Trabajo Final:

Arquitectura de Sistemas de Elaboración de Datos II

Carrera: Ingeniería en Informática

Comisión: Sábados de 13:30 a 17:30 (2Q 2016)

Docente: Ing. Federico Gabriel D´Angiolo

Integrantes:

*Calonge Federico.

*Cabot Lucas.

*Fernández Alejandro.

Índice

1-Introducción y objetivo.

Este trabajo final consiste en desarrollar un sistema de seguridad el cual se implementará para una caja fuerte. El mismo contará con:

- Un servo-motor que funciona como cerradura para la caja fuerte.
- Un sistema de alarma (la cual sonará mediante un zumbador/buzzer si la contraseña fue 3 veces mal ingresada).
- Un teclado compuesto de varios pulsadores (de esta manera ingresará el usuario la clave).
- Un display LCD donde se mostrarán los **** que representa la clave ingresada y será la interfaz por la cual el usuario podrá interactuar con el sistema.
- Un RESET oculto que se debe presionar para REactivar el sistema (más abajo esto se explicará).

Todo este circuito (el **hardware**) es "controlado" por un **microcontrolador PIC**, al cual lo programamos (mediante **software**) en un lenguaje de programación de ALTO nivel, C; mediante el uso de XC8 para MPLAB. MPLAB es nuestro entorno de desarrollo, en el cual el programa que cargamos ahí (ver ***Apéndice***), lo transferimos de la PC al PIC mediante el uso de un programador (Pickit 2). De esta manera, cargamos nuestro programa al uC para poder ingresar los datos, compararlos, prender los leds, etc.

Para este trabajo, utilizamos el PIC16F877A. Elegimos este PIC principalmente por su gran número de pines (40) y por su barata comercialización. Necesitábamos muchos pines para las conexiones con el teclado y el LCD.

Para el sistema de cierre, utilizamos un mini servo SG90, el cual se moverá 90 grados en un sentido para cerrar y 90 grados en sentido contrario para abrir, este servo es controlado mediante PWM. Para controlarlo, basta con variar el tiempo que la señal permanece en alto nivel, respecto de su periodo (variar el ancho de pulso), así se logra controlar la posición correcta. Este funcionamiento se mostrará en detalle más adelante en este trabajo.

Además, para saber si la caja está abierta o cerrada, hacemos uso de un pulsador que hace contacto con la parte superior de la caja.

2- Desarrollo.

2.1- Lista de materiales utilizados.

- Microcontrolador PIC 16F877A.
- 2 capacitores de 15pF y 1 cristal de 4MHZ para el circuito del oscilador XT.
- Regulador de tensión de 9 a 5volts (L7805) y batería de 9V para la alimentación.
- 2 pulsadores, uno utilizado para ver si la caja está abierta o cerrada y el otro para implementar el reset (junto con 1 resistencia de 10K para cada uno para hacer uso del circuito “Pull Up” para el reset).
- 1 Display LCD.
- 1 Led rojo y 1 Led verde para indicar el correcto o incorrecto ingreso de la clave.
- 10 Pulsadores y 10 resistencias de 10K más, para indicar los números del 0 al 9 y así formar los circuitos “Pull Up” para el teclado.
- Mini Servo SG90.
- Protoboard y cables necesarios.
- Mini Zumbador electrónico.

2.2- Descripción de funcionamiento.

