PROGRAMACIÓN DE MICROCONTROLADORES PIC CON LENGUAJE C

Tomo 1

GESTIÓN DE PUERTOS E INTERRUPCIONES EXTERNAS

Sixto Reinoso V.
Luis Mena
Marco Pilatasig
Jorge Sánchez

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Programación de microcontroladores PIC con Lenguaje C, Tomo I
Gestión de puertos e interrupciones externas
Sixto Reinoso; Luis Mena; Marco Pilatasig y Jorge Sánchez

Primera edición electrónica. Octubre de 2018
ISBN: 978-9942-765-36-9
Revisión científica: Juan Pablo Pallo Noroña y Julio Enrique Cuji Rodriguez

Universidad de las Fuerzas Armadas ESPE
Crnl. Ing. Ramiro Pazmiño O.
Rector

Publicación autorizada por:
Comisión Editorial de la Universidad de las Fuerzas Armadas ESPE
Cpnv. Hugo Pérez
Presidente

Edición y producción
David Andrade Aguirre
daa06@yahoo.es

Diseño
Pablo Zavala A.

Universidad de las Fuerzas Armadas ESPE
Av. General Rumiñahui s/n, Sangolquí, Ecuador.
htpp//www.espe.edu.ec

PROGRAMACIÓN DE MICROCONTROLADORES PIC CON LENGUAJE C

Tomo I
GESTIÓN DE PUERTOS E INTERRUPCIONES EXTERNAS

Sixto Reinoso V.
Luis Mena
Marco Pilatasig
Jorge Sánchez

Dedicatoria

Una vez más gracias a DIOS por todo lo que nos da y que es mucho. “Y cantarán de los caminos del SEÑOR, porque grande es la gloria del SEÑOR” (salmo 138:5).

A la memoria de Adriana Alexandra Reinoso V.

CAPÍTULO 1

EL COMPILADOR CCS Y LOS MICROCONTROLADORES PIC

En este capítulo se describen brevemente el software y el hardware a utilizarse en el desarrollo de las aplicaciones propuestas. Es importante hacer un recordatorio de las sentencias y elementos generales que soporta el compilador CCS, y que son utilizados en los programas propuestos en el texto. Detalles, mayor información y características adicionales se pueden obtener del sitio web oficial de CCS en *www.ccsinfo.com* o en manuales y textos que sobre CCS existen ampliamente en el mercado. Luego del análisis del software a utilizarse se describen las características más relevantes de los microcontroladores PIC, dando énfasis a los PICs que se van a utilizar en el desarrollo de las aplicaciones.

ESTRUCTURA DE UN PROGRAMA EN CCS

Para escribir un programa en C que acepte el Compilador CCS, se deben tener en cuenta los elementos que forman el programa los mismos que se indican en la figura 1.1.

Directivas: Controlan como debe generar el compilador el código de máquina.

Programas o Funciones: Bloques de código escrito en instrucciones de lenguajes de alto nivel. Como en C común siempre existirá la función principal *main*.

Instrucciones: Sentencias que indican que va a realizar el programa y determinan la ejecución del mismo. Las instrucciones se separan o finalizan con punto y como (;). Ejemplo: x= x*10;

Comentarios: Describen el significado de las líneas del código. Útil para leer las instrucciones de un programa.

Las directivas, funciones y otros elementos que usa el compilador se irán describiendo según las necesidades de programación. Se dará un vistazo general a los comentarios, datos, variables, instrucciones y funciones que se necesitan para empezar a programar utilizando compilador CCS.

