIMER0_IR |= 0x1; //anulem peticio int.
29 *VICVectAddr = 0x00; //EOI del VIC
30 //(end of int.)
31 }
32
33
34 void initimer1(void)
35 {
36 *TIMER1_TCR = B1; // Timer0 Reset
37 *TIMER1_PR = 0x0FFFFF; // Prescaler
38 *TIMER1_MR0 = 0X0A; // Match0
39 *TIMER1_MCR = B0 + B1;
40 *TIMER1_TCR = B0; // Timer0 ON
41 *VICIntSelect&=~B5; // Timer0 Int. es IRQ
42 *VICVectAddr1 = (int)TIMER1_VectISR;
43 *VICVectCntl1 = B5 + 0x5;
44 *VICIntEnable|=B5;
45 }
46
47 void TIMER1_ISR(void){
48 j = (j+1)%10;
49 *TIMER1_IR |= 0x1; //anulem peticio int.
50 *VICVectAddr = 0x00; //EOI
51 }
52
53
54 void initdisplay(void)
55 {
56 *PINSEL0&=0x000FFFFF; // Pins P0.10-15 son GPIO
57 *PINSEL1&=0xF3FFFF00; // Pins P0.16-19 i P0.29 son GPIO
58 *IODIR0|=P10+P11+P12+P13+P14+P15+P16+P17+P18+P19+P29; // Pins P0.10:19 i P0
       .29 son sortides
59 *IOCLR0=P29; // Posem a 0 bit P0.29.
```

```
60 }
61
62 void retard(int n)
63 {
64 tics=0; //Inicialitzar la variable a 0
65 while (tics<n); //Esperar n tics fins a parar
66 }
67
68
69 int main (void)
70 {
71
72 int k,i=0; //Definimos variables para los bucles
73 int v[8]={P17,P16,P15,P14,P13,P12,P11,P10};//v[i] es el puerto del display i
    .
74 int numero[10]={P10+P11+P13+P15+P16+P17,P13+P15,P10+P11+P14+P15+P17,P11+P13+
    P14+P15+P17,P13+P14+P15+P16,P11+P13+P14+P16+P17,P10+P11+P13+P14+P16+P17,
    P13+P15+P17,P10+P11+P13+P14+P15+P16+P17,P13+P14+P15+P16+P17};//
    Codificamos los 10 numeros en codigo del seven-segments.
75 int posicio[8]={numero[0],numero[1],numero[2],numero[3],numero[4],numero[5],
    numero[6],numero[7]}; // En este vector hay que numero aparece en cada
    display
76 initdisplay(); //Iniciamos los puertos de los displays
77 initimer0(); //Iniciamos y configuramos el Timer0
78 initimer1(); //Iniciamos y configuramos el Timer1
79 *IOCLR0=P18+P19;
80    while(1){
81       *IOCLR0=P18+P19+P29; //Ponemos un 0 a los dos enables i al buffer para
            activar-lo
82       *IOCLR0=P10+P11+P12+P13+P14+P15+P16+P17; //Ponemos todos los puertos a
            0
83       *IOSET0=numero[(i+j)%10]; //Activamos los que tocan para poner la
           figura poisico[i], notese que depende del timer1.
84       *IOSET0=P18; //Activamos el primer latch
