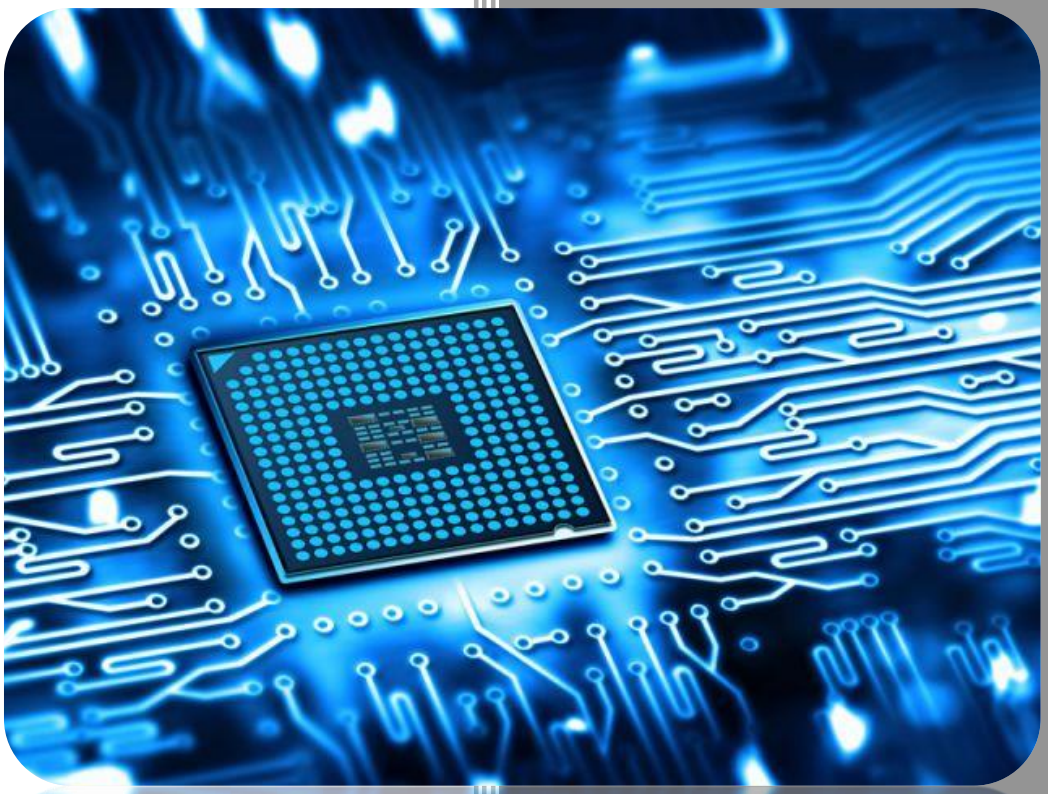
TP2

Circuitos digitales y microcontroladores



UNIVERSIDAD
NACIONAL
DE LA PLATA

Arreche, Cristian Carlos 01515/4
Blasco, Federico Matías 01678/4
20-5-2019

Índice

1. Problema

Implementar con el MCU una cerradura electrónica. Para esto se dispone de un display LCD de 2 líneas, un teclado matricial 4x4 y el HS08. La implementación deberá hacerse con máquinas de estados temporizadas con el módulo RTC.

Requerimientos:

a. Cuando el equipo se inicia deberá mostrar en la primer línea del LCD un reloj funcionando con el formato HH:MM:SS (horas, minutos y segundos) inicializado en la hora de compilación y en la segunda línea el estado de la cerradura “CERRADO”.

b. El sistema debe tener la clave numérica por defecto 2580 de manera de poder activar o desactivar la cerradura. Si el usuario presiona la clave correcta se mostrará en la segunda línea “ABIERTO” durante 5 seg y luego volverá automáticamente al estado por defecto. Si la clave es incorrecta se mostrará el estado “DENEGADO” durante 2 seg y luego volverá automáticamente al estado por defecto. No se mostrarán en el LCD las teclas presionadas.

c. Para modificar la contraseña se deberá presionar la tecla ‘D’ en el estado por defecto. Se deberá mostrar el mensaje “clave actual:” y esperar que ingrese la clave actual. Si la clave es incorrecta se mostrará el estado “DENEGADO” durante 2 seg y luego volverá automáticamente al estado por defecto. Si la clave es correcta se deberá mostrar el mensaje “nueva clave:” y leer cada tecla presionada hasta que se presione ‘D’. Se mostrará el mensaje “Fin ingreso nueva clave” 5 seg y se volverá al estado por defecto. A partir de aquí esta será la nueva clave para abrir la cerradura. Si desea cancelar se puede presionar ‘#’ lo que interrumpe el ingreso y vuelve al estado por defecto sin guardar cambios. Cuando se presione una tecla numérica se deberá mostrar ‘*’.

d. Las acciones de ingreso de claves por parte del usuario requieren un time-out de manera de cancelar la operación si la misma tarda demasiado y volver al estado por defecto. Agregar esta funcionalidad.

2. Introducción

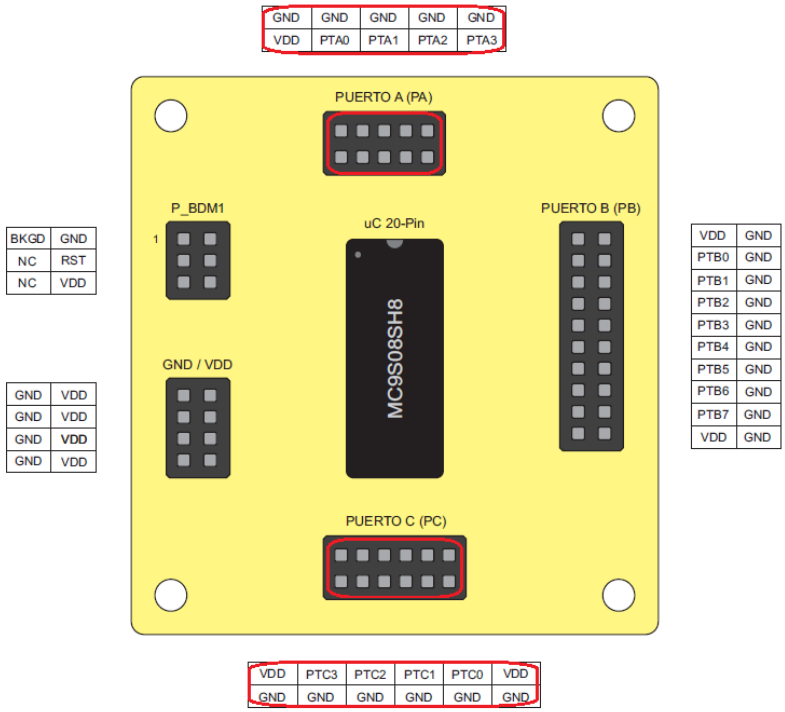
2.1. Hardware

Para la resolución del problema planteado anteriormente, contaremos con diferentes componentes de hardware, estos serán:

- Display LCD de 2 líneas, cada una de 16 caracteres.
- Teclado matricial de 4x4 caracteres.
- Microcontrolador MC9S08SH8 CPJ.