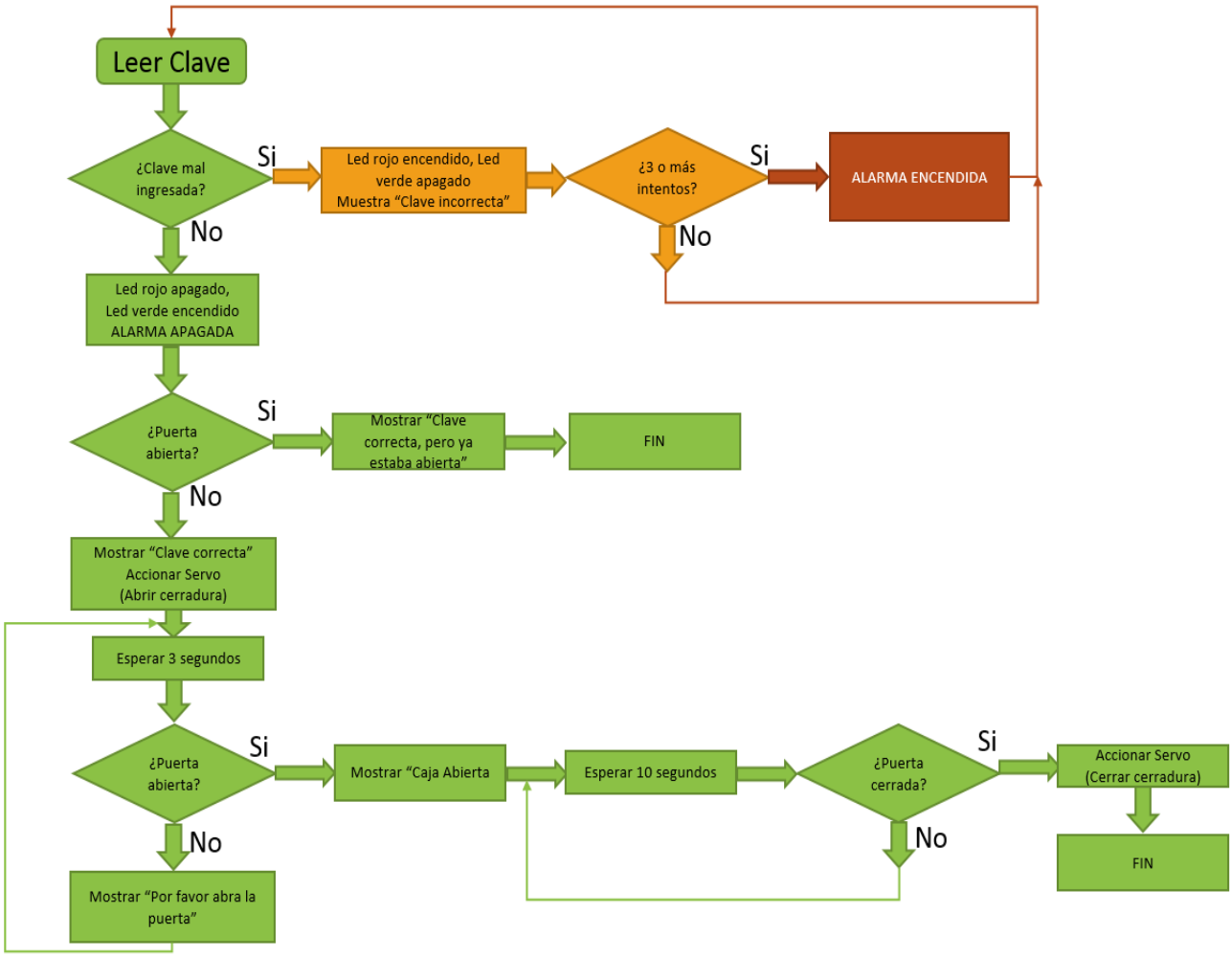


Figura1: Diagrama de flujos del funcionamiento del sistema

Como se puede observar en el diagrama de flujos de la ***Figura1***, el sistema funciona de la siguiente manera, primero se espera que el usuario ingrese una clave de 4 dígitos, a pedido del display con la siguiente información "Bienvenido", y 2 segundos después, "Ingrese clave:", una vez introducida la misma mediante el teclado, se refleja en la pantalla LCD mediante "****" para que el código no se pueda ver. Se produce un reconocimiento y una comparación del valor ingresado, se espera un tiempo, que es el tiempo en que la persona tarda en presionar cada botón por separado, si el valor ingresado es correcto (es decir que si ingresó correctamente los 4 números), se prende un led verde que indica que la clave es correcta; y si la puerta estaba cerrada como es de esperarse, se activa el servomotor que actúa como cerradura, que al ser activado se mueve a la posición -90º, desde su posición original de cierre de 90º (se mueve 180º en total), la cual permite que pueda abrirse la puerta de la caja de seguridad. En el momento que se abre la puerta se indica en el display "Caja abierta". Para saber esto, el sistema cuenta con un contacto en la puerta, es un simple botón "contactor" que permite saber el estado actual de nuestro sistema, si se encuentra abierto o cerrado. Para el caso de que el usuario o persona haya ingresado mal la clave de 4 dígitos, aparece un mensaje de error en la pantalla LCD, se prende un led rojo, y se muestra en el display que la clave es incorrecta, a su vez se indica la cantidad de intentos fallidos, Después del tercer intento se acciona la alarma

(mediante el zumbador) que solo se apaga luego de ingresar la clave correcta. Una vez superados los obstáculos y abierta la caja, el sistema espera que el usuario la cierre, para entonces accionar nuevamente el servo a la posición 90º para cerrar la caja. En el caso de ingresar la clave y que la caja ya haya estado abierta, se muestra el correspondiente mensaje en el display y se cierra el programa.

Pero ahora bien, ***¿cómo hace el sistema el reconocimiento y comparación de la clave?***

En general el código está continuamente detectando cambios en las patitas dispuestas como entradas del PIC, que es donde se conectan los pulsadores que funcionan como teclado, para así cuando se presione una tecla devuelva el carácter presionado (mediante la función "**nroObtenido**()") gracias a la detección de caída a 0V de tensión en el pin correspondiente. Para almacenar la clave de apertura, implementamos un vector de tipo int llamado "**clave**" de longitud 4, donde se almacena el valor que asignamos para nuestra contraseña, la clave es "1123" (o la que nosotros queramos). Por otro lado, tenemos un vector auxiliar llamado "**datos**" que también tiene la misma longitud, el cual es utilizado para guardar cada número que el usuario ingresa mediante el teclado. Una vez completado este vector, se realiza una validación de cada campo del mismo, si los 4 campos del vector de datos coinciden con los 4 campos del vector clave, quiere decir que la persona ingresó bien los 4 dígitos de la clave, de esta manera procedemos a activar el motor para que se abra la puerta en caso de que haya estado cerrada, se prenderá el led verde y se indicará el estado correspondiente en el LCD.

Nota: Un último detalle que hay que tener en cuenta: **Cada vez que el programa llega a su FIN (ver *Figura1*), se debe REINICIAR el sistema con el botón oculto de reset (obviamente con la caja cerrada); y así se vuelve a activar el sistema de seguridad. Esto igualmente lo indicará el LCD diciendo "Reiniciar sistema".

2.3- PIC utilizado.

Para utilizar el microcontrolador PIC16F877A, lo configuramos para utilizar un reloj/clock externo de 4MHZ mediante la siguiente línea en XC8:

#define _XTAL_FREQ 4000000

Usamos este PIC como dijimos anteriormente por su amplia cantidad de puertos y sus 40 pines los cuales la mayoría son de entrada/salida para que los podamos usar. En la ***Figura2*** observamos los pines de conexionado del PIC16F877A.

Este PIC posee 2 VDD y 2 VSS ya que como posee 40 pines, tiene muchos puertos que pueden estar entregando corriente en forma simultánea. Generalmente se conectan los 2 VDD a 5V y los 2 VSS a 0V para que el PIC funcione correctamente, eso fue lo que hicimos.

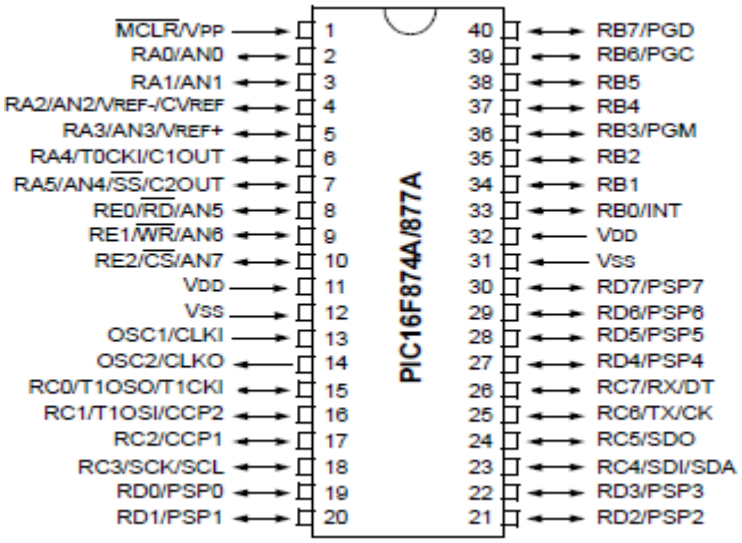


Figura2

2.4-LCD:

Los módulos LCD (Display de cristal líquido), son utilizados para mostrar mensajes que indican al usuario el estado de una máquina o sistema, o para dar instrucciones de manejo, mostrar valores, etc.