85       *IOCLR0=P18; //Cuando ya hemos cargado los valores, desactivamos el
           primer latch
86       *IOSET0=P10+P11+P12+P13+P14+P15+P16+P17; //Podemos todos los puertos a
            uno
87       *IOCLR0=v[i]; //Activamos el display que queremos
88       *IOSET0=P19; //Activamos el segundo Latch
89       retard(1); //Dejamos tiempo de espera para poder cargar bien el valor
           usando el Timer0
90       *IOCLR0=P19; //Desactivamos el latch que escoge que seven-segments
           esta abierto
91       i=(i+1)%8; //Incremento modulo 8.
92    }
93
94 }
```

timers.c

Tambien hemos variado interrupt.s para vectorizar la interrupción de ambos temporizadores, obteniendo el siguiente código:

```
1  .macro IRQHandle in_handle,out_handle
2          .extern \in_handle
3          .global \out_handle
4  \out_handle:
5          stmdb     sp!, {r0-r11, ip, lr}
6          ldr       r0,  =\in_handle
7          mov       lr,  pc
8          bx        r0
9          ldmia     sp!, {r0-r11, ip, lr}
10         subs      pc,  r14, #4
11 .endm
12 IRQHandle TIMER0_ISR,TIMER0_VectISR
13 IRQHandle TIMER1_ISR,TIMER1_VectISR
14
15 # FIQ_Handler code
16 .macro FIQHandle in_handle,out_handle
17         .extern \in_handle
18         .global \out_handle
19 \out_handle:
20                 STMFD     SP!, {R0-R3, LR}
21                 BL        \in_handle
22                 LDMFD     SP!, {R0-R3, LR}
23                 SUBS       PC, LR, #4
24 .endm
```

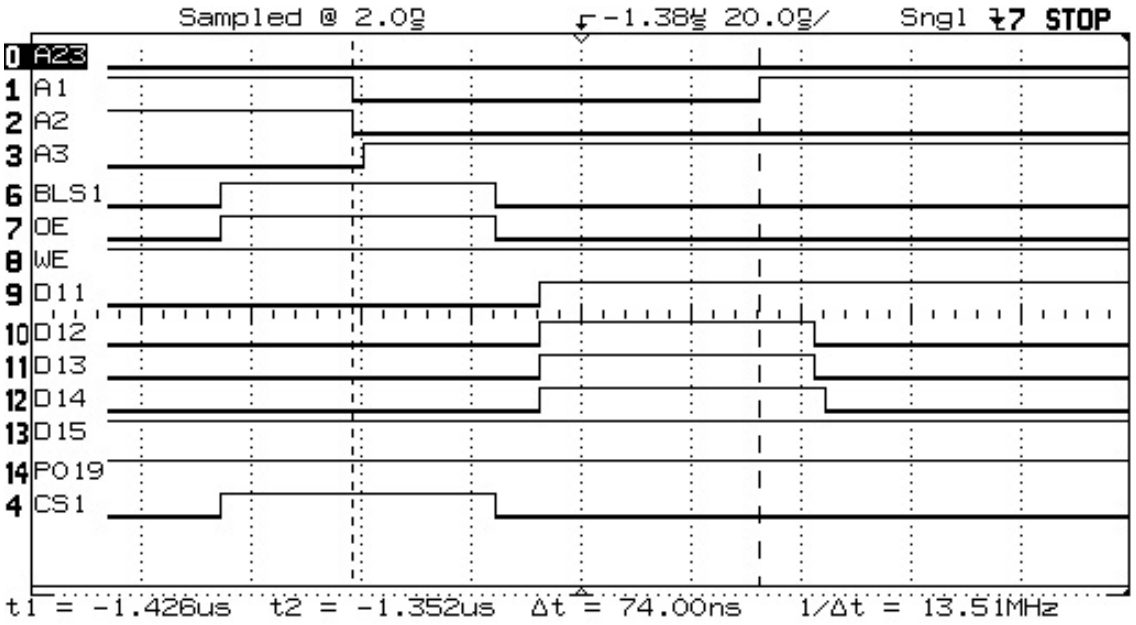
Interrupt.s

Practica 6: Ciclos de bus externos

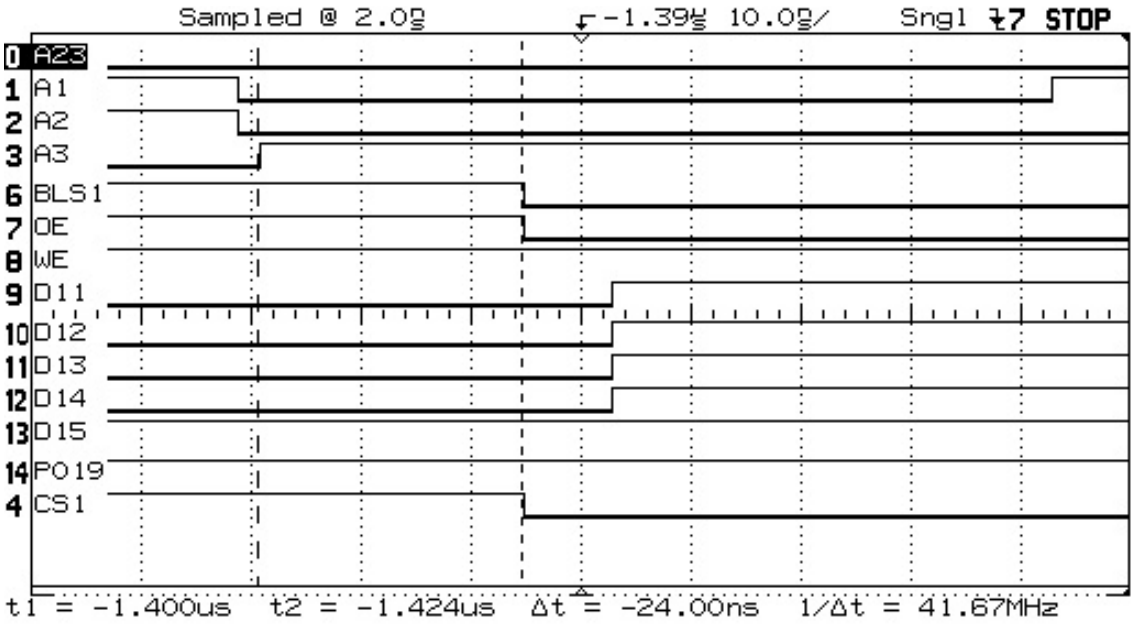
En ésta práctica nuestro objectivo principal es visualizar las formas de las señales del microprocesador en un ciclo de bus. También queremos hacer la medida práctica de los tiempos de acceso de lectura desde el micro a la memoria RAM. Para eso, después de hacer el conexionado y aprender el funcionamiento del analizador, encendemos el mismo y damos nombre a las señales para una mejor comprensión de las capturas obtenidas.

Tiempos de acceso a la memoria RAM

Primero de todo, capturamos un ciclo entero y medimos el tiempo completo. Vemos en la imagen que el período de ciclo de bus es: $T_{bus} = 74ns$.