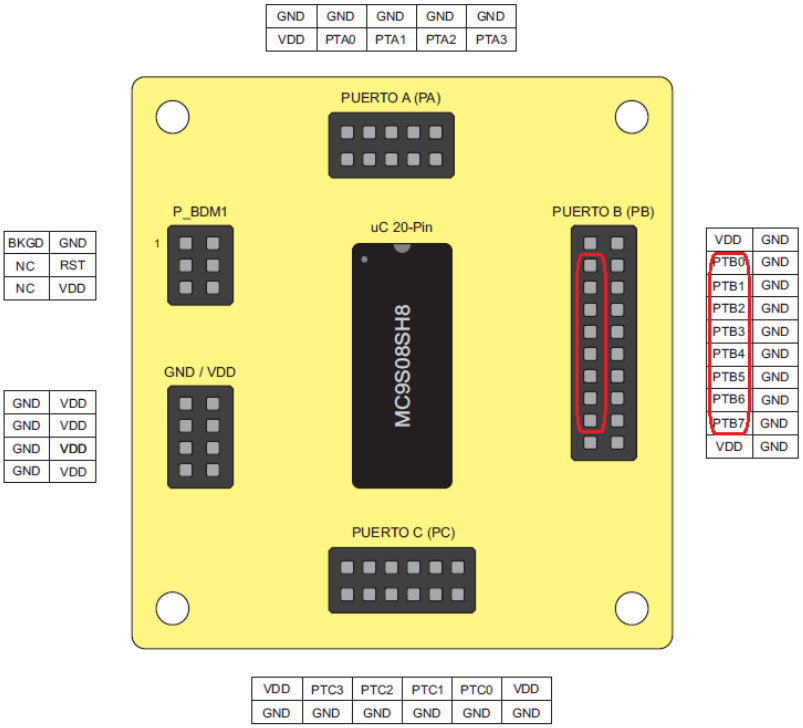
2.1.1. Conexionado display LCD

El conexionado del display se realizará tanto en el puerto A como en el puerto C de la placa, de la forma que se ve a continuación:



2.1.2. Conexionado teclado matricial

El conexionado del teclado matricial se realizará en el puerto B de la placa, como podemos ver a continuación:



2.2. Software

2.2.1. Display LCD

Para la utilización del display contamos con las librerías *lcd.c* y *lcd.h*, donde podemos encontrar una variedad de funciones que nos serán muy útiles a la hora de trabajar con el display. Algunas de ellas, y las más importantes son:

- *LCD_init (param1, param2)*, se encarga de inicializar el LCD para poder comenzar a utilizarlo, donde sus parámetros te permiten setear algunas configuraciones básicas del mismo.
- *LCD_pos_xy(posX, posY)*, como su nombre lo indica, permite escribir en la posición deseada del display.
- *LCD_write_char(carac)*, permite escribir un caracter en la posición indicada por *LCD_pos_xy()*, o por defecto, lo hará en la posición (0,0).
- *LCD_write_string(cadena)*, análogo al *LCD_write_char()*, pero nos permite escribir una cadena de caracteres.

2.2.2. Teclado matricial

A diferencia del display, no contamos con librerías para su procesamiento, por lo que la implementación interna dependerá de nosotros. Para esto se realizarán 2 funciones, donde sus objetivos serán:

- Una primera función que se encargará de hacer el barrido por la matriz de 4x4 del teclado, corroborando si fue o no presionada alguna tecla, y si así lo fuera habría que verificar cuál de ellas fue.
- Una segunda función que se encargará, cuando la función anterior detecte que una tecla fue presionada, de devolver el valor específico de dicha tecla presionada.

La implementación y resolución de estas funciones, las veremos en la sección *3. (resolución)*.

2.2.3. Reloj y time-out

Para implementar un reloj con el formato HH:MM:SS (horas, minutos y segundos), se utilizó una interrupción llamada isrVrtc, la cual es llamada por el mismo microcontrolador cada cierto período de tiempo (a determinar en la resolución).

Este período de tiempo puede ser regulado por nosotros según la “resolución” de tiempo que busquemos. En nuestro caso, como se trata de un reloj donde las unidades mínimas son los segundos, nos alcanzará con que las interrupciones sean cada 1 segundo.

Para el caso del “time-out”, el mismo consiste en limitar el tiempo de permanencia en un determinado estado sin realizar ninguna acción. De la misma manera, se hará uso de la interrupción mencionada anteriormente.

2.2.4. Máquina de estados

Para la resolución de la cerradura electrónica, fue necesario contar con una máquina de estados finita, inicialmente planteada en papel, para un claro entendimiento del problema en cuestión.

Los estados a tener en cuenta fueron determinados por el análisis del problema, donde, dependiendo de la etapa donde nos encontremos, serán las decisiones que se deberán tomar. Consideramos 6 estados diferentes, donde algunos de ellos pueden llegar a tener la misma salida.

Estos 6 estados describirán el sistema de la siguiente manera:

1) El estado por defecto, donde se mostrará el reloj funcionando y la cerradura se encontrará cerrada.
2) Ingreso de clave correcta, donde se mostrará el reloj funcionando y la cerradura pasará a estar abierta.
3) Ingrese de clave incorrecta, donde se mostrará el reloj funcionando y la cerradura notificará un acceso denegado.
4) Solicitud de cambio de clave, donde, en un primer paso, se pedirá que se ingrese la clave actual de la cerradura.
5) Cambio de clave, donde, una vez cumplido el paso anterior, se pedirá que se ingrese la nueva clave de la cerradura.
6) Fin cambio de clave, donde se notificará que la clave fue cambiada con éxito.