En nuestro caso, utilizaremos uno de los más comunes, el LCD 2x16 (2 filas x 16 columnas). Este LCD nos servirá para poder desplegar los dígitos que vamos ingresando a través de nuestro teclado de pulsadores; así mismo indicaremos si la contraseña es correcta o incorrecta para poder conocer el estado de nuestro sistema

El LCD consta de 16 pines, los cuales se muestran en la ***Tabla1.***

PIN ASSIGNMENT		
PIN	SYMBOL	FUNCTION
1	Vss	Power Supply(GND)
2	Vdd	Power Supply(+5V)
3	Vo	Contrast Adjust
4	RS	Instruction/Data Register Select
5	R/W	Data Read/Write
6	E	Enable Signal
7–14	DB0–DB7	Data Bus Line
15	A	Power Supply for LED BL(+)
16	K	Power Supply for LED B/L(–)

Tabla1

Los pines 4, 5 y 6 son **pines de control**, los cuales se explicarán a continuación:

- **Pin 4 (RS):** Corresponde al pin de selección de registro de control de datos (0) o registro de datos (1). Funciona paralelamente a los pines del bus de datos. Cuando RS es 0 el dato presente en el bus pertenece a un registro de control/instrucción; y cuando RS es 1 el dato presente en el bus de datos pertenece a un registro de datos o un carácter.
- **Pin 5 (RW):** Corresponde al pin de Escritura-Write (0) o de Lectura-Read (1). Nos permite escribir un dato en la pantalla o leer un dato desde la pantalla.
- **Pin 6 (E):** Corresponde al pin Enable-habilitación. Si E (0) esto quiere decir que el LCD no está activado para recibir datos, pero si E (1) se encuentra activo y podemos escribir o leer desde el LCD.

De esta manera, los pines **RS** y **E** los usamos en la librería lcd.h, para mandarles un 1 o un 0 mediante software (programando en C). Y al pin RW lo conectamos a GND (un 0 lógico), lo que quiere decir que vamos a escribir (W) y NO leer (R) en la pantalla del LCD.

Los pines 7-14 son pines que corresponden al Bus de datos:

El Bus de datos bidireccional comprende desde los pines 7-14; para realizar la comunicación con el LCD podemos hacerlo utilizando los 8 bits del bus de datos (pines 7-14) o empleando los 4 bits más significativos del bus de datos (pines 11-14). En este caso vamos a usar la comunicación con el bus de 4 bits.

De esta forma, la conexión del LCD nos quedó de la siguiente manera:
Pines de alimentación y brillo:
Pin 1 LCD → GND.
Pin 2 LCD → 5V.
Pin 3 LCD (que sirve para el brillo de la pantalla) → potenciómetro con sus extremos conectados a 5V y a GND.
Pines de control:
Pin 4 LCD → Pin 21 del PIC (al RD2).
Pin 5 LCD → GND.
Pin 6 LCD → Pin 22 del PIC (al RD3).
Pines del Bus de datos:
Pines 7, 8, 9, 10 → Nada, no los usamos ya que se usan si queremos una comunicación con nuestro PIC de 8 bits, pero nosotros usamos una de 4 bits.
Pin 11 LCD → Pin 27 del PIC (al RD4).
Pin 12 LCD → Pin 28 del PIC (al RD5).
Pin 13 LCD → Pin 29 del PIC (al RD6).
Pin 14 LCD → Pin 30 del PIC (al RD7).
Pines para prender las luces de atrás del LCD:
Pin 16 y 15 → No los usamos.

Instrucciones para el manejo del LCD:
Antes de dar las instrucciones las siguientes líneas de código se deben definir antes de la llamada de la librería del LCD:

```
#define RS RD2
#define EN RD3
#define D4 RD4
#define D5 RD5
#define D6 RD6
#define D7 RD7
```

Estos #define hacen referencia a la conexión de los pines del LCD con los del PIC, para así luego usarlos en la librería lcd.h a la cual llamaremos mediante:

```
#include "lcd.h"
```

Esta librería (incluida en el Apéndice), tiene 3 funciones principales que usaremos:
Lcd_Init(); → es la primera función que debe ser llamada, borra la configuración anterior del LCD y lo configura en formato de 4 bits.
Lcd_Clear(); → lo uso para limpiar lo que pusimos anteriormente en el LCD.
Lcd_Set_Cursor(f,c); → indica la posición del acceso al LCD. Por ejemplo (1,2) significa que se va a parar el cursor en la fila 1, columna 2.
Lcd_Write_String("String que quiera"); → me permite imprimir en el LCD un string.
Lcd_Write_Char(´char que quiera´); → me permite imprimir en el LCD un solo carácter.

