Proyecto "pastillero"

Table of Contents

Descargar versión PDF.

Ver video en Politube (si no sale, buscar en politube.upv.es ("pastillero 8051")

1. Objetivo

El propósito principal de este proyecto es enseñar cómo se desarrolla el software para un sistema típico basado en microcontrolador.

Se ha elegido un proyecto que permite aplicar los conocimientos que se van adquiriendo en el transcurso de la asignatura.

Estos conocimientos se aplican desarrollando pequeñas funciones que, al final, formarán un compleja aplicación totalmente operativa y muy similar a la que se desarrollaría en un producto comercial análogo (puede que esto tuviese salida comercial, habría que verlo).

La manera de repartir esas funciones en módulos y de coordinarlos para lograr una aplicación compleja es también parte de este objetivo. El alumno ve así para que sirven todas esas pequeñas funciones que ha ido realizando y cómo facilitan el desarrollo de la aplicación al ir abstrayendo la complejidad del microcontrolador en funciones más fáciles de entender para nosotros.

Desde el punto de vista de metodología didáctica, esto va a ser un aprendizaje mediante proyectos o PBL (Project Based Learning) que busca:

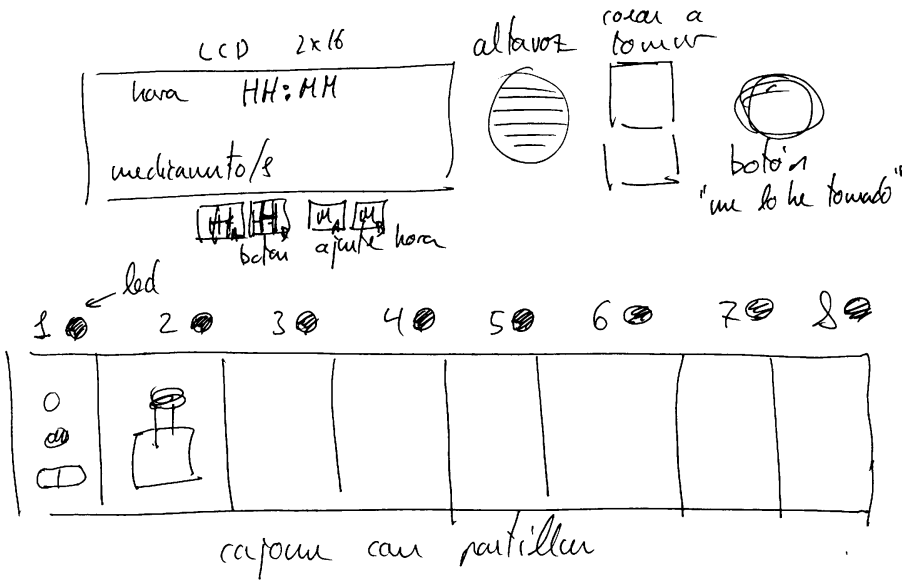
- Motivar, pues el resultado es conseguir algo real que funcionará (espero).

- Aprender aspectos fundamentales de micros en forma de pequeños bloques (manejar puertos, etc.)
- Aprender a usar esos pequeños "bloques" para formar algo más grande (como en la 1a. parte de la asignatura)
- Integrar los bloques y otros proporcionados, como haría un profesional, para formar una aplicación totalmente operativa y lógica desde el punto de vista del mantenimiento del software (facilidad de configuración, de introducción de cambios, de depuración, etc.)

2. Requisitos del sistema

La siguiente figura muestra el croquis de lo que se quiere "crear": un *sistema dosificador de medicamentos diario*, en adelante *pastillero*.

Figure 1. Aspecto del pastillero



Este sistema deberá permitir organizar la administración de hasta 4 tomas de medicamentos en 24 horas.

Dispondrá de 4 cajetines donde introducir los medicamentos con un indicador luminoso por cajetían para indicar cuál se debe usar en la administración.

Una pantalla indicará la hora actual y la denominación de los siguientes medicamentos a tomar. Unos botones permitirán ajustar la hora correcta.

Una señal sonora indicará el momento en el que se debe realizar la toma, y un indicador mostrará en número de dosis o unidades a suministrar.

Un pulsador permite informar de que un medicamento ha sido administrado. Se deberá pulsar éste tantas veces como dosis se deban administrar.

3. Salida digital. Manejo de un display de 7 segmentos

Descargar esta versión.

Vamos a empezar por lo que, creo, es más sencillo de entender: la generación de salidas digitales. Recordar que la salida digital se genera escribiendo en los latch de los puertos.

Como actividad se ha propuesto diseñar un módulo C que permita manejar el display de 7 segmentos del equipo de prácticas.

En posteriores clases se irán comentando distintas problemáticas de esta primera solución que permitirán llegar a una propuesta más óptima.

Cómo objetivo adicional, se pretende ilustrar la manera en que este módulo y otros crean una abstracción más "humana" para gestionar los componentes físicos (leds, pulsadores, ...) conectados al micro y, en las actividades de laboratorio, cómo se puede probar el correcto funcionamiento del módulo antes de pasar a elaborar otro aspecto de la aplicación.

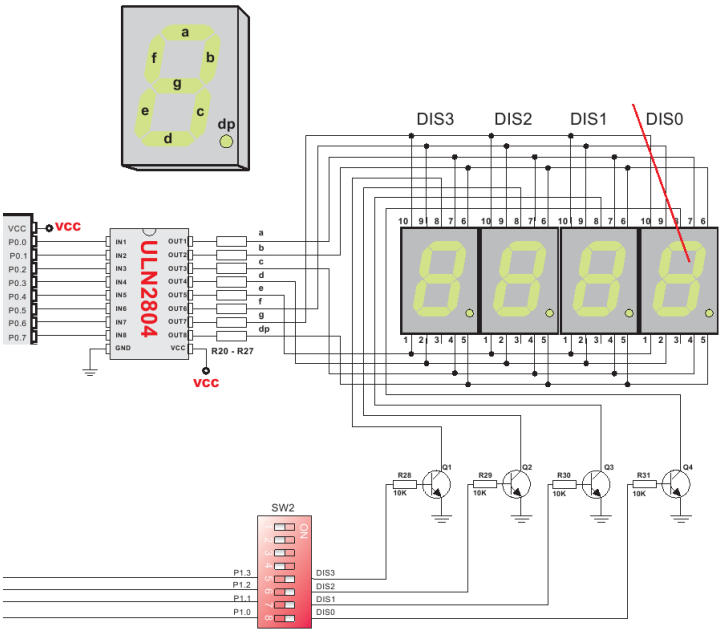
El objetivo que debe resolver el código a desarrollar es mostrar un número entre 0 y 9 en el display. Se proponen entonces los prototipos de funciones del módulo para desarrollar esta tarea que se listan abajo:

```
// Fichero: display7seg.h

#ifndef display7seg_H
#define display7seg_H

/* Función encargada de preparar el harware antes de ser usado */
void display7seg_inicializar(void);

/* Muestra un número entre 0 y 9 en el display de 7 segmentos */
void display7seg_numero(unsigned char valor);


#endif
```

El funcionamiento del display particular del equipo se ha analizado durante la clase con los alumnos. La idea es recrear de la manera más real posible lo que nos encontraremos en nuestro trabajo y cómo enfocar su resolución.