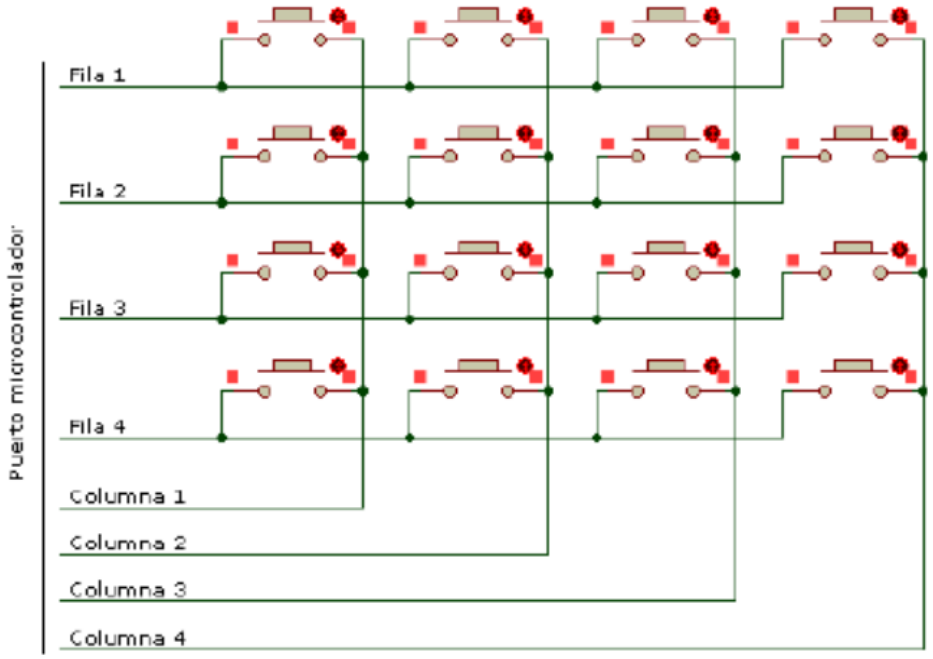
Cada uno de estos estados, tendrá distintas características específicas que, valga la redundancia, son las mencionadas en el problema.

La máquina de estados fue planteada como una máquina de moore, es decir, que la salida del sistema solo depende del estado actual.

3. Resolución

3.1. Teclado Matricial

Las filas y columnas del mismo están dispuestas de la siguiente manera:



Nuestro objetivo es detectar cuándo se presiona una tecla, y determinar qué tecla fue la presionada.

Las columnas del teclado son las entradas, configuradas con una resistencia pull-up interna que es activa en bajo (0 lógico). Por lo que, cuando se presione una tecla, será activado el bit correspondiente a la columna que pertenezca dicha tecla.

Por otro lado, las filas del teclado son las salidas, donde, para cada columna, se va induciendo un 0 a la vez hasta probar todas las combinaciones posibles.

Para la implementación del teclado, hay que tener en cuenta la conexión del mismo al puerto B (PTBD), como se ve a continuación:

C4	C3	C2	C1	F4	F3	F2	F1

Donde, como podemos observar, los 4 bits menos significativos corresponden a las salidas (filas), y los 4 bits más significativos corresponden a las entradas (columnas). De esta manera, la relación entre el valor cada una de las teclas y el valor del puerto B es la siguiente:

TECLA PRESIONADA	TECLA PRESIONADA	VALOR EN PTBD (8 bits)
1	C1F1	0b 1110 1110
2	C2F1	0b 1101 1110
3	C3F1	0b 1011 1110
A	C4F1	0b 0111 1110
4	C1F2	0b 1110 1101
5	C2F2	0b 1101 1101
6	C3F2	0b 1011 1101
B	C4F2	0b 0111 1101
7	C1F3	0b 1110 1011
...	...	...

Teniendo en cuenta todo lo mencionado anteriormente, contaremos con dos funciones, una primera función que será la encargada de hacer el barrido por las filas y columnas detectando cuándo una tecla sea presionada, y una segunda función que será la encargada de determinar qué tecla fue la que se presionó. Por lo tanto, los pseudocódigos para dichas funciones serán los siguientes:

- Función *KEYPAD_Scan(*char):*

```
Para cada fila
      Para cada columna
            Si se presionó la tecla
                  Llamar función devolverTecla
                  Retornar verdadero
Retornar falso
```

- Función devolverTecla(char):

```
 Verificar valor de la variable recibida
      Retornar la tecla que corresponda
```

A su vez, para el correcto funcionamiento del teclado, al tratarse de un teclado con el mismo problema de efecto rebote que todos los pulsadores (como ya se explicó en el TP1), se utilizará un delay de 20ms.

3.2. Reloj y time-out

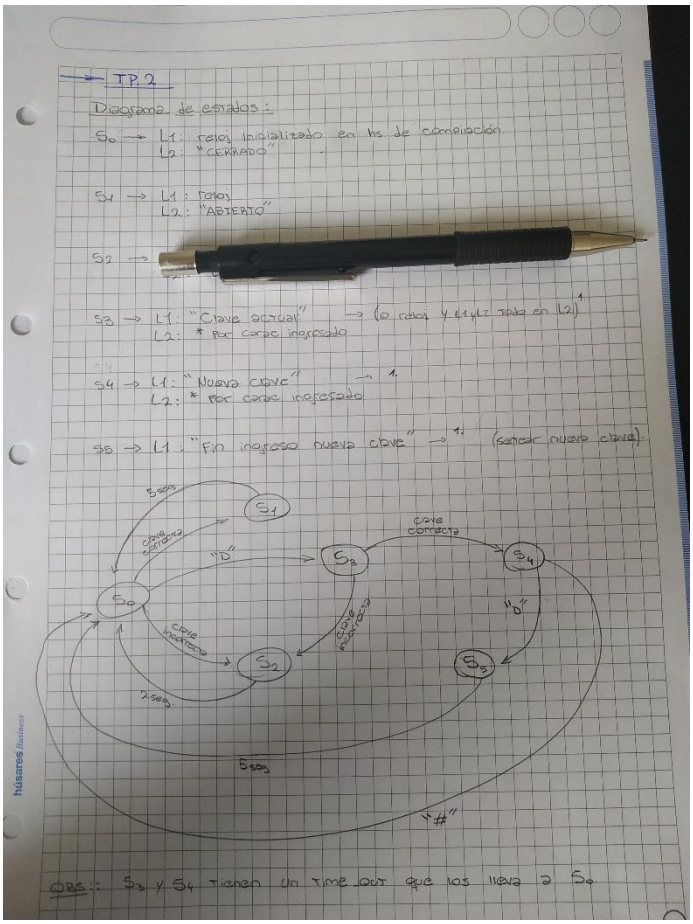
Como mencionamos en la introducción, el funcionamiento tanto del reloj como del time-out depende exclusivamente del uso de la interrupción del flag RTC. Para lograr que la interrupción se realice cada un período de tiempo de 1 segundo, se debió configurar los ajustes del reloj e interrupciones de la siguiente manera:

Name	Value
Component name	RTC1
Device	RTC
Settings	
Clock settings	
Clock select	1-kHz low power oscillator
Internal ref. clock for perip	Disabled
External ref. clock for perip	Disabled
Prescaler	100
RTC modulo value	9
Period	1 s
Interrupts	
Interrupt	Vrtc
RTC Interrupt	Enabled
ISR name	isrVrtc

Como podemos ver, la interrupción RTC se encuentra habilitada, y con los valores seteados de “Prescaler” y “RTC modulo value” se consiguió un período de interrupción de 1 segundo. La utilidad de esta interrupción, quedará más clara en la sección *3.4 (programa principal).*

3.3. Máquina de estados

La implementación para el problema de la cerradura electrónica será realizada mediante una máquina de estados temporizada con el módulo RTC mencionado anteriormente.