2.5 Servomotor SG90

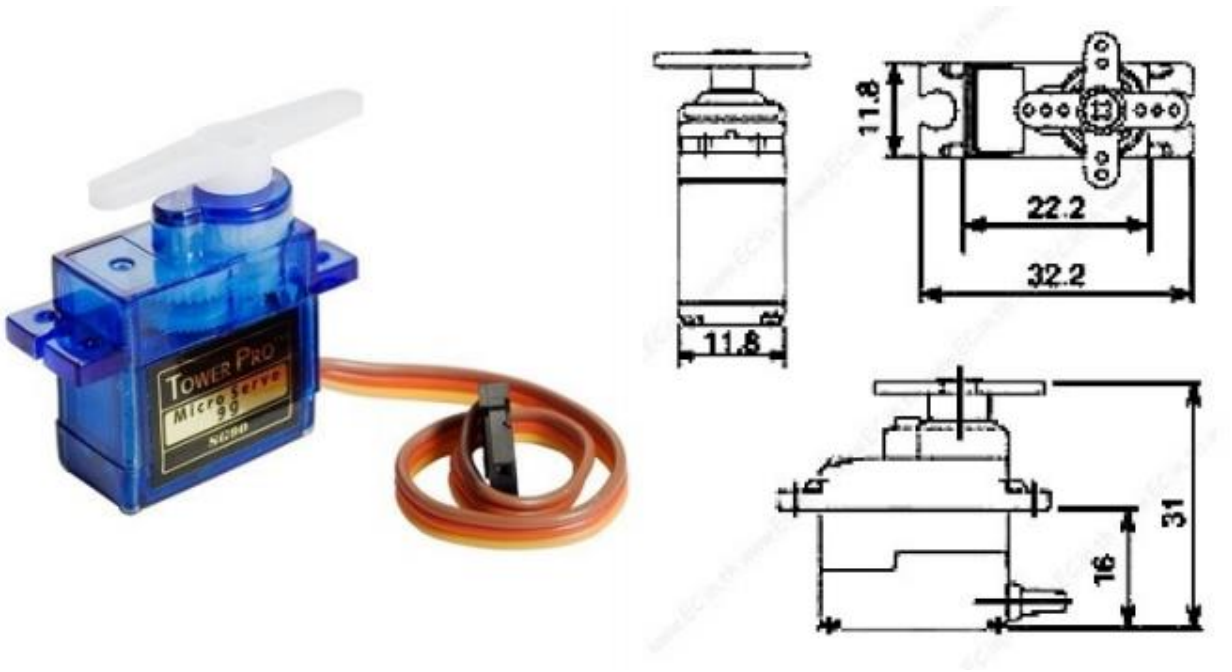


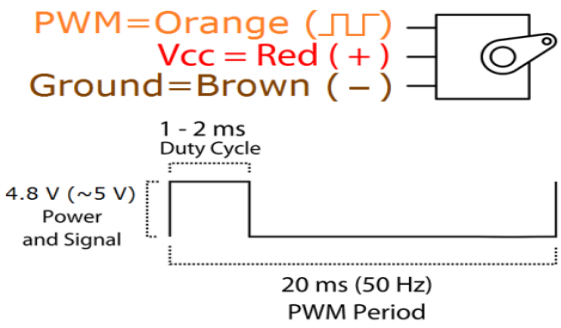
Figura3

En la ***Figura3*** se puede observar el aspecto de este servomotor, que es bastante económico y de bajo consumo energético, lo que lo hace ideal para proyectos de aprendizaje con microcontroladores. Los cables en el conector están distribuidos de la siguiente forma: Rojo =Alimentación (+), Marrón = Tierra, Naranja= Señal PWM.

Especificaciones:

- Peso: 9 gramos
- Medidas: 22.2 mm x 11.8 mm x 31 mm
- Velocidad: 0.1 segundos/60° - 4.8V
- Torque: 1.8 Kgf cm - 4.8V
- Voltaje de funcionamiento: 4.8v (~5v)
- Temperatura de funcionamiento: 0 ℃ - 55 ℃
- Ángulo de rotación: 180° (90º en cada dirección)

Lo interesante de este servo, es que con él se pueden controlar 3 posiciones, 90º, -90º y 0º. A nosotros nos alcanza con utilizar las posiciones 90º (cierre) y -90º (apertura). La forma de controlar las posiciones del servo por medio del PWM puede observarse en la ***Figura4***. El periodo de la señal PWM debe ser de 20ms, el ciclo de trabajo debe ser de 2ms para la posición 90º, algo también interesante es que no importa que la señal PWM se repita indefinidamente, mientras el periodo sea de 20ms con ciclo de trabajo de 2ms, se mantendrá en la posición 90º. Luego para abrir la caja usamos el giro de 180º, con la posición -90º, que se obtiene al tener un ciclo de trabajo de 1ms en un periodo de 20ms. Finalmente está también la alternativa de usar la posición 0, con un periodo de 20ms y un ciclo de trabajo de 1.5ms, pero por razones de comodidad y seguridad preferimos usar solamente 90º y -90º.



Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

Figura4

Respecto a la señal PWM, la controlamos en el PIC mediante delays en nuestra programación en C, como se puede apreciar en el código para mover el servo 90° (posición de apertura de la caja) en la ***Figura5.1***, que muestra un delay de 2ms para el ciclo de trabajo y un delay de 18ms para el ciclo de espera, por lo que sumaría un tiempo de 20ms. Toda esta señal se repite durante aproximadamente 4 segundos (que equivale a 200 periodos, por eso pusimos ***veces<200***), que son más que suficientes para rotar el servo 180º. Cabe destacar que este tiempo lo elegimos de forma suficiente y arbitraria, ya que no importa cuántas veces se repita el periodo de la señal, el servo se detiene una vez alcanzada su posición. En la ***Figura5.2*** observamos el código para mover el servo a -90° (posición de cierre de la caja).

```
while(veces<200)
    {
        pwm=1;
        __delay_us(1000);
        pwm=0;
        __delay_us(19000);
        veces++;
    }
```

Figura5.2: Cierre del servo en -90°.

```
int veces=0;
while(veces<200){
    pwm=1;
    __delay_us(2000);
    pwm=0;
    __delay_us(18000);
    veces++;
    }
```

Figura5.1: **Apertura del servo en 90º.**

El conexionado del SG90 se realizó de la siguiente manera: la masa al negativo de la batería de 9volts, el VCC a la salida del L7805 y el PWM al pin RC2 del PIC. Al RC2 lo seteamos como salida mediante la siguiente línea dentro del main:

TRISCbits.TRISC2=0;

Además, anteriormente lo definimos fuera del main como “pwm” para utilizarlo dentro del main y mandarle un 1 o un 0 haciendo referencia sólo a “pwm” y no a “PORTCbits.RC2”; esto también lo hicimos para todos los pulsadores (incluyendo los del teclado), el buzzer y los leds. Por ej. en el caso del servo lo definimos así:

#define pwm PORTCbits.RC2

2.6 Pulsadores

Pulsadores para los números (nuestro teclado):

Utilizamos un teclado que contaba con 12 pulsadores (de los cuales usamos 10 para representar a los números del 0 al 9) que encontramos en los materiales de reciclaje de la facultad; el detalle de conexionado con el PIC es el siguiente (teniendo en cuenta el circuito "Pull Up" explicado más abajo):

- El botón 1 al RB0.
- El botón 2 al RB1.
- El botón 3 al RB2.
- El botón 4 al RB3.
- El botón 5 al RB4.
- El botón 6 al RB5.
- El botón 7 al RB6.
- El botón 8 al RB7.
- El botón 9 al RC5.
- El botón 0 al RC4.