La siguiente figura muestra el esquema de interconexión de los displays del equipo.

Figure 2. Esquemático del display de 7 segmentos.



Los segmentos del display se necienden poniendo a 0 el bit del puerto 0 que corresponda. El display seleccionado es el que tenga a 1 el bit del puerto P1 que le corresponda.

Se propone pues la siguente implementación de las funciones.

```
// Fichero: display7seg.c

#include "display7seg.h"
```

```
sfr P1=0x90;
sfr P0=0x80;

void display7seg_inicializar(void)
{
   P1=0xF1;   // activar display de menor peso y dejar los bits de mayor peso a 1
   P0=0xFF;   // todos los segmentos del display apagados

}

void display7seg_numero(unsigned char valor)
{
   // enviar al puerto 0 la combinación que enciende los segmentos
   // para mostrar el número pasado como argumento
   switch(valor)
   {
   case 0:
      P0=0x40;
      break;
   case 1:
      P0=0x79;
      break;
   case 2:
      P0=0x24;
      break;
   case 3:
      P0=0x30;
      break;
   case 4:
      P0=0x19;
      break;
   case 5:
      P0=0x12;
      break;
   case 6:
      P0=0x02;
      break;
   case 7:
      P0=0x78;
      break;
   case 8:
      P0=0x00;
      break;
   case 9:
      P0=0x18;
      break;
   }
}
```

Ahora es el momento de validar el código. Para ello se desarrolla el siguiente programa principal sencillo para probar su funcionamiento.

```
// Fichero: test7seg.c

// programa para probar el correcto funcionamiento de las funciones desarrolladas
// para el display

sfr P1=0x90;

#include "display7seg.h"

void pausa(void) {

   unsigned int j;

   for(j=0;j<10000;j++);

}
```

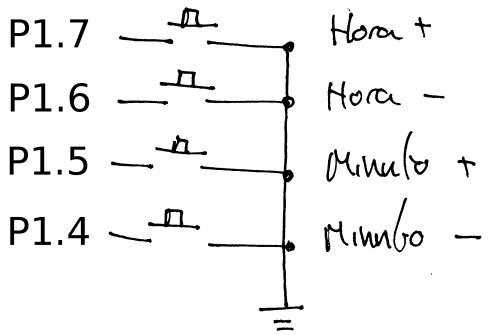
```
void main(void)
{

   unsigned char numerin;
   unsigned char i;
   unsigned int j;

   pausa(); // para saltarse el fallo del grabador de la placa Mikroe

   display7seg_inicializar();

   numerin=0;
   display7seg_numero(numerin);

   while(1) {
      if ((P1 & 0x80)==0)   // si se pulsa P1.7
      {
         if (numerin < 9)
         {
            numerin = numerin+1;
            display7seg_numero(numerin);
            while ((P1 & 0x80)==0);  // esperar a que se suelte la tecla
         }
      }

      if ((P1 & 0x40)==0)   // si se pulsa P1.6
      {
         if (numerin > 0)
         {
            numerin = numerin-1;
            display7seg_numero(numerin);
            while ((P1 & 0x40)==0);
         }
      }
   }
}
```

4. Lectura digital. Pulsadores para la hora.

Se van a utizar ahora los pines de los puertos para hacer lecturas digitales.

Como ejemplo se utilizarán unos pulsadores conectados al puerto P1 que se conectan según el siguiente diagrama. Observar que se aprovecha al rresistencia de pull-up interna de la célula del puerto con lo que los pulsadores lo único que hacen es poner a masa el pin para generar un "0".

Figure 3. Diagrama de conexión de los pulsadores para modificar la hora



Siguiendo la filosofía de la primera parte de la asignatura y del apartado anterior, se ha propuesto en clase una serie de funciones que se encapsularán en módulo que abstraerán la funcionalidad de estos pulsadores y los detalles de implementación física.

El siguiente fichero de cabecera contiene los prototipos y definiciones para emplear los pulsadores.

```
// fichero: pulsadores.h

#ifndef pulsadores_H  // esto es el "protector" de cabeceras
#define pulsadores_H  // que evita múltiples inclusiones de una cabecera

// preparar el hardware relacionado con el teclado
void pulsadores_inicializar(void);

// etiquetas propuestas a devolver por la función de teclado
enum {PULS_NADA, PULS_HORA_MAS, PULS_HORA_MENOS, PULS_MINUTO_MAS, PULS_MINUTO_MENOS};
unsigned char pulsadores_leer(void);

#endif // fin del protector de cabecera
```

Y aquí la implementación de las funciones propuestas. La función de lectura de pulsadores lee el puerto completo, aplica máscaras y analiza la combinación obtenida. Destacar cómo se hace el filtrado de los pulsadores, pues se consigue que pulsaciones simulatneas de varias teclas no afecta a la aplicación.

```
// Fichero: pulsadores.c

#include "pulsadores.h"

sfr P1=0x90;

// preparar el hardware de lectura de pulsadores
void pulsadores_inicializar(void){

   P1 |= 0xF0; // la semana que viene explicamos esto

}

// leer un pulsador y devolver una "etiqueta" que lo identifica
unsigned char pulsadores_leer(void){

   //unsigned char valor;
   //unsigned char res;

   //valor=P1;
   //res=valor&0xF0;

   switch(P1 & 0xF0){   // en lugar de switch(res), ahorrannndo!!!
      case 0x70:
         return(PULS_HORA_MAS);
         break;
      case 0xB0:
         return(PULS_HORA_MENOS);
         break;
      case 0xD0:
         return(PULS_MINUTO_MAS);
         break;
      case 0xE0:
         return(PULS_MINUTO_MENOS);
         break;
      default:
         return(PULS_NADA);
         break;
   }
}
```

Este módulo habría que validarlo. Se hace en el siguiente apartado junto con el módulo de sonido.