A su vez, para los ingresos de claves por parte del usuario es necesario tener en cuenta un time-out, es decir, se contará con determinado tiempo para poder realizar el ingreso, pasado este tiempo, automáticamente se volverá al estado por defecto sin guardar ningún tipo de cambio. El tiempo de time-out fue fijado en 15 segundos.

Para la implementación de la máquina de estados, tenemos varias funciones:

- cambiarEstado(state):
  - Se encarga de cambiar el estado al nuevo estado recibido por parámetro. Además, se encarga de volver a inicializar todas las variables en sus valores por defecto, como, por ejemplo, la variable que mantiene el tiempo de permanencia en un mismo estado (timeInState), y de limpiar la segunda línea de la pantalla LCD para no arrastrar salidas anteriores.
- comprobarClave(unsigned char):
  - Comprueba que la clave ingresada mediante el teclado matricial sea correcta, y retorna si es correcta o no, una vez ingresadas las 4 teclas. De esta manera se evita que se pueda adivinar la clave mediante prueba y error. Luego de la comprobación de clave, esta misma función es la encargada de cambiar el estado actual al estado que corresponda.
- timeOut():
  - En esta función, si el tiempo de permanencia en un mismo estado es igual al tiempo de timeout prefijado (15 segundos) se cambia del estado actual al estado por defectov(CERRADO).
- setearClave(unsigned char):
  - Esta función es utilizada para el ingreso de una nueva clave para la cerradura. Esto solo es posible presionando la tecla “D” en el estado por defecto, y seguido a esto, ingresar correctamente la clave actual de la cerradura. Se recibe por parámetro cada una de las teclas presionadas, una a la vez, y en función del valor que tenga esta tecla, realiza distintos procesamientos:
    - Si se presiona la tecla ´#´ se cancela la operación, por lo tanto, se cambia el estado actual (*PONER NOM DEL ESTADO*) al estado por defecto, ya que significa que el usuario se arrepintió de cambiar la clave.
    - Si se presiona la tecla ´D´ y la clave tiene un total de 4 caracteres, se guarda la clave y se pasa al estado “finNuevaClave”.
    - Si la tecla ingresada es válida y no se llegó al límite de 4 caracteres de largo, se ingresa en la posición correspondiente del vector de caracteres que contiene el ingreso de la nueva contraseña.
    - Si no se ingresó a ninguna de las condiciones anteriormente mencionadas, quiere decir que ocurrió algún tipo de error. Estos pueden ser que se haya excedido el largo permitido de caracteres en la clave o que se haya presionada la tecla ‘*’ (carácter inválido para la clave). En estos casos se vuelve al estado cambiarClave, para poder volver a intentarlo.
- ejecutarEstado():

- Esta función es la base del sistema y es llamada constantemente por el programa principal. Se encarga de hacer el procesamiento y el llamado a los módulos correspondientes dependiendo del estado en el que nos encontremos en esa ejecución.

  Además, la salida del LCD se ve reflejada por esta misma función, es decir, también imprime en el display el texto que corresponda según el estado actual.

3.4. Programa principal

Todo el código utilizado para la resolución de este problema, fue modularizado de la mejor manera posible, lo cual nos permite tener un programa principal corto y entendible. El pseudocódigo para el programa principal es el siguiente:

- Programa principal

```
Inicializar variables, puertos y periféricos
      Si se disparó  la interrupción
            Incrementar el tiempo en el estado
            Incrementar en un segundo el reloj
      Hacer procesamiento según el estado actual
      Chequear si llegó el timeout del estado
```

4. Anexo

4.1. Main.c

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "stdio.h"
#include "lcd.h"


#ifdef __cplusplus
extern "C"
#endif

typedef
enum{abierto,cerrado,denegado,cambiarClave,nuevaClave,finNuevaC
lave}state;
#define TIMEOUT 15

extern volatile unsigned char RTC_flag;//Extern porque se
inicializa en el init, volatile por ser global

void MCU_init(void); /* Device initialization function
declaration */

//Inicializacion de variables globales para el timer
unsigned char dig1='1';
unsigned char dig2='6';
unsigned char dig3='5';
unsigned char dig4='1';
unsigned char dig5='0';
unsigned char dig6='0';
unsigned char seg=0;
unsigned char min=51;
```

```
unsigned char hora=16;
int cont=0;
state estado=cerrado;
unsigned char pass[] = "2580";
unsigned char passAux[] = "    ";
unsigned short dimL = 4;
unsigned short indice = 0;
unsigned short indice2 = 0;
unsigned char probando;
unsigned short timeInState = 0;
unsigned short cond = 1;


////////////////////////////////////////////////////////////////
//////////////////////////////////////
void reloj(void){
    if(cont == 9){//Para el reloj
      seg++;
      dig5=(seg/10)+'0';
      dig6=(seg%10)+'0';
      cont=0;
      timeInState++;
      if(seg!=60){
          LCD_pos_xy(6,0);
          LCD_write_char(dig5);
          LCD_write_char(dig6);
      }
      else{
          LCD_pos_xy(6,0);
          LCD_write_char(dig5);
          LCD_write_char(dig6);
          min++;
          seg=0;
          dig3=(min/10)+'0';
          dig4=(min%10)+'0';
          if(min != 60){
              LCD_pos_xy(3,0);
              LCD_write_char(dig3);
              LCD_write_char(dig4);
          }
          else{
              min=0;
              hora++;
              dig1=(hora/10)+'0';
              dig2=(hora%10)+'0';
              if(hora!=24){
                  LCD_pos_xy(0,0);
                  LCD_write_char(dig1);
                  LCD_write_char(dig2);
              }
              else{
                  hora=0;
              }
          }
      }
    }
}

////////////////////////////////////////////////////////////////
//////////////////////////////////////
```

```
void delay(unsigned short n){//Funcion que recibe por par□tro
la cantidad de miliseg de delay
  unsigned short temp;
  unsigned short cant;
  /*Aca sabemos que 1a ejecucion del for equivale a 23 ciclos
de reloj
  y que hacemos 8000 ciclos por milisegundo, eso equivale a mas
o menos
  348 ejecuciones del for por milisegundo*/
  for(cant=n; cant>0; cant--){
    for(temp=348; temp>0; temp--);
  }
}

////////////////////////////////////////////////////////////////
////////////////////////////////////
char devolverTecla(char keyPressed){
    switch(keyPressed){
      case 0b01110111:
          return 'D';
          break;
      case 0b01111011:
              return 'C';
              break;
      case 0b01111101:
              return 'B';
              break;
      case 0b01111110:
              return 'A';
              break;
      case 0b10110111:
              return '#';
              break;
      case 0b10111011:
              return '9';
              break;
      case 0b10111101:
              return '6';
              break;
      case 0b10111110:
              return '3';
              break;
      case 0b11010111:
              return '0';
              break;
      case 0b11011011:
              return '8';
              break;
      case 0b11011101:
              return '5';
              break;
      case 0b11011110:
              return '2';
              break;
      case 0b11100111:
              return '*';
              break;
      case 0b11101011:
              return '7';
              break;
      case 0b11101101:
```