Cabe destacar, que todas las entradas correspondientes a los pulsadores del teclado (los descriptos arriba: RB0, RB1, etc.), fueron definidas como entrada en nuestro programa en C de la siguiente forma (como ejemplo consideramos sólo al pulsador 9 y al 0):

```
TRISCbits.TRISC4=1; //pulsador 0
TRISCbits.TRISC5=1; //pulsador 9
```

Cada uno de los 10 pulsadores NO los conectamos directamente a los pines del PIC, sino que formamos un circuito "Pull-Up" individual (para cada pulsador) de VCC=5V y con resistencias R1=10K (ver ***Figura6***); lo único que comparten los pulsadores son las masas.

De esta manera, mientras un botón no se presione, el PIC lee 1 lógico (5V) en la entrada del pin correspondiente. Solamente le llega un 0 lógico (0V) en el momento que un botón es presionado.

Pulsador para el RESET:

Fue realizado gracias al **pin MLCR**, logrando que cuando se presiona el pulsador, cambie de 5V a 0V, de esa forma funcionaria como RESET. El esquema que utilizamos es el de "PULL UP" (ver ***Figura6***).

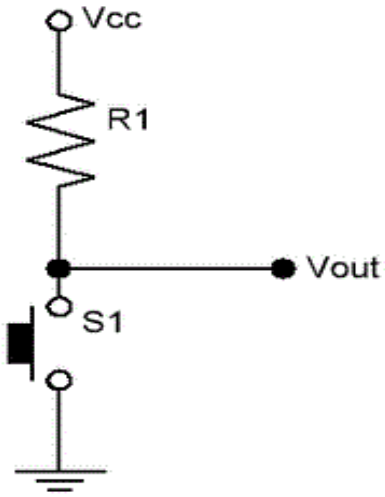


Figura6

En este esquema, VCC=5V y Vout es el pin MCLR del PIC. Cuando es presionado el botón, Vout está conectado a masa, actúa como circuito cerrado, tengo 0v de salida, y el PIC recibe un 0 lógico; cuando suelto el botón vuelvo al estado original (5V), el botón en ese estado esta como circuito abierto, y el PIC recibe un 1 lógico.

Nota: Se puede optar por no utilizar la resistencia Pull Up (R1 en la ***Figura6*), pero es recomendable poner una resistencia de un valor alto (10K por ej.) para que no le llegue tanta corriente a los pines del microcontrolador.

Pulsador para el contactor (de la caja):

Este pulsador (**Figura6.1**) está conectado al RC0, y funciona también con un circuito Pull Up de 5volts y resistencia de 10k. La única diferencia notoria es que **funciona de forma inversa a los botones de números y del reset**, cuando está presionado, el circuito está abierto, y cuando no está presionado, el circuito está cerrado, es decir que funciona a la inversa de los pulsadores normales, el sistema lógicamente interpreta como "caja abierta" cuando el contacto está abierto (cerrando el circuito), es decir que cuando le llegan 0 volts a RC0, quiere decir que la caja está abierta.

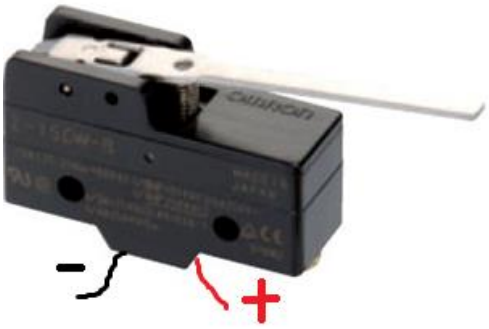


Figura6.1

2.7-Zumbador/buzzer

El Zumbador (ó buzzer en inglés), es un transductor electroacústico que produce un sonido o zumbido continuo de un mismo tono. Sirve como mecanismo de señalización o aviso, y se utiliza en múltiples sistemas, como en automóviles, electrodomésticos y despertadores.

Su construcción consta de dos elementos, un electroimán y una lámina metálica de acero. Al accionarse, la corriente pasa por la bobina del electroimán y produce un campo magnético que hace vibrar la lámina de acero sobre la armadura. De esta manera convierte energía eléctrica en energía acústica. Nosotros utilizamos un zumbador de 5V y de un tamaño pequeño: 9.6x5.0MM (ver ***Figura7***). Como funciona a 5V, quiere decir que al mandarle un 1 lógico desde el PIC a la terminal positiva, se produce el zumbido continuo (si la terminal negativa está a 0V). El símbolo electrónico del buzzer con sus respectivas terminales se puede observar en la ***Figura7.1***.