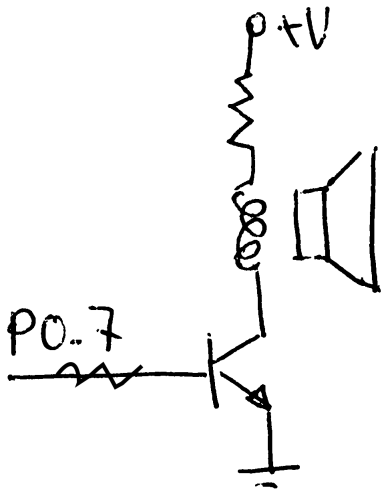
5. Manipulación bit a bit. Un altavoz

Descargar esta versión

Se propone emplear el altavoz como ejemplo de manejo de los puertos mediante instrucciones de manipulación de bits individuales. El altavoz se ha conectado según el siguiente diagrama al pin P2.0 del microcontrolador.

En clase, en lugar de un altavoz convencional se ha empleado uno cerámico y se ha eliminado el transistor.

Figure 4. Diagrama de conexión del altavoz



En clase se ha propuesto como ejercicio generar la nota musical LA en el altavoz. Para ello se debía sintonizar el contador de un bucle de retardo a partir de "imaginar" qué instrucciones máquina se generaban y que el micro emplea un reloj de 8 Mhz.

La nota LA tiene una frecuenciade 400 y pico Hz. (no recuerdo cuánto me habeis dicho).

A continuación está la cabecera propuesta de un nuevo módulo para gestionar el sonido.

```
// Fichero: sonido.h

#ifndef sonido_H
#define sonido_H

// preparar el hardware
void sonido_inicializar(void);

// generar la nota LA
void sonido_LA(void);

#endif
```

Y la implementación.

```
// Fichero: sonido.c

#include "sonido.h"


sfr P2=0xA0;
sbit altavoz = P2^0;

// preparar el hardware
void sonido_inicializar(void) {

   // nada, de momento
}

// en el equipo de prácticas saca una nota LA desafinada
#define PAUSA 75
```

```
void sonido_LA(void) {

   unsigned int data i,j;

   j = 50;
   while(j > 0) {
      j--;
      altavoz = 1;
      i = PAUSA;
      while(i > 0) {
         i--;
      }
      altavoz = 0;
      i = PAUSA;
      while(i > 0) {
         i--;
      }
   }
}
```

La verdad es que no acertamos mucho en el ensamblador generado para poder sintonizar el bucle. Aquí va el ensamblador que genera el compilador, que siempre suele ser más listo que nosotros (entre otras cosas, por eso es mejor programar en C).

```
C51 COMPILER V7.05   SONIDO                                                                04/22/2008 16:


C51 COMPILER V7.05, COMPILATION OF MODULE SONIDO
OBJECT MODULE PLACED IN sonido.OBJ
COMPILER INVOKED BY: C:\silabs\mcu\IDEfiles\C51\BIN\C51.EXE sonido.c BROWSE DEBUG OBJECTEXTEND CODE LISTI

stmt level    source

   1          // Fichero: sonido.c
   2
   3          #include "sonido.h"
   1      =1  // Fichero: sonido.h
   2      =1
   3      =1  #ifndef sonido_H
   4      =1  #define sonido_H
   5      =1
   6      =1  void sonido_inicializar(void);
   7      =1  void sonido_LA(void);
   8      =1
   9      =1  #endif
  10      =1
   4
   5
   6          sfr P2=0xA0;
   7          sbit altavoz = P2^0;
   8
   9          // preparar el hardware
  10          void sonido_inicializar(void) {
  11   1
  12   1              // nada, de momento
  13   1      }
  14
  15          // en el equipo de prácticas saca una nota LA desafinada
  16          #define PAUSA 75
  17          void sonido_LA(void) {
  18   1
  19   1         unsigned int data i,j;
  20   1
  21   1         j = 1000;
  22   1         while(j > 0) {
  23   2                j--;
  24   2            altavoz = 1;
  25   2            i = PAUSA;
```

```
  26   2            while(i > 0) {
  27   3               i--;
  28   3            }
  29   2            altavoz = 0;
  30   2            i = PAUSA;
  31   2            while(i > 0) {
  32   3               i--;
  33   3            }
  34   2         }
  35   1      }
  36
C51 COMPILER V7.05   SONIDO                                                              04/22/2008 16:

ASSEMBLY LISTING OF GENERATED OBJECT CODE


            ; FUNCTION sonido_inicializar (BEGIN)
                                          ; SOURCE LINE # 10
                                          ; SOURCE LINE # 13
0000 22                 RET
            ; FUNCTION sonido_inicializar (END)

            ; FUNCTION sonido_LA (BEGIN)
                                          ; SOURCE LINE # 17
;---- Variable 'i' assigned to Register 'R6/R7' ----
;---- Variable 'j' assigned to Register 'R4/R5' ----
                                          ; SOURCE LINE # 21
0000 7C03               MOV     R4,#03H
0002 7DE8               MOV     R5,#0E8H
0004         ?C0002:
                                          ; SOURCE LINE # 22
0004 D3                 SETB    C
0005 ED                 MOV     A,R5
0006 9400               SUBB    A,#00H
0008 EC                 MOV     A,R4
0009 9400               SUBB    A,#00H
000B 402A               JC      ?C0008
                                          ; SOURCE LINE # 23
000D ED                 MOV     A,R5
000E 1D                 DEC     R5
000F 7001               JNZ     ?C0009
0011 1C                 DEC     R4
0012         ?C0009:
                                          ; SOURCE LINE # 24
0012 D2A0               SETB    altavoz
                                          ; SOURCE LINE # 25
0014 7E00               MOV     R6,#00H
0016 7F4B               MOV     R7,#04BH
0018         ?C0004:
                                          ; SOURCE LINE # 26
                                          ; SOURCE LINE # 27
0018 EF                 MOV     A,R7
0019 1F                 DEC     R7
001A 7001               JNZ     ?C0010
001C 1E                 DEC     R6
001D         ?C0010:
                                          ; SOURCE LINE # 28
001D EF                 MOV     A,R7
001E 4E                 ORL     A,R6
001F 70F7               JNZ     ?C0004
0021         ?C0005:
                                          ; SOURCE LINE # 29
0021 C2A0               CLR     altavoz
                                          ; SOURCE LINE # 30
0023 7E00               MOV     R6,#00H
0025 7F4B               MOV     R7,#04BH
0027         ?C0006:
                                          ; SOURCE LINE # 31
0027 D3                 SETB    C
0028 EF                 MOV     A,R7
```

```
0029 9400              SUBB    A,#00H
002B EE                MOV     A,R6
002C 9400              SUBB    A,#00H
002E 40D4              JC      ?C0002
                                           ; SOURCE LINE # 32
C51 COMPILER V7.05   SONIDO                                                    04/22/2008 16:

0030 EF                MOV     A,R7
0031 1F                DEC     R7
0032 70F3              JNZ     ?C0006
0034 1E                DEC     R6
0035         ?C0011:
                                           ; SOURCE LINE # 33
0035 80F0              SJMP    ?C0006
                                           ; SOURCE LINE # 34
                                           ; SOURCE LINE # 35
0037         ?C0008:
0037 22                RET
             ; FUNCTION sonido_LA (END)

C51 COMPILER V7.05   SONIDO                                                    04/22/2008 16:

NAME                                  CLASS   MSPACE  TYPE    OFFSET  SIZE
====                                  =====   ======  ====    ======  ====


P2 . . . . . . . . . . . . . . . . .  SFR      DATA   U_CHAR   00A0H  1
altavoz. . . . . . . . . . . . . . .  ABSBIT   -----  BIT      00A0H  1
sonido_inicializar . . . . . . . . .  PUBLIC   CODE   PROC     0000H  -----
sonido_LA. . . . . . . . . . . . . .  PUBLIC   CODE   PROC     0000H  -----
  i. . . . . . . . . . . . . . . . .  * REG *  DATA   U_INT    0006H  2
  j. . . . . . . . . . . . . . . . .  * REG *  DATA   U_INT    0004H  2


MODULE INFORMATION:   STATIC OVERLAYABLE
   CODE SIZE        =     57    ----
   CONSTANT SIZE    =   ----    ----
   XDATA SIZE       =   ----    ----
   PDATA SIZE       =   ----    ----
   DATA SIZE        =   ----    ----
   IDATA SIZE       =   ----    ----
   BIT SIZE         =   ----    ----
END OF MODULE INFORMATION.


C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

Para probar este módulo y el de pulsadores se propone el siguiente programa principal.