```
                return '4';
                break;
        case 0b11101110:
                return '1';
                break;
    }
}

////////////////////////////////////////////////////////////////////
////////////////////////////////////
char KEYPAD_Scan(char * key){
    unsigned short fila_act;
    unsigned short col_act;
    char f = 0b00000001;
    char c = 0b00000001;
    char cAux;
    char Aux;
    unsigned char PTBDAux = 0b00000001;//C4C3C2C1F4F3F2F1

    for(fila_act=0;fila_act<4;fila_act++){
        PTBD = ~(PTBDAux<<fila_act);
        for(col_act=0;col_act<4;col_act++){
            cAux = c<<4;
            Aux = cAux|f;
            Aux = ~Aux;
            if(PTBD == Aux){
                (*key) = devolverTecla(Aux);
                while(PTBD==Aux);//Para que no salga hasta que
no se suelte la tecla
                return(1);
            }
            c = c<<1;
        }
        f = f<<1;
        c = 0b00000001;
    }
    return(0);
}


////////////////////////////////////////////////////////////////////
////////////////////////////////////
void cambiarEstado(state estadoNuevo){
    estado=estadoNuevo;
    timeInState=0;
    indice=0;
    indice2=0;
    LCD_pos_xy(0,1);
    LCD_write_string("                ");
    cond = 1;
}

////////////////////////////////////////////////////////////////////
////////////////////////////////////
void comprobarClave(unsigned char tecla){
    probando=pass[indice];
    if (probando!=tecla){
        cond=0;
    }
    if((cond)&&(indice!=(dimL-1))){//Si le pega a la tecla, y
el indice sigue siendo valido
```

```
            indice++;
        }else{
            if(!cond){
                if(indice == 3){
                    cambiarEstado(denegado);
                    LCD_pos_xy(0,1);
                    LCD_write_char(tecla);
                }
                indice++;

            }else{
                if(estado==cerrado){
                    cambiarEstado(abierto);
                }else{//Estoy en el estado cambiar clave
                    cambiarEstado(nuevaClave);
                }
            }
        }
}


////////////////////////////////////////////////////////////////
////////////////////////////////////
void timeOut(){
    if(timeInState == TIMEOUT){
        cambiarEstado(cerrado);
    }
}


////////////////////////////////////////////////////////////////
////////////////////////////////////
void setearClave(unsigned char tecla){
    if(tecla == '#'){
        cambiarEstado(cerrado);
    }else{
        if(tecla == 'D' && indice2==4){
            pass[0]=passAux[0];
            pass[1]=passAux[1];
            pass[2]=passAux[2];
            pass[3]=passAux[3];
            cambiarEstado(finNuevaClave);
        }
        else{
            if( (tecla!='*') && (indice2 != 4) ){
                passAux[indice2]=tecla;
                indice2++;
            }else{
                LCD_pos_xy(0,1);
                cambiarEstado(cambiarClave);
            }
        }

    }
}


////////////////////////////////////////////////////////////////
////////////////////////////////////
void ejecutarEstado(){
    unsigned char tecla;
```

```
    switch(estado){
        case cerrado:
            if(KEYPAD_Scan(&tecla)){
                if(tecla=='D'){
                    cambiarEstado(cambiarClave);
                }else{
                    comprobarClave(tecla);
                    delay(20);
                }
            }
            LCD_pos_xy(0,1);
            LCD_write_string("CERRADO          ");
            break;
        case abierto:
            LCD_pos_xy(0,1);
            LCD_write_string("ABIERTO          ");
            if(timeInState == 5){
                cambiarEstado(cerrado);
            }
            break;
        case denegado:
            LCD_pos_xy(0,1);
            LCD_write_string("DENEGADO        ");
            if(timeInState == 2){
                cambiarEstado(cerrado);
            }
            break;
        case cambiarClave:
            //ejecutarCambiarClave();
            LCD_pos_xy(0,1);
            LCD_write_string("Clave act:");
            if(KEYPAD_Scan(&tecla)){
                LCD_pos_xy(indice+10,1);
                LCD_write_char('*');
                comprobarClave(tecla);
                delay(20);
            }
            break;
        case nuevaClave:
            //ejecutarNuevaClave();
            LCD_pos_xy(0,1);
            LCD_write_string("Nueva clave:");
            if(KEYPAD_Scan(&tecla)){
                setearClave(tecla);
                LCD_pos_xy(indice2+11,1);
                LCD_write_char('*');
            }
            break;
        case finNuevaClave:
            //ejecutarFinNuevaClave();
            LCD_pos_xy(0,1);
            LCD_write_string("FIN NUEVA CLAVE");
            if(timeInState == 5){
                cambiarEstado(cerrado);
            }
            break;
    }
}
```

```
////////////////////////////////////////////////////////////////
////////////////////////////////////
////////////////////////////////////////////////////////////////
////////////////////////////////////

void main(void) {

  PTBDD = 0x0F;
  PTBPE = 0xF0;//Seteo la entrada PTB con una resistencia pull
up interna

  MCU_init();   /* call Device Initialization */
  LCD_init(_2_LINES|DISPLAY_8X5,DISPLAY_ON|CURSOR_ON);

  LCD_pos_xy(0,0);
  LCD_write_char(dig1);
  LCD_write_char(dig2);
  LCD_write_char(':');
  LCD_write_char(dig3);
  LCD_write_char(dig4);
  LCD_write_char(':');
  LCD_write_char(dig5);
  LCD_write_char(dig6);



  for(;;) {

      if(RTC_flag == 1){
          cont++;
          RTC_flag=0;
          reloj();
      }//Termina el chequeo de la hora
      ejecutarEstado();
      timeOut();
      }
}
```

4.2. MCUinit.c

```
/*
**
###################################################################
####
**     This code is generated by the Device Initialization
Tool.
**     It is overwritten during code generation.
**     USER MODIFICATION ARE PRESERVED ONLY INSIDE INTERRUPT
SERVICE ROUTINES
**     OR EXPLICITLY MARKED SECTIONS
**
**     Project   : DeviceInitialization
**     Processor : MC9S08SH8CPJ
**     Version   : Component 01.008, Driver 01.08, CPU db:
3.00.066
**     Datasheet : MC9S08SH8 Rev. 3 6/2008
**     Date/Time : 2019-04-29, 15:06, # CodeGen: 1
**     Abstract  :
**         This module contains device initialization code
**         for selected on-chip peripherals.
```

```
**     Contents  :
**         Function "MCU_init" initializes selected peripherals
**
**     Copyright : 1997 - 2010 Freescale Semiconductor, Inc.
All Rights Reserved.
**
**     http      : www.freescale.com
**     mail      : support@freescale.com
**
###################################################################
####
*/