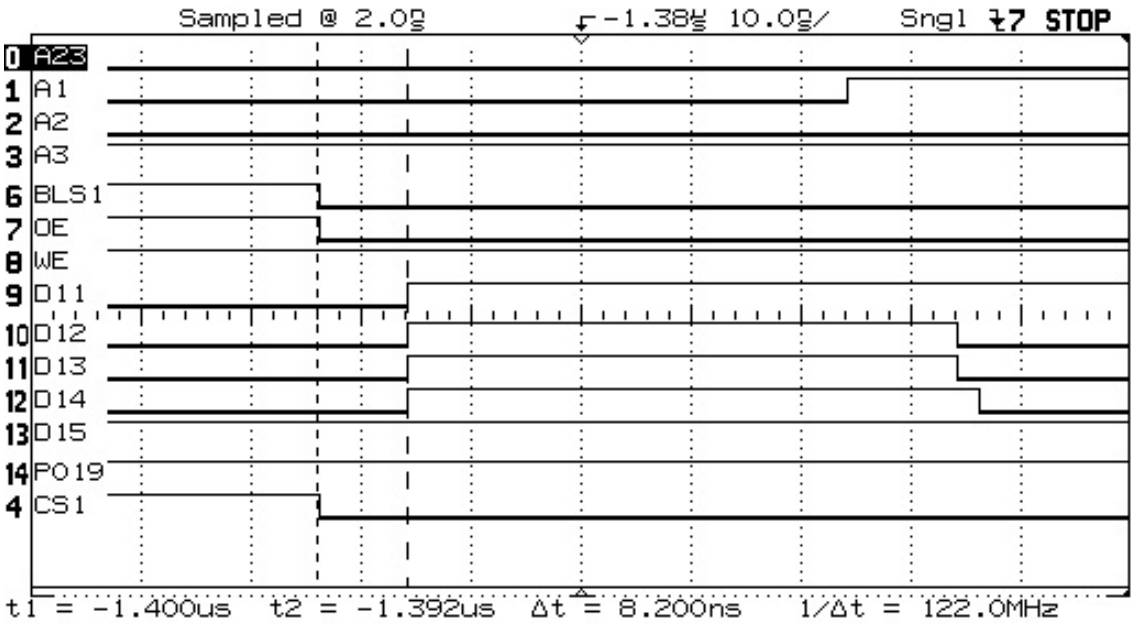


Posteriormente, medimos el tiempo que tarda el micro desde que cambia la dirección de memoria hasta que activa el *Output Enable* para dar la señal a la memoria. En nuestro caso, tenemos: $t_{ADDOE} = 24ns$.



Són tiempos también determinantes los que hay entre la activación de la señal de Chip Select y la respuesta de la memoria, pero dado que la señal de CS i la de OE se activan de manera conjunta (o tan parecida que el analizador no precisa diferencia) mediremos los tiempos con la referencia del OE. Falta pues el tiempo de acceso, es decir, el tiempo que

va entre la activación del OE por parte del micro y la respuesta en forma de datos en el bus por parte de la memoria. En nuestro cas, tenemos $t_{OEDB} = 8.2ns$.



Por lo tanto, con una simple suma podemos obtener el tiempo entre la activación de la memoria y el del dato: $t_{ABDB} = t_{ADDOE} + t_{OEDB} = 32.2ns$.

Comparación con los tiempos de datasheet

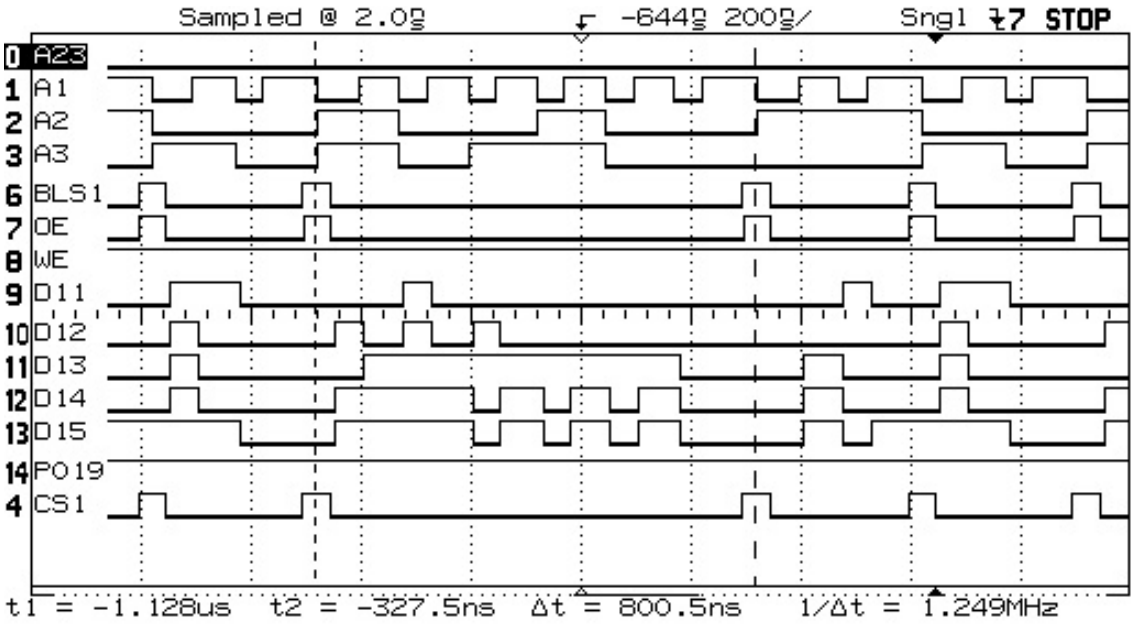
Posteriormente, establecemos la comparación entre los tiempos que nos da el fabricante y los reales. Obtenemos:

Taula 1: Comparación de tiempos

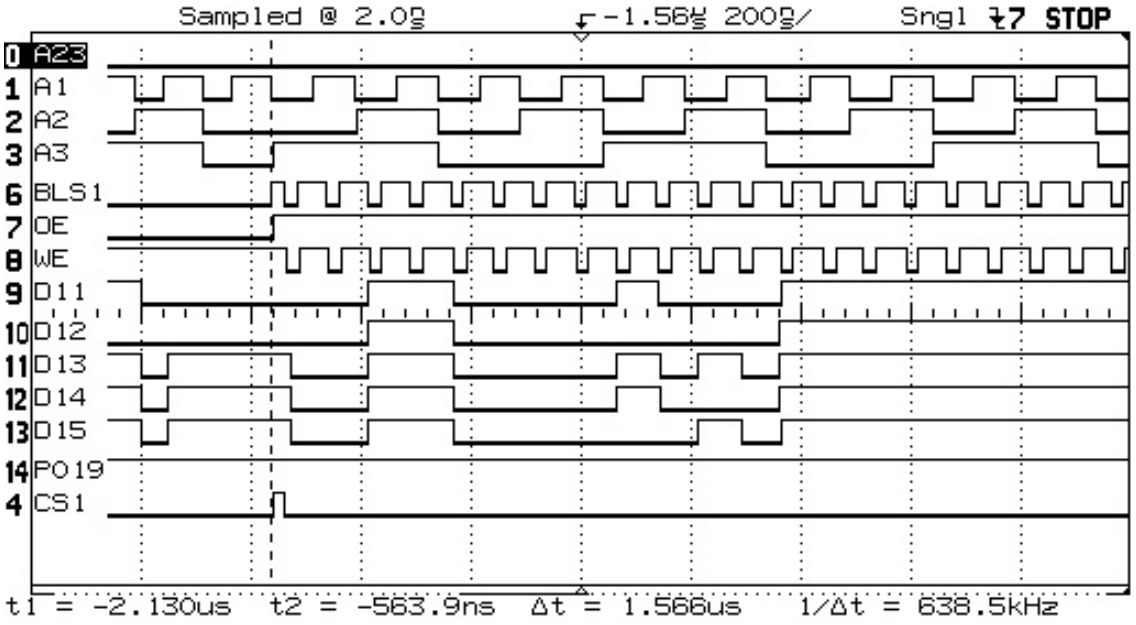
Tiempo	Tiempo Fabricante (ns)	Tiempo experimental (ns)
t_{ABDB}	[27, 42]	32.2
t_{OEDB}	12	8.2
t_{ADDOE}	[15,30]	24

Ciclos Burst

Los ciclos Burst se caracterizan por ser lecturas o escrituras a memoria en posiciones consecutivas. De ésta manera la memoria va avanzando unidad a unidad, y la lectura (o escritura) se puede hacer de modo más eficiente. Vemos un ciclo Burst de lectura:



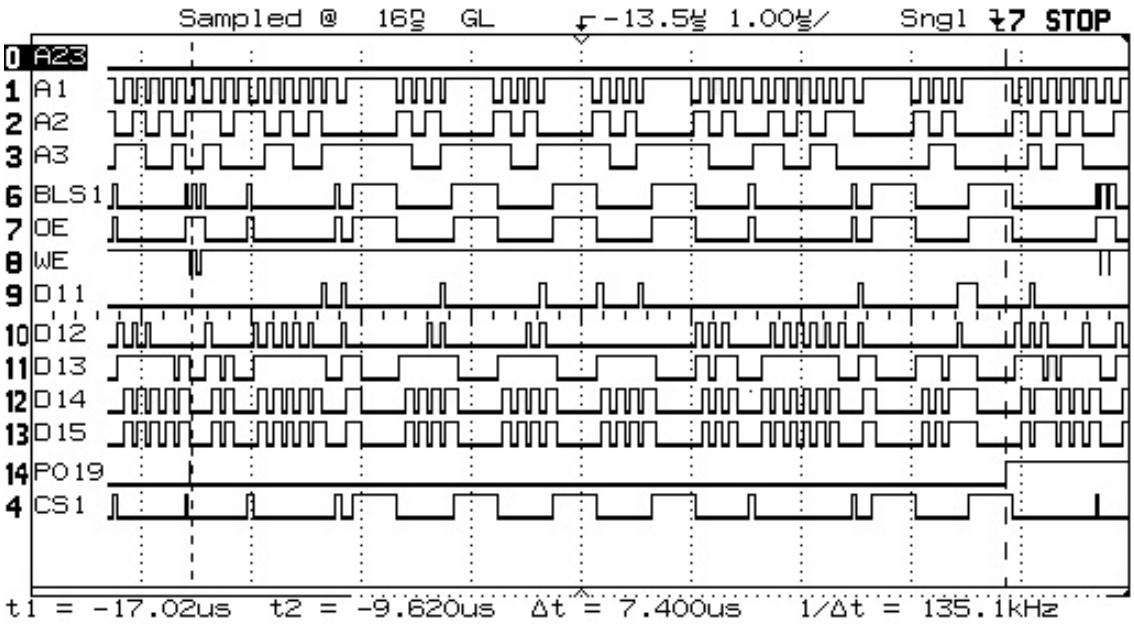
Ahora un ciclo Burst de escritura:



Observamos pues que las diferencias entre uno y otro ciclo radican en la forma de las señales OE, WE y BLS1. En el caso de la escritura en ráfaga, el WE se activa y desactiva en cada escritura porque el microcontrolador escribe el dato en el momento en que el WE se desactiva (y, por tanto, por cada escritura, necesitamos que se desactive una vez).

Frecuencia TIMER1

Mostrando el pin P19, observamos la frecuencia del TIMER1. Vemos, en la siguiente imagen que entre dos interrupciones $T_{TIMER1} = 7.4\mu s$ i por lo tanto que la $f_{TIMER1} = 135.1kHz$.