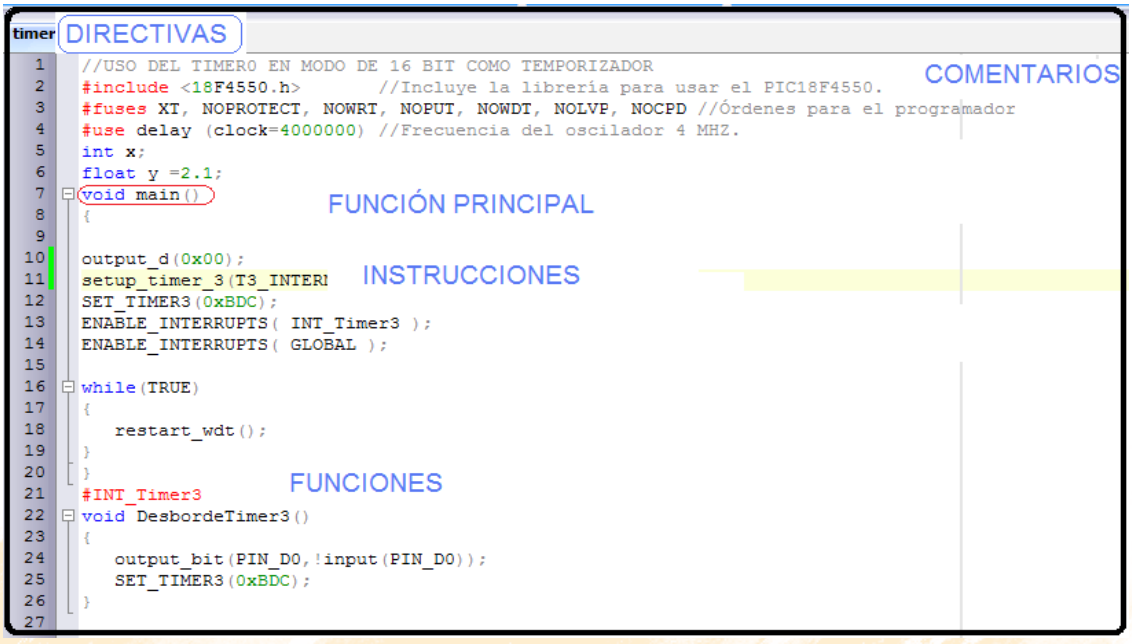


Figura 1.1. Estructura de un programa en C para el compilador PCWHD – CCS.
Elaborado por los autores.

Comentarios

Los comentarios son útiles para interpretar el código de un programa. Se puede colocar en cualquier parte del programa y cumplir con los estándares del ANSI C original. Se pueden escribir en dos formas:

Colocando //. A continuación se escribe el comentario o leyenda correspondiente.

Ejemplos: ***// Esta es la forma de colocar un comentario en una línea.***
delay_ms (200);***//retardo de 200 milisegundos***
#include <18F4550.h> ***//Incluye la librería para utilizar el PIC18F4550***

Colocando /* */. Dentro de los símbolos se inserta el texto del comentario. Cuando se quiere realizar un comentario de varias líneas se utiliza esta opción.

Ejemplos: **/* Otra forma de comentario */.**
delay_ms (200);***/*retardo de 200 milisegundos */***
#include <18F4550.h> ***/*Incluye la librería para utilizar el PIC18F4550. Con esta directiva se define la serie del microcontrolador a utilizarse en el programa */***

Tipos de datos

El compilador CCS acepta los siguientes tipos de datos:

Tipo	Tamaño	Rango	Descripción
Int1 Short	1 bit	0 a 1	Entero de 1 bit
Int Int8	8 bit	0 a 255	Entero
Int16 Long	16 bit	0 a 65,535	Entero de 16 bit
Int32	32 bit	0 a 4,294,967,295	Entero de 32 bit
Float	32 bit	$1.5x10^{-45}$ a $3.402x10^{38}$	Coma flotante
Char	8 bit	0 a 255	Carácter
Void	-	-	Sin valor
Signed Int8	8 bit	-128 a +127	Entero con signo
Signed Int16	16 bit	-32,768 a +32,767	Entero largo con signo
Signed Int32	32 bit	-2147483648 a +2147483647	Entero 32 bits con signo

Tabla 1.1. Tipos de datos aceptados por el compilador CCS.
Fuente: García Breijo. Compilador C CSS y Simulador PROTEUS para Microcontroladores PIC.

Constantes

Las constantes pueden ser decimal, octal, hexadecimal y binario.