/* MODULE MCUinit */

#include <mc9s08sh8.h>                 /* I/O map for
MC9S08SH8CPJ */
#include "MCUinit.h"

/* Standard ANSI C types */
#ifndef int8_t
typedef signed char int8_t;
#endif
#ifndef int16_t
typedef signed int int16_t;
#endif
#ifndef int32_t
typedef signed long int int32_t;
#endif

#ifndef uint8_t
typedef unsigned char uint8_t;
#endif
#ifndef uint16_t
typedef unsigned int uint16_t;
#endif
#ifndef uint32_t
typedef unsigned long int uint32_t;
#endif

/* User declarations and definitions */
/*   Code, declarations and definitions here will be preserved
during code generation */
volatile unsigned char RTC_flag=0;
/* End of user declarations and definitions */


/*
**
===================================================================
====
**     Method      :  MCU_init (component MC9S08SH8_20)
**
**     Description :
**         Device initialization code for selected peripherals.
**
===================================================================
====
*/
void MCU_init(void)
{
```

```
  /* ### MC9S08SH8_20 "Cpu" init code ... */
  /*  PE initialization code after reset */
  /* Common initialization of the write once registers */
  /* SOPT1: COPT=0,STOPE=0,IICPS=0,BKGDPE=1,RSTPE=0 */
  SOPT1 = 0x02U;
  /* SPMSC1:
LVWF=0,LVWACK=0,LVWIE=0,LVDRE=1,LVDSE=1,LVDE=1,BGBE=0 */
  SPMSC1 = 0x1CU;
  /* SPMSC2: LVDV=0,LVWV=0,PPDF=0,PPDACK=0,PPDC=0 */
  SPMSC2 = 0x00U;
  /*  System clock initialization */
  /*lint -save  -e923 Disable MISRA rule (11.3) checking. */
  if (*(unsigned char*far)0xFFAFU != 0xFFU) { /* Test if the
device trim value is stored on the specified address */
    ICSTRM = *(unsigned char*far)0xFFAFU; /* Initialize ICSTRM
register from a non volatile memory */
    ICSSC = *(unsigned char*far)0xFFAEU; /* Initialize ICSSC
register from a non volatile memory */
  }
  /*lint -restore Enable MISRA rule (11.3) checking. */
  /* ICSC1: CLKS=0,RDIV=0,IREFS=1,IRCLKEN=0,IREFSTEN=0 */
  ICSC1 = 0x04U;                       /* Initialization of the
ICS control register 1 */
  /* ICSC2:
BDIV=1,RANGE=0,HGO=0,LP=0,EREFS=0,ERCLKEN=0,EREFSTEN=0 */
  ICSC2 = 0x40U;                       /* Initialization of the
ICS control register 2 */
  while(ICSSC_IREFST == 0U) {          /* Wait until the source
of reference clock is internal clock */
  }
  /* GNGC:
GNGPS7=0,GNGPS6=0,GNGPS5=0,GNGPS4=0,GNGPS3=0,GNGPS2=0,GNGPS1=0,
GNGEN=0 */
  GNGC = 0x00U;
  /* Common initialization of the CPU registers */
  /* PTASE: PTASE4=0,PTASE3=0,PTASE2=0,PTASE1=0,PTASE0=0 */
  PTASE &= (unsigned char)~(unsigned char)0x1FU;
  /* PTBSE:
PTBSE7=0,PTBSE6=0,PTBSE5=0,PTBSE4=0,PTBSE3=0,PTBSE2=0,PTBSE1=0,
PTBSE0=0 */
  PTBSE = 0x00U;
  /* PTCSE: PTCSE3=0,PTCSE2=0,PTCSE1=0,PTCSE0=0 */
  PTCSE &= (unsigned char)~(unsigned char)0x0FU;
  /* PTADS: PTADS4=0,PTADS3=0,PTADS2=0,PTADS1=0,PTADS0=0 */
  PTADS = 0x00U;
  /* PTBDS:
PTBDS7=0,PTBDS6=0,PTBDS5=0,PTBDS4=0,PTBDS3=0,PTBDS2=0,PTBDS1=0,
PTBDS0=0 */
  PTBDS = 0x00U;
  /* PTCDS: PTCDS3=0,PTCDS2=0,PTCDS1=0,PTCDS0=0 */
  PTCDS = 0x00U;
  /* ### Init_RTC init code */
  /* RTCMOD: RTCMOD=9 */
  RTCMOD = 0x09U;                      /* Set modulo register
*/
  /* RTCSC: RTIF=1,RTCLKS=0,RTIE=1,RTCPS=0x0B */
  RTCSC = 0x9BU;                       /* Configure RTC */
  /* ### */
  /*lint -save  -e950 Disable MISRA rule (1.1) checking. */
  asm CLI;                             /* Enable interrupts */
  /*lint -restore Enable MISRA rule (1.1) checking. */
```

```
} /*MCU_init*/


/*lint -save  -e765 Disable MISRA rule (8.10) checking. */
/*
**
============================================================
====
**      Interrupt handler : isrVrtc
**
**      Description :
**          User interrupt service routine.
**      Parameters  : None
**      Returns     : Nothing
**
============================================================
====
*/
__interrupt void isrVrtc(void)
{
  /* Write your interrupt code here ... */
  RTC_flag=1;//Se cumplio el plazo de 100ms
  RTCSC_RTIF=1;//Se borra en un solo paso
}
/* end of isrVrtc */


/*lint -restore Enable MISRA rule (8.10) checking. */

/*lint -save  -e950 Disable MISRA rule (1.1) checking. */
/* Initialization of the CPU registers in FLASH */
/* NVPROT:
FPS7=1,FPS6=1,FPS5=1,FPS4=1,FPS3=1,FPS2=1,FPS1=1,FPDIS=1 */
static const unsigned char NVPROT_INIT @0x0000FFBDU = 0xFFU;
/* NVOPT: KEYEN=0,FNORED=1,SEC01=1,SEC00=0 */
static const unsigned char NVOPT_INIT @0x0000FFBFU = 0x7EU;
/*lint -restore Enable MISRA rule (1.1) checking. */



extern near void _Startup(void);

/* Interrupt vector table */
#ifndef UNASSIGNED_ISR
  #define UNASSIGNED_ISR ((void(*near const)(void)) 0xFFFF) /*
unassigned interrupt service routine */
#endif

/*lint -save  -e923 Disable MISRA rule (11.3) checking. */
/*lint -save  -e950 Disable MISRA rule (1.1) checking. */
static void (* near const _vect[])(void) @0xFFC0 = { /*
Interrupt vector table */
/*lint -restore Enable MISRA rule (1.1) checking. */
         UNASSIGNED_ISR,               /* Int.no. 31
VReserved31 (at FFC0)          Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 30 Vacmp (at
FFC2)                 Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 29
VReserved29 (at FFC4)          Unassigned */
```