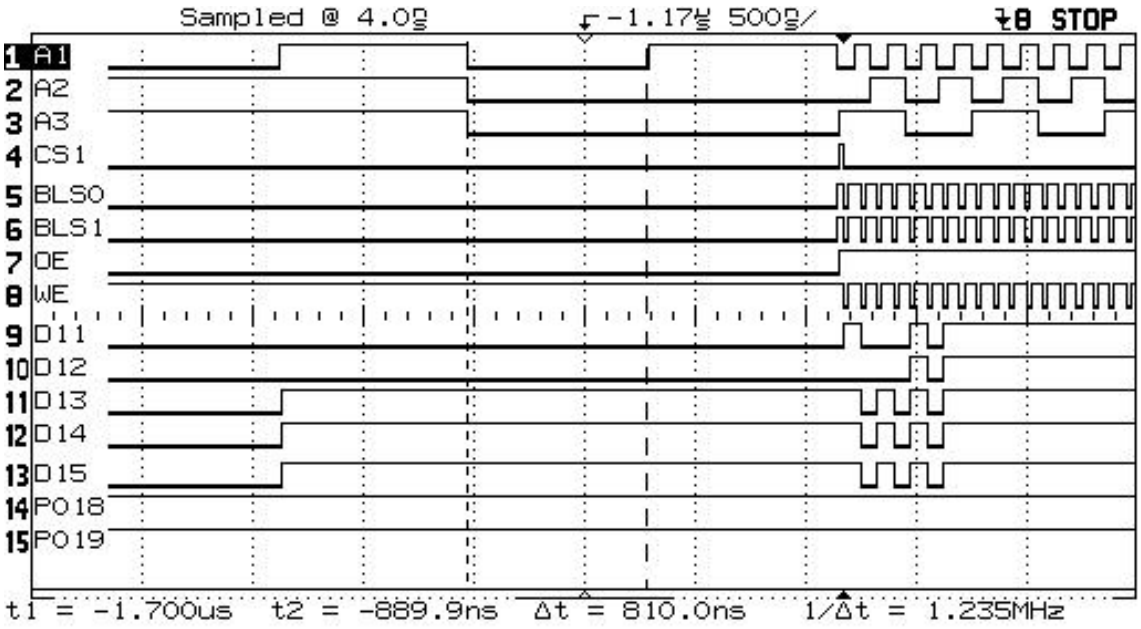


Ciclos de espera

En ésta sección, ya medido los tiempos de acceso hemos programado BCFGn para que nos diera 31 tiempos de espera en letura (pero no en escritura). Concretamente, el código utilizado ha sido:

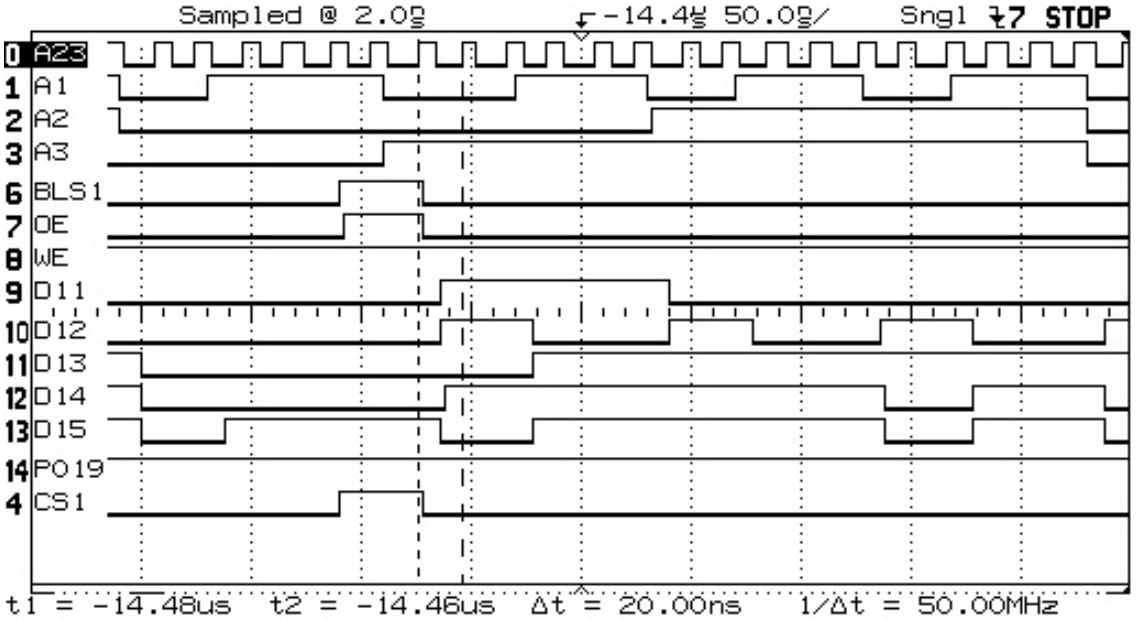
```
1 *BCFG0|=0x3F0;
2 *BCFG1|=0x3F0;
3 *BCFG2|=0x3F0;
4 *BCFG3|=0x3F0;
```

Podemos comprovar que efectivamente cambia la longitud del ciclo de bus en la siguiente imagen:

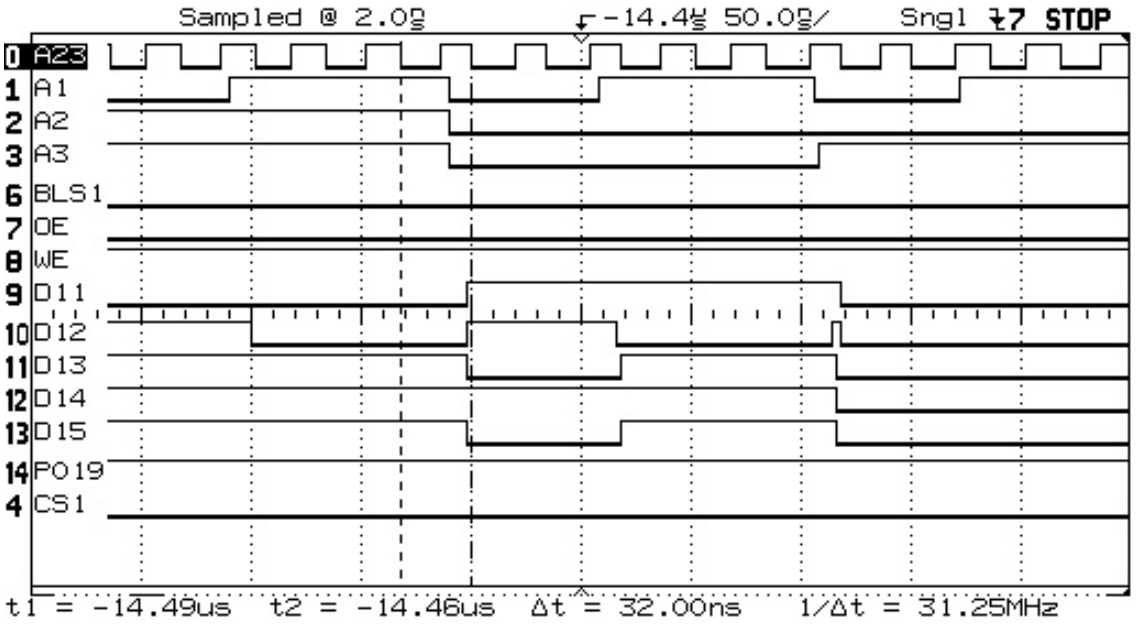


Cambio de la frecuencia de clock

Por último, utilizamos el código que se nos ofrecía en la práctica para cambiar la frecuencia de reloj y poder observar el efecto en el analizador. Primero comprobamos la frecuencia inicialmente, obteniendo: $f_{clock} = 50MHz$, tal como esperábamos.