453	Decimal
0705	Octal (0)
0x1C5	Hexadecimal(0x)
0b111000101	Binario (0b)
‘x’	Carácter
‘\010’	Carácter octal
‘\xA5’	Carácter hexadecimal
“abcdef”	Cadena

Tabla 1.2. Ejemplos de constantes aceptados por el compilador CCS.
Fuente: García Breijo. Compilador C CSS y Simulador PROTEUS para Microcontroladores PIC.

También se definen las constantes con sufijo.

Int8	83U
Long	70UL
Signed INT16	93L
Float	3.14f
Char	Comillas simples ‘C’

Tabla 1.3. Otra forma de definir constantes en el compilador CCS.
Fuente: García Breijo. Compilador C CSS y Simulador PROTEUS para Microcontroladores PIC.

Se utilizan caracteres especiales como:

\n	Cambio de línea
\r	Retorno de carro
\t	Tabulación
\b	Backspace (retroceso)

Tabla 1.4. Uso de caracteres especiales en el compilador CCS.
Fuente: García Breijo. Compilador C CSS y Simulador PROTEUS para Microcontroladores PIC.

Hay que aclarar que el retorno de carro (\r), pone el cursor al principio de la línea actual, y el cambio de línea (\n) avanza una. Tabulación y backspace, trabajan de forma similar al teclado.

Variables

Las variables sirven para nombrar posiciones de memoria RAM. Se debe declarar obligatoriamente, de acuerdo a las reglas ANSI del lenguaje C. El formato es:

TIPO NOMBRE [=VALOR INICIAL];

TIPO: Se refiere al tipo de datos indicados en la tabla 1.1.

NOMBRE: Un nombre válido de acuerdo a las normas ANSI del lenguaje C común. Algunas reglas mencionamos aquí:

- Deben empezar con una letra o guión bajo. Ejemplo: amperios, _7segmentos;
- No debe empezar con números o caracteres especiales. Ejemplos: 3voltios, 7segmentos, -voltaje, %contador, &luz; contienen números y caracteres especiales que no admite las variables.

- No debe tener el mismo nombre que una "palabra reservada" del lenguaje. Ejemplo: for, if, int, while, do, entre otras son palabras reservadas del lenguaje que no pueden ser usadas.
- No pueden llevar tildes. Ejemplo: Número, ecuación, lección, dígito; son palabras que tienen tilde y no deben usarse.
- No debe existir espacios en blanco. Ejemplo: la ciudad. Hay un espacio en blanco entre las palabras la y ciudad. En esta forma no se puede utilizar.

[=VALOR INICIAL]. Este argumento es opcional (se puede omitir). A una variable se puede asignar un valor de inicio o en su defecto no poner ningún valor. Ejemplo:

int voltaje = 20;
int1 LED;

Las variables pueden ser de tipo LOCAL o GLOBAL. Las variables locales se utilizan sólo en la función donde se encuentran declaradas y las globales en todas las funciones del programa. Ambas variables deben declararse antes de cualquier función y fuera de ella. Las variables globales son puestas a cero cuando se inicia la función principal *main* (). Ejemplo:

PROGRAMA:

```
#include <18F4550.h>
#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000) // F_OSC =12MHz
#byte tris_b = 0xF93
#bit b0= 0xF81.0
int puertos;                    //variable global
void main()
{
        int8 voltaje;           //variable local
}
void función (void)
{
        float res ;             //variable local
}
```

Operadores

El compilador CCS acepta la mayor parte de los operadores: aritméticos, de asignación, relacionales, lógicos y punteros del lenguaje C ANSI original.

1.6.1 Aritméticos

Los operadores aritméticos que soporta el compilador CCS, se resume en la tabla 1.5.

Símbolo	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo, resto de una división entera
++	Incremento
- -	Decremento
Sizeof	Determina el tamaño, en bytes de un operando

Tabla 1.5. Operadores de aritméticos.
Fuente: CCS C Compiler Manual PCD.

De Asignación

Las operaciones de asignación se indican en la tabla 1.6.