```
// Fichero: test_puls.c

// programa para probar el funcionamiento de los pulsadores
// y el sonido

#include "display7seg.h"
#include "pulsadores.h"
#include "sonido.h"

void pausa(void) {
   unsigned int j;

   for(j=0;j<10000;j++);
}

// programa principal
void main(void)
{

   unsigned char numero;
```

```
   unsigned char tecla;

   pausa(); // para saltarse el fallo del grabador de la placa Mikroe

   display7seg_inicializar();
   pulsadores_inicializar();
   sonido_inicializar();

   //sonido();

   numero = 5;
   display7seg_numero(numero);

   while(1) { // superbucle
      tecla = pulsadores_leer();

      if ((tecla == PULS_HORA_MAS) && (numero < 9))
      {
            numero = numero + 1;
            display7seg_numero(numero);
            sonido_LA();
            while (pulsadores_leer() != PULS_NADA) {};   // esperar a que se suelte la tecla
      } else if ((tecla == PULS_HORA_MENOS) && (numero > 0))
      {
            numero = numero - 1;
            display7seg_numero(numero);
            sonido_LA();
            while (pulsadores_leer() != PULS_NADA) {};   // esperar a que se suelte la tecla
       }
   } /* del while(1) */
}
```

6. Operaciones L-M-E. Modificación display 7 segmentos. Indicadores luminosos

Descargar esta versión

Descargar esta versión con glcd

Por lo que se ha visto hasta ahora, los puestos del microcontrolador se pueden manejar en grupos de 8 líneas o bit a bit.

En muchos casos, un puerto se suele utilizar para conectar más de un dispositivo, cada uno de los cuales puede tener más de una línea de control. Pare esos casos, la mejor manera de hacer programas en esta arquitectura es conseguir que las operaciones que se hacen sobre un dispositivo en un puerto compartido no afecten a los otros dispositivos conectados a ese mismo puerto.

La opción más viable es emplear las instruccciones de lectura-modificación-escritura que trabajan operando con la información contenida en los latch de los puertos (o en cualquier otro registro SFR).

Como ejercicio en clase, se han modificado las rutinas que gestionan el display de 7 segmentos para liberar la línea P0.7, quedando el módulo de la siguiente manera.

```
// Fichero: display7seg.c

#include "display7seg.h"

sfr P1=0x90;
sfr P0=0x80;

void display7seg_inicializar(void)
{
   P1 &= 0xF0;   // activar display de menor peso
   P1 |= 0x01;
```

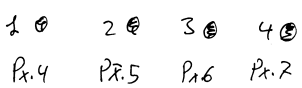
```
   P0 |= 0x7F;   // todos los segmentos del display apagados
}

// enviar al puerto 0 la combinacion que enciende los segmentos
// para mostrar el numero pasado como argumento
// esta versión utiliza instrucciones L-M_E para no tocar P0.7
void display7seg_numero(unsigned char valor)
{
   P0 |= 0x7F;   // apagar los segmentos
   switch(valor)
   {
   case 0:
      P0 &= 0xC0;
      break;
   case 1:
      P0 &= 0xF9;
      break;
   case 2:
      P0 &= 0xA4;
      break;
   case 3:
      P0 &= 0xB0;
      break;
   case 4:
      P0 &= 0x99;
      break;
   case 5:
      P0 &= 0x92;
      break;
   case 6:
      P0 &= 0x82;
      break;
   case 7:
      P0 &= 0xF8;
      break;
   case 8:
      P0 &= 0x80;
      break;
   case 9:
      P0 &= 0x98;
      break;
   }
}
```

Ahora que P0.7 está libre, se ha aprovechado para remodelar el módulo de sonido para que emplee esa línea.

Utilizando la misma filosofía, se ha desarrollado un módulo para indicadores luminosos que marcan el cajetín con medicamentos que se ha de consumir.

Se propone la siguiente asignación de indicadores conectados al puerto P3.

Figure 5. Asignación de indicadores a pines de puerto



Se propone un módulo cuya cabecera es:

```
// Fichero: indicador.h

#ifndef indicador_H
#define indicador_H

void indicador_inicializar(void);
```

```
enum {INDICADOR_ON, INDICADOR_OFF};
void indicador_modificar(unsigned char num_indic, unsigned char estado);

#endif
```

Y la implementación es la siguiente:

```
// Fichero: indicador.c
// cajon 1=P3.4, 2=P3.5, 3=P3.6, 4=P3.7

#include "indicador.h"

sfr P3 = 0xB0;

void indicador_inicializar(void) {
      P3 |= 0x0F;      // todos apagados
}

void indicador_modificar(unsigned char num_indic, unsigned char estado) {

      if ((num_indic < 1) || (num_indic > 4))
         return;

      if (estado == INDICADOR_ON) {
                  P3 &= ~(1 << (num_indic+3)); // encender
      } else {
                  P3 |= (1 << (num_indic+3)); // apagar
      }
}
```

7. Interrupciones externas. Gestión del pulsador de tomas

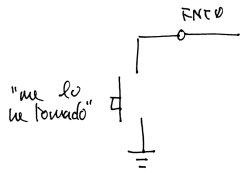
Descargar esta versión.

Se pretende ahora mostrar un ejemplo de como configurar el sistema de interrupciones del 51 y desarrollar rutinas de servicio.

Para ello se conectará el pulsador de tomas a la entrada de interupción externa 0 y se desarrollará el software capaz de descontar una unidad de las tomas cada vez que se produzca una pulsación.

El siguiente diagrama muestra cómo se ha conectado el pulsador de tomas.

Figure 6. Diagrama de conexión del pulsador de tomas



Para esta implementación se propone utilizar una variable global con la cantidad de tomas a ingerir. El pulsador deberá producir una interrupción cada vez que se pulse y la rutina de servicio descontará una unidad de la variable contador.

Como prueba de funcionamiento, el programa principal será un bucle que actualizará el display de 7 segmentos contínuamente con el valor de la variable compartida.

Se proponen la siguiente implementación. Destacar cómo se ha hecho la programación del sistema de interrupciones siguiendo los pasos explicados en clase y que la interrupción se está programando por flanco de bajada.

```
// Fichero: test_puls.c

#include "display7seg.h"
#include "pulsadores.h"
#include "sonido.h"
#include "indicador.h"

#include <reg51.h>

// esto es para el GLCD. Como si no estuviese
sbit PLCD_CS1=P2^0;
sbit PLCD_CS2=P2^1;
sbit PLCD_E=P2^4;

unsigned char tomas_contador;

void pausa(unsigned long cnt) {
   while(cnt--){};
}

// programa principal ------------------------------------------------------
void main(void)
{
   pausa(10000); // para saltarse el fallo del grabador de la placa Mikroe

   PLCD_CS1=1;PLCD_CS2=1;PLCD_E = 0; // liberar las lineas del GLCD
   display7seg_inicializar();
   pulsadores_inicializar();
   sonido_inicializar();
   indicador_inicializar();

   tomas_contador = 9;
   display7seg_numero(tomas_contador);

   // programar interrupción externa 0
   EA = 0;   // desconectar sistema interrupciones
   IT0 = 1;  // programar INT0 por flanco de bajada
   EX0 = 1;  // habilitar interrupción INT0
   EA = 1;   // habilitar sistema de interrupciones

   while(1) {// superbucle
      display7seg_numero(tomas_contador);
      pausa(100);
   }
}

// servicio interrupción externa 0 -----------------------------------------
// pin P3.2/INT0
void tomas_ISR (void) interrupt 0 {

   if (tomas_contador > 0) {
      tomas_contador--;
   }
}
```

Aparentemente, este programa es correcto y no ha dado ningún problema. Sin embargo, un concepto fundamental cuando se emplean interrupciones, programación paralela, etc son las denominadas *condiciones de carrera*. Este programa no da problema por ser la variable contador de 8 bits, si se tratase de una varible de 16 bits, podríamos encontrarnos con esta situación.

8. Timers. Llevando la cuenta del tiempo y haciendo tareas periódicas

Descargar esta versión

El pastillero debe saber gestionar el transcurso del tiempo para saber cuándo administrar una medicación.

Esta es la excusa perfecta para ilustrar cómo se puede llevar la cuenta del transcurso del tiempo utilizando un timer y combinándolo con el sistema de interrupciones.

Este ejemplo pretende mostrar las posibilidades de los timers para crear tareas periódicas. Asímismo se pretende dar las líneas generales de cómo se suele llevar la cuenta del tiempo real en un sistema microcontrolador que no esté dotado (como la mayoría) de un reloj de tiempo real hardware.

Se proponen la siguientes funciones para el módulo C encargado de gestionar el tiempo (`reloj.h`).

```
// reloj.h

#ifndef reloj_H
#define reloj_H

void reloj_inicializar(void);
unsigned long reloj_rtc_leer(void);
void reloj_wall_get(unsigned char *hora, unsigned char *minuto, unsigned char *segundo);
void reloj_wall_set(unsigned char hora, unsigned char minuto, unsigned char segundo);

#endif
```

Las funciones pretenden que el manejo de este módulo sea muy intiutivo usando nuestro modo "humano" de representar la hora del día.

Y esta es la posible implementación de estas funciones (reloj.c).

```
// Fichero: reloj.c
// Gestion del tiempo humano

#include "display7seg.h"
#include "sonido.h"
#include "reloj.h"

#include <reg51.h>

#define TICK_TIME 5 / 5 mS entre ticks

// 5 ms a 8 Mhz, 1 CM=12/8 uS, 5 ms = 13333 CM, carga= 65536-13333 -> CBEBh
#define RECARGA_TH0 0xF2
#define RECARGA_TL0 0xFB

static unsigned long rtc; // variable para guardar el tiempo

// puesta a punto del sistema de medición de tiempo y tareas periódicas ------
void reloj_inicializar(void) {

   rtc = 0; // empezamos en las 0:00:00

   // configurar interrupciones según reglas de clase
   EA = 0;
   // configurar timer 0
   TR0 = 0;
   TF0 = 0;
   TMOD &= 0xF0; // modo 1
   TMOD |= 0x01;
   TH0 = RECARGA_TH0;
   TL0 = RECARGA_TL0;
   ET0 = 1;
   EA = 1;
   TR0 = 1; // arrancarrrr
}

// rutina de servicio de la ieterrupción del timer 0 --------------------------
void reloj_tick_ISR (void) interrupt 1 {

   // lo primero es que el reloj sea los más preciso posible
   TR0 = 0;
   // TF0 = 0; // borrar flag, no hace falta, lo hace el hardware
```

```
   TH0 = RECARGA_TH0;
   TL0 = RECARGA_TL0;
   TR0 = 1; // arrancar

   // actualizar el reloj de tiempo real
   rtc++;

   // refrescar el display de 7 segmentos
   display7seg_refrescar();

   // tocar si hay algo que tocar
   sonido_musica_tick();
}

// lectura del reloj de tiempo real ----------------------------------------
unsigned long reloj_rtc_leer(void) {

   unsigned long tmp;

   EA=0; // leer la variable contador evitando una "condición de carrera"
   tmp = rtc;
   EA=1;

   return(rtc);
}

// lectura del reloj en formato "humano" -------------------------------------
void reloj_wall_get(unsigned char *hora, unsigned char *minuto, unsigned char *segundo) {

   unsigned long tmp;

   tmp = reloj_rtc_leer();

   tmp = tmp / (1000/TICK_TIME);
   *segundo = tmp % 60; // extraer los segundos
   tmp = tmp/60;
   *minuto = tmp % 60;  // los minutos
   tmp = tmp/60;
   *hora = tmp % 60;   // las horas
   // y se podría continuar igual sacando días, semanas, ...
}