```
        UNASSIGNED_ISR,                 /* Int.no. 28
VReserved28 (at FFC6)           Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 27
VReserved27 (at FFC8)           Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 26 Vmtim (at
FFCA)                Unassigned */
        isrVrtc,                        /* Int.no. 25 Vrtc (at
FFCC)                  Used */
        UNASSIGNED_ISR,                 /* Int.no. 24 Viic (at
FFCE)                Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 23 Vadc (at
FFD0)                Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 22
VReserved22 (at FFD2)           Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 21 Vportb (at
FFD4)                Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 20 Vporta (at
FFD6)                Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 19
VReserved19 (at FFD8)           Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 18 Vscitx (at
FFDA)                Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 17 Vscirx (at
FFDC)                Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 16 Vscierr
(at FFDE)               Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 15 Vspi (at
FFE0)                Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 14 Vtpm2ovf
(at FFE2)              Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 13 Vtpm2ch1
(at FFE4)              Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 12 Vtpm2ch0
(at FFE6)              Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 11 Vtpm1ovf
(at FFE8)              Unassigned */
        UNASSIGNED_ISR,                 /* Int.no. 10
VReserved10 (at FFEA)           Unassigned */
        UNASSIGNED_ISR,                 /* Int.no.  9 VReserved9
(at FFEC)            Unassigned */
        UNASSIGNED_ISR,                 /* Int.no.  8 VReserved8
(at FFEE)            Unassigned */
        UNASSIGNED_ISR,                 /* Int.no.  7 VReserved7
(at FFF0)            Unassigned */
        UNASSIGNED_ISR,                 /* Int.no.  6 Vtpm1ch1
(at FFF2)              Unassigned */
        UNASSIGNED_ISR,                 /* Int.no.  5 Vtpm1ch0
(at FFF4)              Unassigned */
        UNASSIGNED_ISR,                 /* Int.no.  4 VReserved4
(at FFF6)            Unassigned */
        UNASSIGNED_ISR,                 /* Int.no.  3 Vlvd (at
FFF8)                Unassigned */
        UNASSIGNED_ISR,                 /* Int.no.  2 Virq (at
FFFA)                Unassigned */
        UNASSIGNED_ISR,                 /* Int.no.  1 Vswi (at
FFFC)                Unassigned */
        _Startup                        /* Int.no.  0 Vreset (at
FFFE)              Reset vector */
};
/*lint -restore Enable MISRA rule (11.3) checking. */
```

```
/* END MCUinit */

/*
**
###################################################################
####
**
**     This file was created by Processor Expert 5.00 [04.48]
**     for the Freescale HCS08 series of microcontrollers.
**
**
###################################################################
####
*/
```

4.3. MCUinit.h

```
/*
**
###################################################################
####
**     This code is generated by the Device Initialization
Tool.
**     It is overwritten during code generation.
**     USER MODIFICATION ARE PRESERVED ONLY INSIDE EXPLICITLY
MARKED SECTIONS.
**
**     Project   : DeviceInitialization
**     Processor : MC9S08SH8CPJ
**     Version   : Component 01.008, Driver 01.08, CPU db:
3.00.066
**     Datasheet : MC9S08SH8 Rev. 3 6/2008
**     Date/Time : 2019-04-29, 15:06, # CodeGen: 1
**     Abstract  :
**         This module contains device initialization code
**         for selected on-chip peripherals.
**     Contents  :
**         Function "MCU_init" initializes selected peripherals
**
**     Copyright : 1997 - 2010 Freescale Semiconductor, Inc.
All Rights Reserved.
**
**     http      : www.freescale.com
**     mail      : support@freescale.com
**
###################################################################
####
*/

#ifndef __DeviceInitialization_H
#define __DeviceInitialization_H 1

/* Include shared modules, which are used for whole project */


/* User declarations and definitions */
```

```
/*   Code, declarations and definitions here will be preserved
during code generation */
/* End of user declarations and definitions */

#ifdef __cplusplus
extern "C" {
#endif
extern void MCU_init(void);
#ifdef __cplusplus
}
#endif
/*
**
===================================================================
====
**     Method      :  MCU_init (component MC9S08SH8_20)
**
**     Description :
**         Device initialization code for selected peripherals.
**
===================================================================
====
*/



/*lint -save  -e765 Disable MISRA rule (8.10) checking. */
__interrupt void isrVrtc(void);
/*
**
===================================================================
====
**     Interrupt handler : isrVrtc
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
**
===================================================================
====
*/




/*lint -restore Enable MISRA rule (8.10) checking. */

/* END DeviceInitialization */

#endif
/*
**
###################################################################
####
**
**     This file was created by Processor Expert 5.00 [04.48]
**     for the Freescale HCS08 series of microcontrollers.
**
```

```
**
####################################################################
####
*/
```

4.4. lcd.c

```
/*
 * lcd.c
 *
 *  Created on: 30/07/2013
 *      Author: pc2
 */

//* Basic Character LCD functions
//* By Foo Pereira
//* 01/15/08
//*****************************************************************
**************/
#include "lcd.h"
#include "derivative.h" /* include peripheral declarations */

union ubyte
{
    char _byte;
    struct
    {
        unsigned char b0 : 1;
        unsigned char b1 : 1;
        unsigned char b2 : 1;
        unsigned char b3 : 1;
        unsigned char b4 : 1;
        unsigned char b5 : 1;
        unsigned char b6 : 1;
        unsigned char b7 : 1;
    } bit;
};

// Display configuration global variable
static char lcd_mode;

//*****************************************************************
*************
//* A simple delay function (used by LCD functions)
//*****************************************************************
*************
//* Calling arguments
//* unsigned char time: aproximate delay time in microseconds
//*****************************************************************
*************
void LCD_delay_ms (unsigned char time)
{
    unsigned int temp;
    for(;time;time--) for(temp=(BUS_CLOCK/23);temp;temp--);
}

//*****************************************************************
*************
//* Send a nibble to the LCD
```

```
//*************************************************************
*************
//* Calling arguments
//* char data : data to be sent to the display
//*************************************************************
*************
void LCD_send_nibble(char data)
{
    union ubyte my_union;
    my_union._byte = data;
    // Output the four data bits
    LCD_D4 = my_union.bit.b0;
    LCD_D5 = my_union.bit.b1;
    LCD_D6 = my_union.bit.b2;
    LCD_D7 = my_union.bit.b3;
    // pulse the LCD enable line
    LCD_ENABLE = 1;
    for (data=20; data; data--);
    LCD_ENABLE = 0;
}

//*************************************************************
*************
//* Write a byte into the LCD
//*************************************************************
*************
//* Calling arguments:
//* char address : 0 for instructions, 1 for data
//* char data : command or data to be written
//*************************************************************
*************
void LCD_send_byte(char address, char data)
{
  unsigned int temp;
    if(address==0)
        LCD_RS = 0;               // config the R/S line
    else
        LCD_RS = 1;
    LCD_ENABLE = 0;                 // set LCD enable line to 0
    LCD_send_nibble(data >> 4);     // send the higher nibble
    LCD_send_nibble(data & 0x0f);   // send the lower nibble
    for (temp=1000; temp; temp--);
}