Símbolo	Significado
+=	Asignación de suma: m +=n, es lo mismo m= m+n
-=	Asignación de resta: m -=n, es lo mismo m= m-n
*=	Asignación de multiplicación: m *=n, es lo mismo m= m*n
/=	Asignación de división: m /=n, es lo mismo m= m/n
%=	Asignación del resto de la división: m %=n, es lo mismo m= m%n
<<=	Asignación de desplazamiento a la izquierda: m <<=n, es lo mismo m= m<<n
>>=	Asignación de desplazamiento a la derecha: m >>=n, es lo mismo m= m>>n
&=	Asignación AND de bits: m &=n, es lo mismo m= m&n
|=	Asignación OR de bits: m |=n, es lo mismo m= m|n
^^=	Asignación de OR exclusivo de bits
^=	Asignación de suma: m ^=n, es lo mismo m= m^n
~=	~= Asignación de negación de bits

Tabla 1.6. Operadores de asignación.
Fuente: CCS C Compiler Manual PCD.

Relacionales

Compara dos operandos y da un resultado entero: 1 (verdadero); 0 (falso). La siguiente tabla ilustra estos operadores: Los operadores relacionales se muestran en la tabla 1.7.

Símbolo	Significado
<	Menor que
>	Mayor que
>=	Mayor o igual que
<=	Menor o igual que
==	Igual
!=	Diferente
?:	Expresión condicional

Tabla 1.7. Operadores relacionales.
Fuente: CCS C Compiler Manual PCD.

Lógicos

Al igual que los operadores relacionales, éstos devuelven 1 (verdadero), 0 (falso) tras la evaluación de sus operandos. La tabla siguiente ilustra estos operadores. Los operadores lógicos que utiliza el compilador, se indica en la tabla 1.8.

Símbolo	Significado
!	NOT
&&	AND
\|\| | OR |

Tabla 1.8. Operadores lógicos.
Fuente: Manual del usuario del Compilador CCS.

De bits

Estos operadores permiten actuar sobre los operandos a nivel de bits y sólo pueden ser de tipo entero (incluyendo el tipo char). Las operaciones de bits están representadas en la tabla 1.9.

Símbolo	Significado
~	Complemento a 1 (Inversión)
&	AND
^^	XOR
\| | OR |

Tabla 1.9. Operadores de bits.
Fuente: Manual del usuario del Compilador CCS.

Operadores de desplazamiento

Estos operadores utilizan dos operandos enteros (tipo int): el primero es el elemento a desplazar y el segundo, el número de posiciones de bits que se desplaza.

>>	Desplazamiento a la derecha.
<<	Desplazamiento a la izquierda

Tabla 1.10. Operadores de desplazamiento.
Fuente: Manual del usuario del Compilador CCS.

El formato es: *m Operando n.*

Donde:

m, es el elemento a desplazar.

Operando, desplazamiento a la izquierda <<, o desplazamiento a la derecha >>, y

n, indica el número de posiciones que se desplazarán los bits.

Ejemplo*:* En una variable x se almacena el número binario 111000101. Si se realizan las operaciónes indicadas, cuáles son los nuevos valores de x para cada caso.

a. x= 11000101 << 1
b. x= 11000101 << 2
c. x= 11000101 << 3
d. x= 11000101 >> 1
e. x= 11000101 >> 2
f. x= 11000101 >> 3

En todos los casos los "unos" son reemplazados por "ceros", desde el bit menos significado y los bits que superan la capacidad del entero del bit más significativo se desbordan y desaparecen.

El 1 después del signo << indica que los bits se desplazarán una posición a la izquierda. Por tanto, el nuevo valor de x es 10001010. En este caso es lo mismo que realizar una multiplicación al número binario por 2.

El 2 después del signo << indica que los bits se desplazarán dos posiciones a la izquierda. Por tanto, el nuevo valor de x es 00010100. En este caso es lo mismo que realizar una multiplicación al número binario por 4.

El 3 después del signo << indica que los bits se desplazarán tres posiciones a la izquierda. Por tanto, el nuevo valor de x es 00101000. En este caso es lo mismo que realizar una multiplicación al número binario por 8.