// escritura en el reloj del sistema usando formato "humano" -----------------
void reloj_wall_set(unsigned char hora, unsigned char minuto, unsigned char segundo) {

   unsigned long tmp;

   tmp = hora; /* pasar hora humana a reloj del sistema */
   tmp = tmp*60 + minuto;
   tmp = tmp*60 + segundo;
   tmp = tmp*(1000/TICK_TIME);

   EA=0; // endosar el reloj evitando una "condición de carrera"
   rtc = tmp;
   EA=1;
}
```

Destacar de esta propuesta los siguientes aspectos fundamentales:

- Se emplea una rutina de servicio para el timer con recarga software del contador. Es una solución adecuada si el micro no ofrece otra posibilidad (por ejemplo, timers con autorrecarga con mayor capacidad). *Este reloj atrasará*.
- La función para establecer la hora tiene especial precaución en evitar desbordes en las operaciones con los tipos de datos enteros.
- El acceso al reloj del sistema `clock` se hace evitando *condiciones de carrera* y lo más rápidamente posible.

y los pulsadores para actualizar la hora del sistema.

```
// Fichero: main.c

#include "display7seg.h"
#include "pulsadores.h"
#include "sonido.h"
#include "indicador.h"
#include "reloj.h"
#include "canciones.h"


#include <reg51.h>

// esto es para el GLCD. Como si no estuviese
sbit  PLCD_CS1 =P2^0;
sbit  PLCD_CS2 =P2^1;
sbit  PLCD_E  =P2^4;

void pausa(unsigned long cnt) {
   while(cnt--){};
}

// programa principal
void main(void)
{
   unsigned long segundos;

   pausa(10000); // para saltarse el fallo del grabador de la placa Mikroe
   PLCD_CS1=1;PLCD_CS2=1;PLCD_E = 0; // liberar las lineas del GLCD

   display7seg_inicializar();
   pulsadores_inicializar();
   sonido_inicializar();
   indicador_inicializar();
   reloj_inicializar();

   // poner los segundos en el indicador
   segundos = reloj_rtc_leer() / 200;
   display7seg_numero(segundos);

   //sonido_frecuencia(440);
   sonido_musica_tocar(popcorn);
   //sonido_musica_tocar(coca_cola);
   //sonido_musica_tocar(where_is);

   while(1) { // superbucle
      pausa (1000);      // no agobiar al reloj desconectando demasiado las interrupciones
      segundos = reloj_rtc_leer() / 200;
      display7seg_numero(segundos);
   } /* del while(1) */
}
```

9. Gestor de medicamentos

Falta un módulo para gestionar los medicamentos depositados en los cajones. Se propone en módulo `medicinas.c` para realizar esta tarea.

Dicho módulo se ha desarrollado en C sin utilizar ninguna característica específica del microcontrolador. Ello permite que se pueda prescindir del compilador para el micro y emplear un entorno de desarrollo para el PC que ofrece características bastante más potentes a la hora de depurar este módulo.

Mi práctica habitual es intentar desarrollar el máximo de código independiente de la plataforma y hacerlo en un entorno potente como por ejemplo el Borland C++ Builder. Una vez depurado el código, se lleva al compilador para el micro con bastante seguridad de que va a funcionar correctamente.

Aquí puedes descargar la versión desarrollada en Builder de una versión anterior para que veas la idea.

Aquí va el listado de `medicinas.h`

```
//---------------------------------------------------------------------------
// medicinas.h

#ifndef medicinasH
#define medicinasH

#define NUM_CAJONES 4

/* preparar el gestor de medicamentos */
void medicinas_inicializar(void);

void medicinas_prescripcion_set(unsigned char n_cajon, unsigned char hora, unsigned char minuto,
 unsigned char num_tomas, unsigned char *texto);
/* consultar la configuración de un cajon */
void medicinas_prescripcion_get(unsigned char n_cajon, unsigned char *hora, unsigned char *minuto,
 unsigned char *num_tomas, unsigned char **texto);


//---------------------------------------------------------------------------
#endif
```

Y el listado de `medicinas.c`

```
/* Fichero: medicinas.c
   Para la gestión de la base de datos de medicamentos en cajones
*/

#include "medicinas.h"
#include <string.h>

/* estructura para mantener la cantidas de tomas por cajon */
/* y la hora de la toma */
static struct {
   unsigned char num_tomas;
   unsigned char hora;
   unsigned char minuto;
   unsigned char *prescripcion;
   } cajon [NUM_CAJONES];

// prepara y limpiar los cajones -----------------------------------------------
void medicinas_inicializar(void) {
   data unsigned char i;

   for (i=0;i<NUM_CAJONES;i++) {
      cajon[i].num_tomas = 0; // indica que está vacio
   }
}

// establecer la configuración de un cajon -------------------------------------
void medicinas_prescripcion_set(unsigned char n_cajon, unsigned char hora, unsigned char minuto,
unsigned char num_tomas, unsigned char *texto) {
   if ((n_cajon >=1) && (n_cajon<=NUM_CAJONES)) { // cajon válido
      cajon[n_cajon-1].hora = hora;
      cajon[n_cajon-1].minuto = minuto;
      cajon[n_cajon-1].num_tomas = num_tomas;
      cajon[n_cajon-1].prescripcion = texto;
   }
}

// consultar la configuración de un cajon --------------------------------------
void medicinas_prescripcion_get(unsigned char n_cajon, unsigned char *hora, unsigned char *minuto,
unsigned char *num_tomas, unsigned char **texto) {
   if ((n_cajon >=1) && (n_cajon<=NUM_CAJONES)) { // cajon válido
      *hora = cajon[n_cajon-1].hora;
      *minuto = cajon[n_cajon-1].minuto;
```