//*************************************************************
*************
//* LCD initialization
//*************************************************************
*************
//* Calling arguments:
//* char mode1 : display mode (number of lines and character
size)
//* char mode2 : display mode (cursor and display state)
//*************************************************************
*************
void LCD_init(char mode1, char mode2)
{
    char aux;
    // Configure the pins as outputs
    LCD_ENABLE_DIR = 1;
    LCD_RS_DIR = 1;
```

```
    LCD_D4_DIR = 1;
    LCD_D5_DIR = 1;
    LCD_D6_DIR = 1;
    LCD_D7_DIR = 1;
    // Set the LCD data pins to zero
    LCD_D4 = 0;
    LCD_D5 = 0;
    LCD_D6 = 0;
    LCD_D7 = 0;
    LCD_RS = 0;
    LCD_ENABLE = 0;       // LCD enable = 0

    LCD_delay_ms(15);
    // LCD 4-bit mode initialization sequence
    // send three times 0x03 and then 0x02 to finish
configuring the LCD
    for(aux=0;aux<3;++aux)
    {
      LCD_send_nibble(3);
      LCD_delay_ms(5);
    }
    LCD_send_nibble(2);
    // Now send the LCD configuration data
    LCD_send_byte(0,0x20 | mode1);
    LCD_send_byte(0,0x08 | mode2);
    lcd_mode = 0x08 | mode2;

    LCD_send_byte(0,1);
    LCD_delay_ms(5);
    LCD_send_byte(0,6); //entry mode set I/D=1 S=0
}

//************************************************************
*************
//* LCD cursor position set
//************************************************************
*************
//* Calling arguments:
//* char x : column (starting by 0)
//* char y : line (0 or 1)
//************************************************************
*************
void LCD_pos_xy(char x, char y)
{
  char address;
  if (y) address = LCD_SEC_LINE; else address = 0;
  address += x;
  LCD_send_byte(0,0x80|address);
}

//************************************************************
*************
//* Write a character on the display
//************************************************************
*************
//* Calling arguments:
//* char c : character to be written
//************************************************************
*************
//* Notes :
//* \f clear the display
```

```
//* \n and \r return the cursor to line 1 column 0
//*************************************************************
*************
void LCD_write_char(char c)
{
  switch (c)
    {
      case '\f' :
        LCD_send_byte(0,1);
        LCD_delay_ms(5);
        break;
      case '\n' :
      case '\r' :
        LCD_pos_xy(0,1);
        break;
      default:
        LCD_send_byte(1,c);
   }
}

//*************************************************************
*************
//* Write a string on the display
//*************************************************************
*************
//* Calling arguments:
//* char *c : pointer to the string
//*************************************************************
*************
void LCD_write_string (char *c)
{
    while (*c)
    {
      LCD_write_char(*c);
      c++;
    }
}

//*************************************************************
*************
//* Turn the display on
//*************************************************************
*************
void LCD_display_on(void)
{
    lcd_mode |= 4;
    LCD_send_byte (0,lcd_mode);
}

//*************************************************************
*************
//* Turn the display off
//*************************************************************
*************
void LCD_display_off(void)
{
    lcd_mode &= 0b11111011;
    LCD_send_byte (0,lcd_mode);
}
```

```
//*************************************************************
*************
//* Turn the cursor on
//*************************************************************
*************
void LCD_cursor_on(void)
{
  lcd_mode |= 2;
    LCD_send_byte (0,lcd_mode);
}

//*************************************************************
*************
//* Turn the cursor off
//*************************************************************
*************
void LCD_cursor_off(void)
{
    lcd_mode &= 0b11111101;
    LCD_send_byte (0,lcd_mode);
}

//*************************************************************
*************
//* Turn on the cursor blink function
//*************************************************************
*************
void LCD_cursor_blink_on(void)
{
    lcd_mode |= 1;
    LCD_send_byte (0,lcd_mode);
}

//*************************************************************
*************
//* Turn off the cursor blink function
//*************************************************************
*************
void LCD_cursor_blink_off(void)
{
    lcd_mode &= 0b11111110;
    LCD_send_byte (0,lcd_mode);
}
```

4.5. lcd.h

```
/*
 * lcd.h
 *
 *  Created on: 30/07/2013
 *      Author: pc2
 */

#ifndef LCD_H_
#define LCD_H_


// The following defines set the default pins for the LCD
display
#ifndef LCD_ENABLE           // if lcd_enable is not defined
```

```
#define LCD_ENABLE      PTCD_PTCD3   // LCD enable pin on PTC3
#define LCD_ENABLE_DIR  PTCDD_PTCDD3     // LCD enable pin
direction

#define LCD_RS          PTCD_PTCD2   // LCD r/s pin on PTB2
#define LCD_RS_DIR      PTCDD_PTCDD2     // LCD r/s pin
direction

#define LCD_D4          PTAD_PTAD0    // LCD data D4 pin
#define LCD_D4_DIR      PTADD_PTADD0  // LCD data D4 pin
direction

#define LCD_D5          PTAD_PTAD1    // LCD data D5 pin
#define LCD_D5_DIR      PTADD_PTADD1  // LCD data D5 pin
direction

#define LCD_D6          PTAD_PTAD2    // LCD data D6 pin
#define LCD_D6_DIR      PTADD_PTADD2  // LCD data D6 pin
direction

#define LCD_D7          PTAD_PTAD3    // LCD data D7 pin
#define LCD_D7_DIR      PTADD_PTADD3  // LCD data D7 pin
direction

#endif

#define LCD_SEC_LINE    0x40    // Address of the second line
of the LCD

// LCD configuration constants
#define CURSOR_ON       2
#define CURSOR_OFF      0
#define CURSOR_BLINK    1
#define CURSOR_NOBLINK  0
#define DISPLAY_ON      4
#define DISPLAY_OFF     0
#define DISPLAY_8X5     0
#define DISPLAY_10X5    4
#define _2_LINES        8
#define _1_LINE         0


//************************************************************
*************
//* Prototypes
//************************************************************
*************
void LCD_delay_ms (unsigned char time);
void LCD_send_nibble(char data);
void LCD_send_byte(char address, char data);
void LCD_init(char mode1, char mode2);
void LCD_pos_xy(char x, char y);
void LCD_write_char(char c);
void LCD_write_string (char *c);
void LCD_display_on(void);
void LCD_display_off(void);
void LCD_cursor_on(void);
void LCD_cursor_off(void);
void LCD_cursor_blink_on(void);
void LCD_cursor_blink_off(void);
```

```
#define BUS_CLOCK 8000

#endif /* LCD_H_ */
```