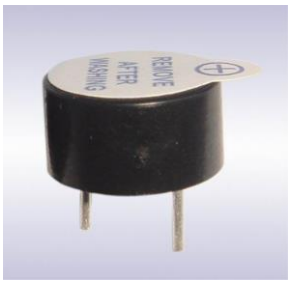
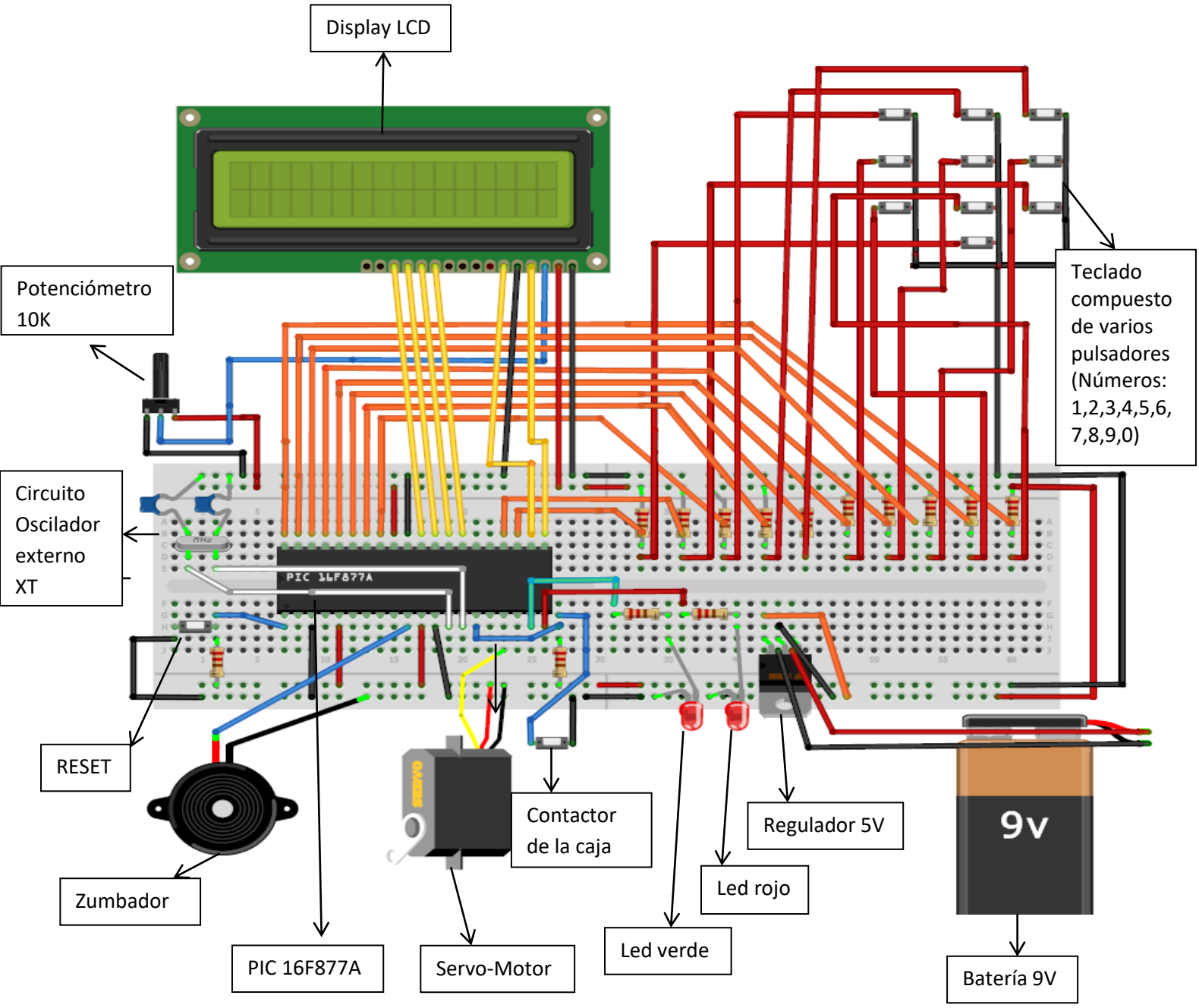


Figura7

Figura7.1

3-Esquema de conexionado final (Protoboard).

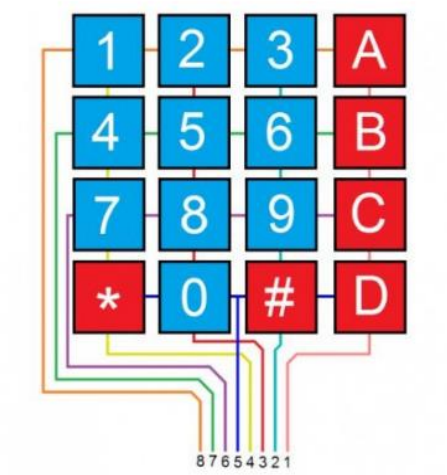


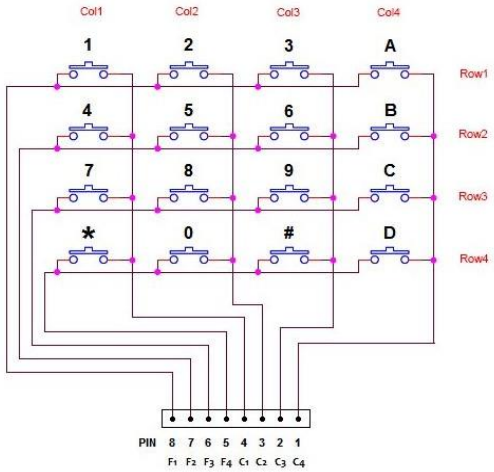
***Notas:**

- Los 2 capacitores del circuito oscilador XT son de 15pF y el cristal es de 4MHZ.
- Todas las resistencias son de 10K a excepción de las 2 resistencias en los leds (que son de 100 ohms).

4-Mejoras a implementar.

- Usar otro material para la caja (aunque este proyecto fue hecho para que el sistema de seguridad funcione adecuadamente, no para que sea una caja fuerte de verdad).
- Guardar la clave en la memoria eeprom del PIC para que esté disponible en todo momento aunque se corte la alimentación (ya que es una memoria no-volátil).
- Usar un teclado matricial como el siguiente:





- Usar el módulo ESP8266 – WIFI mediante sus distintos protocolos para mandar un **mail** a la persona dueña de la caja fuerte indicando que se la quieren abrir.
- Usar el Módulo Lector RFID-RC522 para poder utilizar tarjetas y llaveros especiales RFID para el ingreso de una persona “especial”; utilizando una comunicación serial sincrónica PIC con dicho módulo.
- Implementar un sistema de prevención para que no olviden cerrar la puerta de la caja.
- Implementar un sistema de finalización prudencial en caso de que se ponga la clave correcta y luego no se abra la caja, para evitar que alguien ajeno la abra posteriormente.
- Sacar el reset para Reactivar el sistema y utilizar otra lógica para esto.

5-Conclusión.

Como cierre final del proyecto podemos decir que pudimos terminar en tiempo y forma con la realización del mismo, funcionando correctamente. Tuvimos algunos problemas en el transcurso del TL, pero que simplemente los solucionamos investigando detalladamente en los datasheets de los respectivos componentes. Pudimos reforzar todo lo visto en clases y anteriores TPs, como la programación en C de un microcontrolador usando Pickit2 y XC8 en MPLAB.

6-Bibliografía utilizada.

Páginas web:

- La siguiente web la utilizamos para visualizar los datasheets de los distintos componentes que usamos en el circuito: ***http://www.datasheetcatalog.com/***
- Algunos ejemplos y dudas con el PIC16F877A: ***http://www.todopic.com.ar/***
- ***https://es.wikipedia.org/wiki/Zumbador***

Manuales:

- Manual Conexión Pickit. Autor: D'Angiolo Federico Gabriel
- Manual MPLAB X IDE1. Autor: D'Angiolo Federico Gabriel

7-Apéndice.