```
      *num_tomas = cajon[n_cajon-1].num_tomas;
      *texto = cajon[n_cajon-1].prescripcion;
   }
}
```

10. Uniéndolo todo ... para tener el pastilerooo

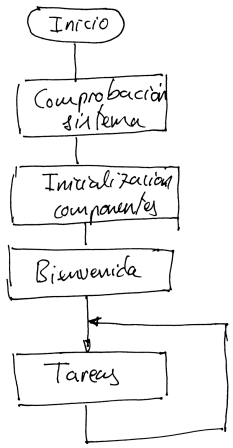
Descargar esta versión.

Recordemos que hemos aplicado una aproximación bottom-up para el desarrollo del pastillero. Es decir, hemos ido desarrollando piezas pequeñas muy concretas y las hemos ido probando.

En este punto ya hay suficientes piezas y se puede empezar a desarrollar el programa principal del pastillero con los elementos de los que se dispone. Además de los módulos listados aquí, se han añadido un módulo musical (gracias, Raúl, por las partituras) y un módulo para gestionar el display gráfico (que falta documentar y arreglar un poco).

Empezaremos viendo el siguiente organigrama que muestra el flujo de la ejecución de un programa típico para microcontrolador.

Figure 7. Organigrama de un programa típico para microcontrolador



Más o menos cada bloque tiene el siguiente cometido:

- *Comprobación sistema* se encarga de hacer una prueba de los elementos electrónicos. En general, todo aquello susceptible de averiarse y provocar un funcionamiento indebido del sistema, y siempre que el sistema tenga capacidad para hacer esas comprobaciones. Por ejemplo, alimentación, memoria RAM y ROM, dispositivos conectados, etc.

- *Inicialización componentes* llamará a las funciones de inicialización/configuración de los distintas partes de la aplicación (inicialiación de periféricos, de variables, etc.).

  En el ejemplo que se mostrará, se inicializará el sistema de tomas.

- *Bienvenida* da un mensaje de entrada al sistema. Típico de las aplicaciones que dispone de sistema de visualización. Pensar, por ejemplo, en el mensaje que da un móvil al encenderlo.

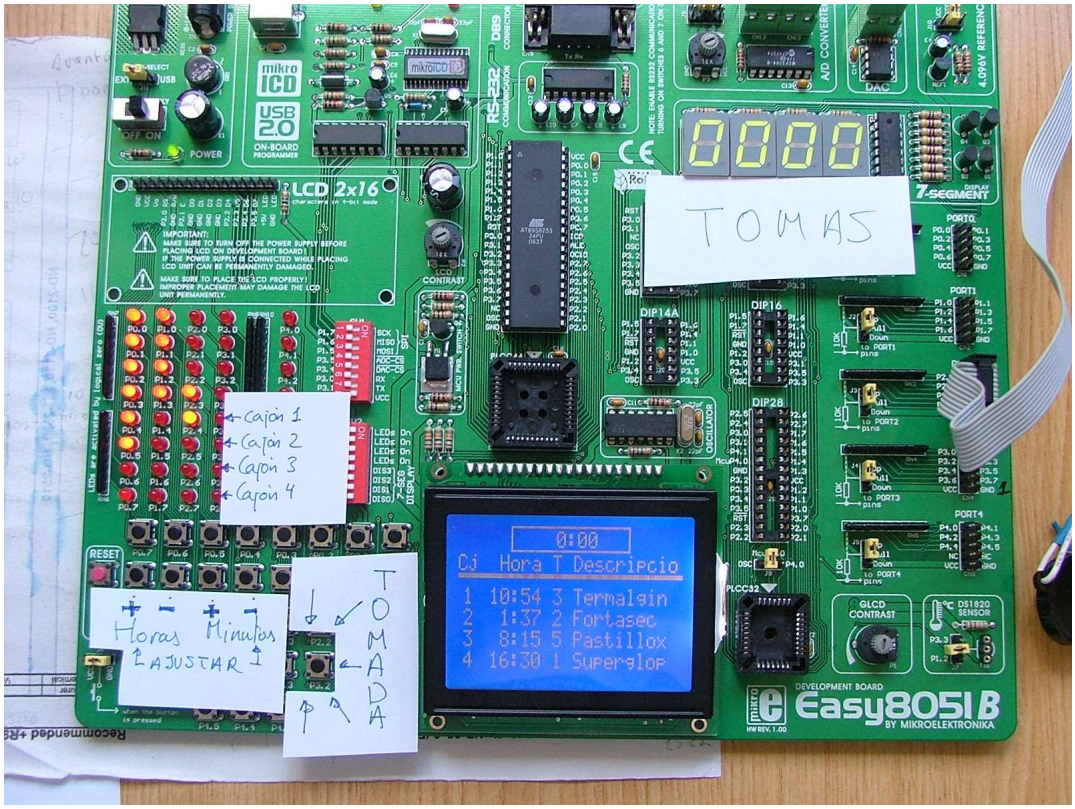
- *Tareas* es el conjunto de funcionalidades de la aplicación que se ejecutan, secuencialmente, en un bucle infinito denominado *superbucle*.

  En los casos de aplicaciones sencillas ésta es la manera usual. Cuando se complican un poco, las tareas problemáticas/críticas son realizadas mediante el servicio de interrupciones (este va a ser nuestro caso). En las ap-

licaciones más complejas se emplea un planificador de tareas que, normalmente, aporta un pequeño sistema operativo.

Este es el aspecto que tiene ahora el pastillero funcionando el la placa de prácticas.

Figure 8. Aspecto del pastillero sobre la placa de prácticas



Y aquí va el listado del módulo principal (el que contiene a main()) del pastillero bastante decente.

```
/* Fichero: main.c
   Programa principal del pastillero de Informática Industrial
   Grupo 223. Curso 2007/2008

   A destacar: En este módulo no se ve ningún microcontrolador por ninguna parte, ¿no?
   ¡Pues esa es la idea! Igual que en la primera parte, hemos dividido y ocultado y este módulo
   puede funcionar igual en un 51, en una Nintendo DX o en un cohete espacial.

   En realidad si que se vé en la interrupcion externa porque así lo hicimos en clase, pero
   eso se puede arreglar tambien
*/

#include "display7seg.h"
#include "pulsadores.h"
#include "sonido.h"
#include "indicador.h"
#include "reloj.h"
#include "canciones.h"
#include "glcd.h"
#include "medicinas.h"

#include <stdio.h>
#include <reg51.h>

// una imagen en formato xbm, el médico de los Simpson
#define char char code
#include "med_sim2.xbm"
#undef char
```

```
// prototipos de funciones que se usan antes de su definición
void mostrar_hora(void);
void pulsadores_cambio_hora(void);
void listar_medicinas(void);
void revisar_cajones(void);
void pausa(unsigned long cnt);

// contador de pastillas a tomar de un cajon
unsigned char pastillas_por_tomar;

// programa principal ------------------------------------------------------
void main(void)
{
   static unsigned int cuentas_hasta_actualizar = 0;

   pausa(10000); // para saltarse el fallo del grabador de la placa Mikroe
   //PLCD_CS1=1;PLCD_CS2=1;PLCD_E = 0; // liberar las lineas del GLCD

   // preparar todo el sistema
   glcd_inicializar();
   display7seg_inicializar();
   display7seg_refresco_off();
   pulsadores_inicializar();
   sonido_inicializar();
   indicador_inicializar();
   reloj_inicializar();
   medicinas_inicializar();