Para el desplazamiento a la derecha, los "unos" son reemplazados por "ceros" desde el bit más significativo y al superar la capacidad del número el bit menos significativo desparece.

El 1 después del signo >> indica que los bits se desplazarán una posición a la derecha. Por tanto, el nuevo valor de x es 01100010. En este caso es lo mismo que realizar una multiplicación al número binario por 2.

El 2 después del signo >> indica que los bits se desplazarán dos posiciones a la derecha. Por tanto, el nuevo valor de x es 00110001. En este caso es lo mismo que realizar una multiplicación al número binario por 4.

El 3 después del signo >> indica que los bits se desplazarán tres posiciones a la derecha. Por tanto, el nuevo valor de x es 00011000. En este caso es lo mismo que realizar una multiplicación al número binario por 4.

Punteros

El compilador CCS, soporta también el trabajo con punteros. Los operadores de punteros más comunes se presentan en la tabla 1.10.

Símbolo	Significado
&	Dirección
*	Indirección
->	Puntero estructura

Tabla 1.11. Operadores punteros.
Fuente: Manual del usuario del Compilador CCS.

Instrucciones de control

Las instrucciones de control para los programas con condiciones o que necesiten bucles o lazos repetitivos que admite CCS, son los mismos del ANSI C original:

- IF - ELSE
- SWITCH – CASE
- WHILE
- DO – WHILE
- FOR

- CONTINUE
- RETURN
- BREAK
- GOTO

IF – ELSE

Sirve para tomar decisiones en un programa. El formato es:

```
if (condición)
   Instrucción1;
[else
Instrucción2];
```

Si la condición es verdadera (True o 1) se ejecuta la Instrucción1, si es falsa (False o 0) se ejecuta la instrucción2. Las sentencias que están entre corchetes [], son opcionales. Ejemplo:

```
if (x==1)
   bit_set(port_b,0);
else
   bit_set(port_b,1);
```

Cuando existen varias instrucciones se deben utilizar las llaves {}.

```
if (prender == 0 && k==0)
    {
      k= 1;
      bit_set(port_b,0);
    }
```

Se pueden formar instrucciones anidadas para formar lazos de decisión ELSE – IF.

```
if (condición1)
   Instrucción1;
[else if (condición2)
Instrucción2;]
[else
Instrucción3;]
```

La mejor forma de representar la instrucción IF – ELSE es mediante la figura 1.2. Si la condición es TRUE se ejecutan las instrucciones que están por el lado del THEN o si la condición es FALSE se ejecutan las instrucciones del lado del ELSE.

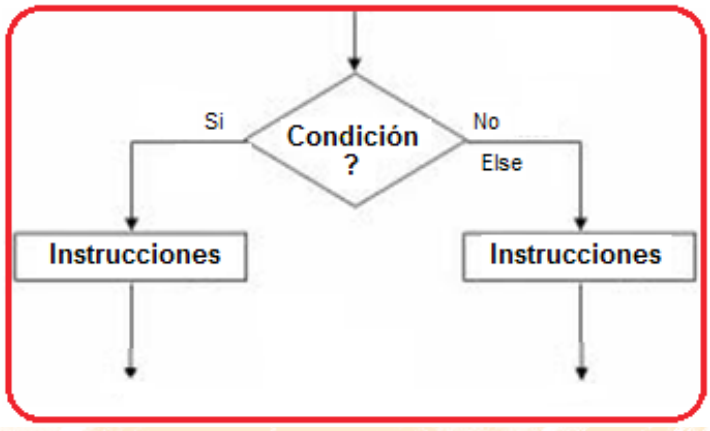


Figura 1.2. Representación de la instrucción IF – ELSE.
Elaborado por los autores.