Código en C que utilizamos para nuestro TL:

```
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#define _XTAL_FREQ 4000000
#pragma config FOSC = XT
#pragma config WDTE = OFF
#pragma config PWRTE = OFF
#pragma config BOREN = ON
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config CP = OFF

//Defino los pines que uso para posteriormente incluir la librería lcd.h:
#define RS RD2
#define EN RD3
#define D4 RD4
#define D5 RD5
#define D6 RD6
#define D7 RD7
//Incluyo la librería lcd.h:
#include "lcd.h"

//Defino los pulsadores que representan los numeros,
//estan todos a 5V y al apretarlos a 0V, RESISTENCIAS PULL UP:
#define cero PORTCbits.RC4
#define uno PORTBbits.RB0
#define dos PORTBbits.RB1
#define tres PORTBbits.RB2
#define cuatro PORTBbits.RB3
#define cinco PORTBbits.RB4
#define seis PORTBbits.RB5
#define siete PORTBbits.RB6
#define ocho PORTBbits.RB7
#define nueve PORTCbits.RC5

//Defino el zumbador, el contacto dentro de la caja para saber si está
//cerrada o no y el pwm para usar el servo:
#define beep PORTEbits.RE2
#define contactor PORTCbits.RC0
#define pwm PORTCbits.RC2

//Defino los leds:
#define ledRojo PORTDbits.RD1
#define ledVerde PORTDbits.RD0

//Funciones auxiliares / rutinas:
int mostrarIntentos(int intentos){
    if (intentos==1){
        Lcd_Set_Cursor(2,8);
        Lcd_Write_String("Quedan 3");
        return 0;
    }
    else if (intentos==2){
        Lcd_Set_Cursor(2,8);
        Lcd_Write_String("Quedan 2");
        return 0;
    }
    else if (intentos==3){
        Lcd_Set_Cursor(2,8);
        Lcd_Write_String("Queda 1"void Lcd_Write_String(char* a)
        return 0;
    }
    else {
        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("INTRUSO, ingresa");
        Lcd_Set_Cursor(2,1);
        Lcd_Write_String("la clave");
        return 1;
    }
}
```

```
72
73  char nroObtenido(){
74      //Al presionar un numero SALE de la funcion nroObtenido
75      //y retorna el valor que corresponde a ese num
76      if(nueve==0){return '9';}
77      else if(ocho==0){return '8';}
78      else if(siete==0){return '7';}
79      else if(seis==0){return '6';}
80      else if(cinco==0){return '5';}
81      else if(cuatro==0){return '4';}
82      else if(tres==0){return '3';}
83      else if(dos==0){return '2';}
84      else if(uno==0){return '1';}
85      else if(cero==0){return '0';}
86      else return 'n';
87  }
```

```
90  void main(){
91
92      //Entradas y salidas de puertos:
93      //Declaro todo el puerto D como salida (lcd y los 2 leds):
94      TRISD=0x00;
95      //Pulsadores 0 y 9 y contactor como entrada (aunque marque warnings estan bien hechos):
96      TRISCbits.TRISC4=1; //pulsador 0
97      TRISCbits.TRISC5=1; //pulsador 9
98      TRISCbits.TRISC0=1; //contactor
99      //Pwm y zumbador como salida:
100     TRISCbits.TRISC2=0; //PWM
101     TRISEbits.TRISE2=0; //BEEP
102     //Pulsadores de 1-8como entrada:
103     TRISBbits.TRISB0=1;
104     TRISBbits.TRISB1=1;
105     TRISBbits.TRISB2=1;
106     TRISBbits.TRISB3=1;
107     TRISBbits.TRISB4=1;
108     TRISBbits.TRISB5=1;
109     TRISBbits.TRISB6=1;
110     TRISBbits.TRISB7=1;
111     //Si quiero puedo hacer TRISB=11111111; para poner a todos los pulsadores (del 1-8) como entrada
112
113     //Mando un 0 a los dos leds y al zumbador:
114     ledRojo=0;
115     ledVerde=0;
116     beep=0;
117
118     int intentos=0;
119     int ejecucion=1;
120     int estabaabierta=0; //vale 0 si esta cerrada y 1 si esta abierta.
121     int error=0; //para ver los intentos
122     int longitudclave=4;
```

```
123     //Vector donde esta la contraseña:
124     int clave[4]={'1','1','2','3'};
125     //Vector donde se ingresa la clave por medio del teclado
126     //y luego se compara con el vector anterior para ver si son iguales o no:
127     int dato[4];
128     int i=0;
129     char numeroingresado;
130
131     Lcd_Init(); //Inicio el LCD.
132
133     Lcd_Clear(); //Limpio el LCD.
134     Lcd_Set_Cursor(1,1); //Fila 1, Columna 1 del LCD.
135     //Escribo "Bienvenido" en la posicion que puse anteriormente (1,1):
136     Lcd_Write_String("Bienvenido");
137     __delay_ms(2000); //Espero 2 seg.
138
139     Lcd_Clear();
140     Lcd_Set_Cursor(1,1);
141     Lcd_Write_String("Ingrese su clave");
142     int veces=0;
143
144     //Para que el servo empiece en una posición inicial:
145     while(veces<200){
146     pwm=1;
147     __delay_us(1000);
148     pwm=0;
149     __delay_us(19000);
150     veces++;
151     }
```

```
153 while(ejecucion==1)
154 {
155       //While de ingreso de datos
156       while(i<longitudclave)
157           {
158           //Le asigno al char numeroingresado, el valor char que retorna nroObtenido dependiendo de cual bonton
159           //se pulso.
160           numeroingresado=nroObtenido();
161           if (numeroingresado!='n')
162                { dato[i]=numeroingresado; //En la posicion i (empieza en 0) lo igualo al numero que ingrese.
163                  Lcd_Set_Cursor(2,i+1); //En la fila 2 (o sea abajo) y columna "i+1", escribe los **** de la
164                 //contraseña ingresada.
165                  Lcd_Write_Char(numeroingresado);//Si quiero que se muestre el número ingresado le paso
166                 //numeroingresado, sino simplemente un '*'.
167                  __delay_ms(500); //Espero medio seg para volver a presionar otro boton.
168                  i++; //Aumento i.
169                }
170           }
171
172       //Ahora los datos se procesan, comparo los 2 vectores de contraseñas
173       //(la guardada y la ingresada por el usuario):
174       if(dato[0]==clave[0] && dato[1]==clave[1] && dato[2]==clave[2] && dato[3]==clave[3])
175       { //Si coinciden...
176         Lcd_Clear();
177         Lcd_Set_Cursor(1,1);
178         Lcd_Write_String("Clave correcta");
179         ledRojo=0;
180         ledVerde=1;
181         beep=0; //Apago el zumbador
182         __delay_ms(500);
```

```
183             if(contactor==0)
184                     {
185                         Lcd_Set_Cursor(2,1);
186                         Lcd_Write_String("Estaba Abierta");
187                         beep=1;
188                         __delay_ms(1000);
189                         beep=0;
190                         estabaabierta=1;
191                         ejecucion=0; //Para que no entre al while y tenga que reiniciar.
192                     }
193             else{
194                     Lcd_Set_Cursor(2,1);
195                     __delay_ms(500);
196                     Lcd_Write_String("Destrabando");
197                     __delay_ms(500);
198                     int veces=0;
199                     while(veces<200)
200                         {
201                             pwm=1;
202                             __delay_us(2000);
203                             pwm=0;
204                             __delay_us(18000);
205                             veces++;
206                         }
207                     estabaabierta=0;
208                     Lcd_Clear();
209                     Lcd_Set_Cursor(1,1); //Fila 1, Columna 1 del LCD.
210                     Lcd_Write_String("Abra caja");
211                }
```

```
212               if (estabaabierta==0)
213               {
214                   int esperaapertura=1;
215                   while (esperaapertura==1)
216                   {
217                       if(contactor==0)
218                       {
219                           Lcd_Clear();
220                           Lcd_Set_Cursor(1,1);
221                           Lcd_Write_String("Caja abierta");
222                           __delay_ms(1000);
223                           esperaapertura=0;
224                       }
225                   }
226                   __delay_ms(3000);
227                   Lcd_Clear();
228                   Lcd_Set_Cursor(1,1); //Fila 1, Columna 1 del LCD.
229                   Lcd_Write_String("Cierre caja");
230
231                   int esperacierre=1;
232                   while (esperacierre==1)
233                       {
234                           if(contactor==1)
235                           {
236                               Lcd_Clear();
237                               Lcd_Set_Cursor(1,1);
238                               Lcd_Write_String("Caja Cerrada");
239                               __delay_ms(1000);
240                               esperacierre=0;
241                           }
242                       }
```

```
                Lcd_Set_Cursor(2,1);
                __delay_ms(500);
                Lcd_Write_String("Trabando");
                __delay_ms(500);
                int veces=0;
                while(veces<200)
                    {
                        pwm=1;
                        __delay_us(1000);
                        pwm=0;
                        __delay_us(19000);
                        veces++;
                    }

                ejecucion=0; //Para que no entre al while y tenga que reiniciar.

              } //Cierre del if (estabaabierta==0).
            } //Cierre del if(dato[0]==clave[0] && dato[1]==clave[1], etc.)

            else
            {//Si no coinciden...
              Lcd_Clear();
              Lcd_Set_Cursor(1,1);
              Lcd_Write_String("Clave Incorrecta");
              intentos++;
              i=0;  //Siempre hay que volver el cursor al equivocar la clave
              ledRojo=1;
              error = mostrarIntentos(intentos);
```

```
            if (error==1)
                {
                    beep=1; //Suena la alarma hasta que ingrese la clave correcta (si es correcta hago beep=0).
                    //Estas 2 lineas de abajo no tendrían que estar, es solo para que no escuchemos
                    //siempre el zumbido en la práctica y solo sean 2 seg y se apague el zumbador.
                    __delay_ms(2000);
                    beep=0;
                    intentos=3;
                }
            //Aca NO tengo que hacer ejecucion=0; porque tiene que volver al while hasta que sea la contra correcta.
           }//Cierre del else
    }//Cierre del while(ejecucion).

    //Al terminarse el programa debemos reiniciar el sistema:
    while (1)
        {
        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("Reiniciar sistema");
        //El programa se quedará en este while hasta que se reinicie
        }
}//Fin del main
```