   // cuenta de tomas que se descontará usando la interupcion externa 0
   pastillas_por_tomar = 0;
   // programar interrupción externa 0
   // acordamos hacer el descontador de tomas con esto
   EA = 0;   // desconectar sistema interrupciones
   IT0 = 1;  // programar INT0 por flanco de bajada
   EX0 = 1;  // habilitar interrupción INT0
   EA = 1;   // habilitar sistema de interrupciones

   // hacer una entrada bonita
   glcd_putxbitmap(1,1,0,med_sim2);
   glcd_font_puts(&font5x7,50,22,"El pastillero");
   glcd_rectangle(52,40,122,60,1);
   glcd_rectangle(57,45,117,55,1);

   pausa(100000);

   // empezamos

   // cargar las medicinas en los cajones, un ejemplo posible
   medicinas_prescripcion_set(1,10,54, 3, "Termalgin");
   medicinas_prescripcion_set(2,1,37, 2, "Fortasec");
   medicinas_prescripcion_set(3,8,15, 5, "Pastillox");
   medicinas_prescripcion_set(4,16,30, 1, "Superglop");

   // preparar la ventana del glcd
   glcd_clear_screen();
   glcd_rectangle(30,0,127-30,10,1);
   putchar_setxy(0,12);
   printf("Cj  Hora T Descripcio");
   glcd_rectangle(0,20,127,21,1);

   // poner la lista de medicamentos
   listar_medicinas();

   // poner las tomas pendientes, 0 en principio
   display7seg_numero(pastillas_por_tomar);
   display7seg_refresco_on();

   // el "superbucle" es esto, donde el micro se queda dando vueltas
   // hasta el infinito
```

```
   while(1) {

      // gestionar el cambio de la hora
      pulsadores_cambio_hora();

   // actualizar el display de tomas
      display7seg_numero(pastillas_por_tomar);

   // tareas a hacer cada "cierto" tiempo,
      // pues hacerlo continuamente no da buen resultado (refrescos, etc.)
   if (cuentas_hasta_actualizar == 0) {
   cuentas_hasta_actualizar = 60000;
   mostrar_hora();
   // mirar si hay un cajon pendiente solo si ya me he terminado la toma
   if (pastillas_por_tomar == 0) {
         revisar_cajones();
   };
   } else {
         cuentas_hasta_actualizar--;
   }
   } /* del while(1) */
}

// hacer un retardo a base de un bucle ----------------------------------------
void pausa(unsigned long cnt) {
   while(cnt--){};
}


// servicio interrupción externa 0 --------------------------------------------
// pin P3.2/INT0
void tomas_ISR (void) interrupt 0 {

   if (pastillas_por_tomar > 0) {
      pastillas_por_tomar--;
   }
}

// comprobar si es una hora para una toma y actuar en consecuencia ------------
void revisar_cajones(void) {
   unsigned char i,j;
   unsigned char hora_toma, minuto_toma, num_tomas;
   unsigned char hora,minuto;
   unsigned char *texto;

   // ver que hora es
   reloj_wall_get(&hora,&minuto,&i); // uso "i" para segundos para ahorrarme un byte

   for (i=1;i<=NUM_CAJONES;i++) {
      medicinas_prescripcion_get(i,&hora_toma, &minuto_toma, &num_tomas, &texto);
   // ver si hay que tomárselo
   if ((num_tomas > 0) && ((hora > hora_toma)||((hora == hora_toma)&&(minuto >= minuto_toma)))) {
      // borramos la toma
   medicinas_prescripcion_set(i,0,0,0,"");
   // encendemos el indicador de cajon que toca
   for (j=1;j<=NUM_CAJONES;j++) {
      indicador_modificar(j,INDICADOR_OFF);
   }
   indicador_modificar(i,INDICADOR_ON);
   // actualizamos la lista de la pantalla
   listar_medicinas();
   // tocamos la musica
   sonido_musica_tocar(popcorn);
   // actualizamos el contador de pastillas a tomar
    pastillas_por_tomar = num_tomas;
   return; // forzamos la salida, no sea que haya otro cajon que cumpla esto
   }
   }
}
```

```
// agrupar aquí la tarea de mostrar la hora en pantalla y gestionar los pulsadores que la cambian
void pulsadores_cambio_hora(void) {

   static unsigned char ultima_tecla = PULS_NADA;
   unsigned char hora, minuto, segundo;
   unsigned char tecla;

   tecla = pulsadores_leer();

   if (tecla != ultima_tecla) {
      if (tecla != PULS_NADA) {
         reloj_wall_get(&hora,&minuto,&segundo);
      switch(tecla) {
     case PULS_HORA_MAS:
        if (hora == 23) {
           hora = 0;
        } else {
           hora ++;
        }
        break;
     case PULS_HORA_MENOS:
               if (hora > 0) {
         hora--;
        } else {
         hora = 23;
      }
      break;
     case PULS_MINUTO_MAS:
        if (minuto == 59) {
           minuto = 0;
        } else {
           minuto ++;
         }
      break;
     case PULS_MINUTO_MENOS:
               if (minuto > 0) {
         minuto--;
      } else {
         minuto = 59;
      }
         break;
       }
          reloj_wall_set(hora,minuto,0);
       mostrar_hora();
   }
   ultima_tecla = tecla;
   }
}

// mostrar la hora donde toque ------------------------------------------------
void mostrar_hora(void) {
   unsigned char hora, minuto, segundo;

   reloj_wall_get(&hora,&minuto,&segundo);
   putchar_setxy(50,2);
   display7seg_refresco_off();
   //printf("%2d:%02d:%02d", (int)hora, (int)minuto, (int)segundo);
   printf("%2d:%02d", (int)hora, (int)minuto);
   display7seg_refresco_on();
}

// listar los medicamentos en el display glcd ---------------------------------
void listar_medicinas(void) {

 unsigned char i,hora,minuto,num_tomas;
 unsigned char *texto;

 display7seg_refresco_off();
 for (i=1;i<=NUM_CAJONES;i++) {
      medicinas_prescripcion_get(i, &hora, &minuto, &num_tomas, &texto);
```

```
      putchar_setxy(3,17+i*9);
   printf("%d",(int)i);
   putchar_setxy(18,17+i*9);
      if (num_tomas == 0) {
  printf("                  ");
   } else {
      printf("%2d:%02d %d %s",(int)hora, (int)minuto, (int)num_tomas,texto);
   }
 }
 display7seg_refresco_on();
}
```

11. Notes per a mi

Escalas:

figura html pdf

metge_simpson ? 25

display 100 30

altavoz 50 10

pastillero ? 25