SWITCH CASE

Se utiliza para tomar una decisión múltiple. Evalúa la expresión y de acuerdo a la constante ejecuta la o las instrucciones. La instrucción *break* evita que continúe ejecutando el siguiente case. Si no existe ningún *case* se ejecuta las instrucciones de default (que es opcional).

```
switch (expresión)
{
case constante_1:
                instrucción_11;
                instrucción_12;
                ...
                break;
case constante_2:
                instrucción_21;
                instrucción_22;
                ...
break;
case constante_n:
```

```
                instrucción_n
                ...
                break;
    [default:
    Instrucciones;]
    }
```

WHILE

WHILE repite una serie de instrucciones, mientras la expresión sea cierta (TRUE o 1).

```
while (expresión)
{
                instrucciones;
}
```

Se puede ejecutar un bucle sin fin.

```
while (1)
{
                instrucciones;
}
```

O también utilizando el formato:

```
while (TRUE)
{
                instrucciones;
}
```

DO WHILE

Con DO WHILE la expresión se evalúa al final de las instrucciones.

```
do
{
instrucciones;
}
while(expresión);
```

Las instrucciones primero se ejecutan y se evalúa la expresión al final, si ésta resulta cierta (TRUE), se repite el bucle hasta que la condición sea falsa (FALSE).

También con DO WHILE, se puede ejecutar un bucle sin fin. Por ejemplo:

```
do
{
        instrucciones;
}
while(1);
```

FOR

El bucle de control FOR sin dudas es el más usado en la mayoría de lenguajes de programación. En CCS es similar al formato del ANSI C original, o sea:

```
for (valor_inicial, valor_final, incremento)
{
        instrucciones;
}
```

Ejemplo:

```
for (j=0; j<=9;++j)
```

Las instrucciones se repiten mientras la variable j sea menor o igual a 9. La variable *j* se incrementa su valor en 1 cada vez que se ejecute una instrucción.

Nota: Recuerde que en C una variable se puede incrementar usando ++j, o bien, j++.

Un bucle sin fin usando FOR se genera mediante el siguiente formato:

```
for ( ; ; )
{
        instrucciones;
}
```

Continue

La instrucción de salto CONTINUE se usa para interrumpir la ejecución normal de un bucle. El control del programa no se transfiere a la primera instrucción después del bucle (como sí hace la instrucción break), es decir, el bucle no finaliza, sino que, finaliza la iteración en curso, transfiriéndose

el control del programa a la condición de salida del bucle, para decidir si se debe realizar una nueva iteración o no.

CONTINUE se puede emplear con cualquiera de los tres bucles, pero no con SWITCH. Al igual que antes el programa no ejecuta las sentencias restantes del bucle donde se encuentra. En bucles FOR o WHILE se comienza a ejecutar el siguiente ciclo del bucle. En DO WHILE se comprueba la condición de salida; si se cumple, se comienza el nuevo ciclo.

RETURN

El formato de la instrucción es:

return (expresión);

return expresión;

Se usa para devolver datos en las funciones. El valor de *expresión* debe ser el mismo definido en la función.

BREAK

La instrucción BREAK sale de la ejecución de un bucle for, while, do y switch.

GOTO

Es un salto incondicional. En C casi no se utiliza porque se pierde la conceptualización de programa estructural y modular.

Funciones de retardo

Cuando se trabaja con microcontroladores es muy frecuente que se necesitan generar retardos; el compilador CCS, proporciona algunas funciones para estas situaciones.

delay_ms(tiempo)

Esta función realiza retardos del valor especificado en la variable tiempo. El valor es un int16 (0-65535) en milisegundos. Ejemplo:

```
delay_ms(500);          //genera un retardo de 500 ms.
```

delay_us(tiempo)

Los retardos conseguidos con esta función son especificados en la variable *tiempo* en microsegundos y es un int16 (0-65535). Ejemplo:

```
delay_us(500);          //genera un retardo de 500 us.
```

delay_cycles(contador)

Con esta función, se realizan retardos de acuerdo al número de ciclos de máquina especificado en la variable *contador*. Los valores van desde 1 a 255. Un ciclo de máquina o instrucción es igual a cuatro ciclos de reloj.