Luego programamos el micro para que trabaje en una frecuencia de $f_{clock} = 31.25MHz$ y comprovamos en el analizador si se cumple el requisito deseado.



Con lo que se observa que hemos conseguido rebajar la frecuencia del micro del mismo hasta los $31.25Mhz$ mediante métodos de software.

Conclusión

En ésta práctica hemos podido practicar lo estudiado en teoria durante todo el curso. Es interesante comprovar la aplicación práctica de nustro estudio, y también es beneficioso para el estudiante poderse enfrentar a éste tipo de trabajos donde se le requiere un tipo diferente de habilidades que en condiciones normales. En nuestro caso tuvimos algun problema a la hora de conectar todas las señales como tocaban y posteriormente fue también dificultuso encontrar un ciclo BURST de escritura. Por suerte, con paciencia conseguimos obtener una captura de cada tipo a modo de ejemplo.

Practica 7: Teclado

Durante esta práctica aprenderemos a detectar si alguna tecla del teclado de la placa de pruebas ha sido pulsada. Usaremos para tal fin una función aportada por el enunciado de la práctica en la que se nos da un método para detectar cuando una tecla está pulsada con un circuito interno. Para el buen funcionamiento del mismo habrá que poner los pines P20=P21=P22=P23=0 para que al pulsar una tecla pueda reaccionar el puerto de salida pertinente. En cuanto se detecte una tecla pulsada, saltará una interrupción y pasaremos a detectar por la vía de un barrido qué tecla se ha pulsado.

En la práctica también se propone un método alternativo (que no ha sido el que hemos implementado). Éste consiste en hacer dos barridos: uno para los 4 pines de entrada, y otro

para los 4 pines de salida.[1] En éste caso, consumiríamos más recursos del microcontrolador, ya que en el propio programa principal deberemos implementar uno de los barridos, pero simplifica el diseño, ya que no deberíamos configurar algunos puertos con interrupción por flancos de subida y bajada.

El programa en cuestión utiliza el *Timer0* para multiplexar los displays. El funcionamiento es simple: cuando pulsamos una tecla, ésta aparece en los displays de 7 segmentos. Cuando dejamos de pulsarla, desaparece y todos los displays se desactivan.

En la realización práctica, necesitamos revisar los registros de interrupción de las entradas, ya que en un primer momento los cargábamos con un valor erróneo y, como resultado, no llegaba a entrar nunca a la interrupción y no se llegaba a ver ningún número.

El código en cuestión es:

```
1  #define __MAIN_C__
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <LPC_Base.h>
6  #include <LPC_SysControl.h>
7  extern void TIMER1_VectISR(void);
8  extern void TIMER0_VectISR(void);
9  unsigned int tics=0, j=-1, t=0,x;
10
11
12 extern void EINT3_VectISR(void);
13 unsigned int polar=0;
14
15 void P030_EINT3_ISR(void) {
16    int k;
17    j=-1;
18    *VICIntEnClr|=B17; //Deshabilitem EINT3
19    if (!polar)              //Canvi de polaritat EINT3
20       *EXTPOLAR=polar=B3;
21    else *EXTPOLAR=polar=0;
22    for(k=0; k<4; k++) {
23       *IOSET1 = P20+P21+P22+P23;
24       *IOCLR1 = 1<<(20+k);
25       x=(~*IOPIN1>>16)&0xF; //Obtenim les coordenades x (Nivell baix actiu)
26       switch (x) {
27       case 1:
28          j = 4*k;
29          break;
30       case 2:
31          j = 4*k+1;
32          break;
33       case 4:
34          j = 4*k+2;
35          break;
36       case 8:
```

[1] Hay que recordar que, para detectar una tecla pulsada, deben estar activos un pin de entrada y un pin de salida.

```
37            j = 4*k+3;
38            break;
39        }
40    }
41    *IOCLR1 = P20+P21+P22+P23;
42    *VICIntEnClr|=B17; //Deshabilitem EINT3
43    *EXTINT|=P3;  //Anulem les int.pendents
44    *VICVectAddr = 0x00; //EOI
45    *VICIntEnable|=B17; //Habilitem EINT3
46 }
47
48 void initeclat(void) {
49    *PINSEL2&=~(P3); // Pins P1.25:16 son GPIO
50    *IODIR1|=P20+P21+P22+P23; // Pins P1.20:23 son sortides.