Librería LCD.h que utilizamos:

```
void Lcd_Port(char a)
{   if(a & 1)
        D4 = 1;
    else
        D4 = 0;
    if(a & 2)
        D5 = 1;
    else
        D5 = 0;
    if(a & 4)
        D6 = 1;
    else
        D6 = 0;
    if(a & 8)
        D7 = 1;
    else
        D7 = 0;
}
void Lcd_Cmd(char a)
{   RS = 0;
    Lcd_Port(a);
    EN  = 1;
    __delay_ms(4);
    EN  = 0;
}

Lcd_Clear()
{   Lcd_Cmd(0);
    Lcd_Cmd(1);
}
```

```
void Lcd_Set_Cursor(char a, char b)
{   char temp,z,y;
    if(a == 1)
    {
      temp = 0x80 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
    else if(a == 2)
    {
        temp = 0xC0 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
}
```

```
52  void Lcd_Init()
53  {
54    Lcd_Port(0x00);
55     __delay_ms(20);
56    Lcd_Cmd(0x03);
57      __delay_ms(5);
58    Lcd_Cmd(0x03);
59      __delay_ms(11);
60    Lcd_Cmd(0x03);
61    Lcd_Cmd(0x02);
62    Lcd_Cmd(0x02);
63    Lcd_Cmd(0x08);
64    Lcd_Cmd(0x00);
65    Lcd_Cmd(0x0C);
66    Lcd_Cmd(0x00);
67    Lcd_Cmd(0x06);
68  }
69
70  void Lcd_Write_Char(char a)
71  {
72     char temp,y;
73     temp = a&0x0F;
74     y = a&0xF0;
75     RS = 1;
76     Lcd_Port(y>>4);
77     EN = 1;
78     __delay_us(40);
79     EN = 0;
80     Lcd_Port(temp);
81     EN = 1;
82     __delay_us(40);
83     EN = 0;
84  }
```

```
86  void Lcd_Write_String(char *a)
87  {
88      int i;
89      for(i=0;a[i]!='\0';i++)
90         Lcd_Write_Char(a[i]);
91  }
92
93  void Lcd_Shift_Right()
94  {
95      Lcd_Cmd(0x01);
96      Lcd_Cmd(0x0C);
97  }
98
99  void Lcd_Shift_Left()
100 {
101     Lcd_Cmd(0x01);
102     Lcd_Cmd(0x08);
103 }
```