51    *IODIR1&=~(P16+P17+P18+P19); // Pins P1.16:19 son entrades
52    *IOCLR1=P20+P21+P22+P23;// Posem a 0 els bits P1.20-P1.23
53    *PINSEL1&=~B28; // Pin P0.30 es EINT3
54    *PINSEL1|=B29;
55    *EXTMODE=B3; // EINT3 es edge sensitive.
56    *EXTPOLAR=0; // EINT3 es falling-edge
57    *EXTINT|=P3;// Anulem qualsevol int. pendent
58    *VICIntSelect&=~B17; // EINT3 es IRQ (i no FIQ)
59    *VICVectAddr2 = (int)EINT3_VectISR;
60    *VICVectCntl2 = B5 + 17; // Fem servir VIC Channel 2
61    *VICIntEnable|=B17;
62 }
63
64 void initimer0(void)
65 {
66 *TIMER0_TCR = B1; // Timer0 Reset
67 *TIMER0_PR = 0x02; // Prescaler
68 *TIMER0_MR0 = 0xF0; // Match0
69 *TIMER0_MCR = B0 + B1;
70 *TIMER0_TCR = B0; // Timer0 ON
71 *VICIntSelect&=~B4; // Timer0 Int. es IRQ
72 *VICVectAddr0 = (int)TIMER0_VectISR;
73 *VICVectCntl0 = B5 + 0x4;
74 *VICIntEnable|=B4;
75 }
76
77
78 void TIMER0_ISR(void)
79 {
80 tics++;
81 *TIMER0_IR |= 0x1; //anulem peticio int.
82 *VICVectAddr = 0x00; //EOI del VIC
83 }
84
85
86 void initimer1(void)
87 {
88 *TIMER1_TCR = B1; // Timer0 Reset
89 *TIMER1_PR = B7; // Prescaler
90 *TIMER1_MR0 = B9; // Match0
```

```
91 *TIMER1_MCR = B0 + B1;
92 *TIMER1_TCR = B0; // Timer0 ON
93 *VICIntSelect&=~B5; // Timer0 Int. es IRQ
94 *VICVectAddr1 = (int)TIMER1_VectISR;
95 *VICVectCntl1 = B5 + 0x5;
96 *VICIntEnable|=B5;
97 }
98
99 void TIMER1_ISR(void){
100    *TIMER1_IR |= 0x1; //anulem peticio int.
101    *VICVectAddr = 0x00; //EOI
102 }
103
104 void initdisplay(void)
105 {
106 *PINSEL0&=0x000FFFFF; // Pins P0.10-15 son GPIO
107 *PINSEL1&=0xF3FFFF00; // Pins P0.16-19 i P0.29 son GPIO
108 *IODIR0|=P10+P11+P12+P13+P14+P15+P16+P17+P18+P19+P29; // Pins P0.10:19 i P0
        .29 son sortides
109 *IOCLR0=P29; // Posem a 0 bit P0.29.
110 }
111
112 void retard(int n)
113 {
114 tics=0; //Inicialitzar la variable a 0
115 while (tics<n); //Esperar n tics fins a parar
116 }
117
118
119 int main (void)
120 {
121 int i=0; //Definimos variables para los bucles
122 int v[8]={P17,P16,P15,P14,P13,P12,P11,P10};//v[i] es el puerto del display i
        .
123 int numero[10]={P10+P11+P13+P15+P16+P17,P13+P15,P10+P11+P14+P15+P17,P11+P13+
        P14+P15+P17,P13+P14+P15+P16,P11+P13+P14+P16+P17,P10+P11+P13+P14+P16+P17,
        P13+P15+P17,P10+P11+P13+P14+P15+P16+P17,P13+P14+P15+P16+P17};//
        Codificamos los 10 numeros en codigo del seven-segments.
124 initdisplay(); //Iniciamos los puertos de los displays
125 initeclat(); //Iniciamos el teclado
126 initimer0(); //Iniciamos y configuramos el Timer0
127 *IOCLR0=P18+P19;
128    while(1){
129       *IOCLR0=P18+P19+P29; //Ponemos un 0 a los dos enables i al buffer para
             activar-lo
130       *IOCLR0=P10+P11+P12+P13+P14+P15+P16+P17; //Ponemos todos los puertos a
             0
131       if(j!=-1) {
132          *IOSET0=numero[(j)%10]; //Activamos los que tocan para poner la
                figura poisico[i], notese que depende del timer1.}
133           *IOSET0=P18; //Activamos el primer latch
134           *IOCLR0=P18; //Cuando ya hemos cargado los valores, desactivamos
                el primer latch
135           *IOSET0=P10+P11+P12+P13+P14+P15+P16+P17; //Podemos todos los
```

```
                puertos a uno
136             *IOCLR0=v[i]; //Activamos el display que queremos
137             *IOSET0=P19; //Activamos el segundo Latch
138             retard(1); //Dejamos tiempo de espera para poder cargar bien el
                    valor usando el Timer0
139             *IOCLR0=P19; //Desactivamos el latch que escoge que seven-segments
                     esta abierto
140             i=(i+1)%8; //Incremento modulo 8.
141         } else {
142            //Desactivar todos los displays
143             *IOSET0=P10+P11+P12+P13+P14+P15+P16+P17; //Podemos todos los
                    puertos a uno
144             *IOSET0=P19; //Activamos el segundo Latch
145         }
146     }
147 }
```

teclat.c

También hemos añadido la siguiente línea de código en el fichero *Interrupt.s*:

```
1 IRQHandle  P030_EINT3_ISR , EINT3_VectISR
```

Práctica 8: Altavoz

Durante esta práctica utilizaremos una salida del microcontrolador para excitar una bocina conectada a ella mediante ondas cuadradas. Es una práctica relativamente sencilla, ya que tan sólo tenemos que activar y desactivar una salida con una cierta frecuencia. No obstante, es una práctica muy vistosa porque al ser capaz de conseguir resultados sonoros diferentes, crea grandes expectativas sobre lo que se puede llegar a hacer con un microcontrolador.

En la práctica, reutilizamos el código de la práctica anterior, la del teclado, para activar la bocina a una cierta frecuencia, dependiendo de la tecla pulsada. Para ello, utilizamos un timer para generar la onda cuadrada para la bocina.

En un principio, intentamos programar una escala melódica, utilizando las frecuencias en una octava a partir del *LA 440 Hz*. No obstante, debido a que excitábamos con un pulso rectangular y a la calidad de la bocina, no éramos capaces de distinguir con facilidad los tonos. En una segunda prueba, pusimos frecuencias muy distintas, y las conseguimos distinguir a la perfección.

El código obtenido fue, en este caso:

```
1 #define __MAIN_C__
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <LPC_Base.h>
```

```
6  #include <LPC_SysControl.h>
7  extern void TIMER1_VectISR(void);
8  extern void TIMER0_VectISR(void);
9  unsigned int tics=0, j=0, t=0,x;
10 extern void EINT3_VectISR(void);
11 unsigned int polar=0;
12
13 void P030_EINT3_ISR(void) {
14    int k;
15    j=0;
16    *VICIntEnClr|=B17; //Deshabilitem EINT3
17    if (!polar)            //Canvi de polaritat EINT3
18       *EXTPOLAR=polar=B3;
19    else *EXTPOLAR=polar=0;
20    for(k=0; k<4; k++) {
21       *IOSET1 = P20+P21+P22+P23;
22       *IOCLR1 = 1<<(20+k);
23       x=(~*IOPIN1>>16)&0xF; //Obtenim les coordenades x (Nivell baix actiu)
24       switch (x) {
25       case 1:
26          j = 4*k;
27          break;
28       case 2:
29          j = 4*k+1;
30          break;
31       case 4:
32          j = 4*k+2;
33          break;
34       case 8:
35          j = 4*k+3;
36          break;
37       }
38    }
39    *IOCLR1 = P20+P21+P22+P23;
40    *VICIntEnClr|=B17; //Deshabilitem EINT3
41    *EXTINT|=P3;  //Anulem les int.pendents
42    *VICVectAddr = 0x00; //EOI
43    *VICIntEnable|=B17; //Habilitem EINT3
44 }
45
46 void initeclat(void) {
47    *PINSEL2&=~(P3); // Pins P1.25:16 son GPIO
48    *IODIR1|=P20+P21+P22+P23; // Pins P1.20:23 son sortides.
49    *IODIR1&=~(P16+P17+P18+P19); // Pins P1.16:19 son entrades
50    *IOCLR1=P20+P21+P22+P23;// Posem a 0 els bits P1.20-P1.23
51    *PINSEL1&=~B28; // Pin P0.30 es EINT3
52    *PINSEL1|=B29;
53    *EXTMODE=B3; // EINT3 es edge sensitive.
54    *EXTPOLAR=0; // EINT3 es falling-edge
55    *EXTINT|=P3;// Anulem qualsevol int. pendent
56    *VICIntSelect&=~B17; // EINT3 es IRQ (i no FIQ)
57    *VICVectAddr2 = (int)EINT3_VectISR;
58    *VICVectCntl2 = B5 + 17; // Fem servir VIC Channel 2
59    *VICIntEnable|=B17;
```

```
60 }
61
62 void inibocina(void) {
63    *PINSEL1 &= ~(B19+B18); //Pin P0.25 com GPIO
64    *IODIR0 |= P25; // Pin P0.25 com sortida.
65 }
66
67 void initimer0(void)
68 {
69 *TIMER0_TCR = B1; // Timer0 Reset
70 *TIMER0_PR = 0x27; // Prescaler
71 *TIMER0_MR0 = 0x01; // Match0
72 *TIMER0_MCR = B0 + B1;
73 *TIMER0_TCR = B0; // Timer0 ON
74 *VICIntSelect&=~B4; // Timer0 Int. es IRQ
75 *VICVectAddr0 = (int)TIMER0_VectISR;
76 *VICVectCntl0 = B5 + 0x4;
77 *VICIntEnable|=B4;
78 }
79
80 void TIMER0_ISR(void)
81 {
82 tics++;
83 *TIMER0_IR |= 0x1; //anulem peticio int.
84 *VICVectAddr = 0x00; //EOI del VIC
85 }
86
87 void retard(int n)
88 {
89 tics=0; //Inicialitzar la variable a 0
90 while (tics<n); //Esperar n tics fins a parar
91 }
92
93
94 int main (void)
95 {
96 int PeriodesSo[8] = {10,100,1000,100,10000,1000000,1,1};
97 int BocinaActivada = 0;
98 initeclat(); //Iniciamos el teclado
99 initimer0(); //Iniciamos y configuramos el Timer0
100 inibocina(); //Iniciamos la bocina
101    while(1){
102       if(tics>PeriodesSo[j%8]) {
103          tics = 0;
104          if(BocinaActivada == 0) {
105             BocinaActivada = 1;
106             *IOSET0 = P25;
107          } else {
108             BocinaActivada = 0;
109             *IOCLR0 = P25;
110          }
111       }
112    }
113 }
```

bocina.c