Ejemplo:

```
delay_cycles( 5 );          // retardo de 5 ciclos instrucción.
```

Es decir, si el microcontrolador trabaja con un oscilador de 4 MHz, el ciclo de instrucción o de máquina (CM) será igual:

*CM= 4*1/4000000Hz = 1us*

Por tanto, en el ejemplo anterior se ha obtenido un retardo de 5 us.

Directiva #use delay (clock=frecuencia)

Las tres funciones de retardo descritas trabajan en conjunto con la directiva ***#use delay.*** Esta directiva indica al compilador que frecuencia de reloj va utilizar el microcontrolador. Ejemplos:

#use delay (clock=4000000) //Indica que MC va utilizar un oscilador de 4MHz.

#use delay (clock=20000000) //Indica que MC va utilizar un oscilador de 20MHz.

También se puede escribir:

#use delay (clock=4M) //Indica que MC va utilizar un oscilador de 4MHz.

#use delay (clock=20M) //Indica que MC va utilizar un oscilador de 20MHz.

Los microcontroladores PIC

Los microcontroladores PIC de Microchip Technology INC., son los número 1 en ventas a nivel mundial, por las diferentes ventajas que ofrecen, entre las que se puede mencionar:

- Diversidad de gamas de microcontroladores para todas las necesidades y aplicaciones que el diseñador pueda imaginarse.
- Soporte técnico documental amplio: libros, manuales, tutoriales, aplicaciones específicas, abundante información en internet.
- Gran cantidad de herramientas de desarrollo software y hardware.
- Compatibilidad en la arquitectura entre las diferentes familias de microcontroladores PIC.
- Unidades funcionales y módulos embebidos: temporizadores, conversores analógicos-digitales, USB, UART, PWM, CCP, etc.
- Precios muy competitivos.

Familias de microcontroladores PIC

Microchip produce diferentes dispositivos que según el número de terminales lo clasifica en:

Gama baja

- Microcontroladores de 8 bits, de bajo coste, de 6 pines y bajas prestaciones. Algunos PIC's de las serie PIC10 se conocen como gama enana.
- PIC12: microcontroladores de 8 bits, de bajo coste, de 8 pines y bajas prestaciones.

Gama media

- PIC16: microcontroladores de 8 bits, con gran variedad de número de pines y prestaciones medias.

Gama alta

- PIC17 y PIC18: microcontroladores de 8 bits, con gran variedad de número de pines y prestaciones medias/altas.
- PIC24: microcontroladores de 16 bits.
- PIC32: microcontroladores de 32 bits.
- dsPIC's. Microcontroladores especializados para el procesamiento de señales digitales.

PIC18: Características fundamentales

Las principales características de los microcontroladores de gama alta PIC18, se puede resumir en los siguientes:

- Arquitectura Harvard, RISC avanzada de 16 bits (instrucciones) con 8 bit de datos.
- 77 instrucciones a nivel de assembler.
- Desde 18 a 80 pines.
- Tecnología CMOS.
- Puertos de I/O con múltiples funciones.
- Módulos CCP. ECCP, PWM y comparadores incorporados.
- Conversores con múltiples canales multiplexados.
- Hasta 64K bytes de memoria de programa (hasta 2 Mbytes en ROMless).
- Multiplicador Hardware 8x8.
- Hasta 3968 bytes de RAM y 1KBytes de EEPROM.
- Frecuencia máxima de reloj 40Mhz. Hasta 10 MIPS.
- Pila de 32 niveles.
- Múltiples fuentes de interrupción.
- Resistencias pull-up programables en el puerto B.
- Temporizadores 8 y 16 bits.
- Comunicación serial sincrónica y asincrónica.
- Periféricos de comunicación avanzados (CAN y USB).

La tabla 1.12, muestra las características generales de los PIC18F2455, PIC18F2550, PICF4455 y PIC18F4550 que son los más populares.

Device	Program Memory		Data Memory		I/O	10-Bit A/D (ch)	CCP/ECCP (PWM)	SPP	MSSP		EUSART	Comparators	Timers 8/16-Bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I^2C™			
PIC18F2455	24K	12288	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F2550	32K	16384	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F4455	24K	12288	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3
PIC18F4550	32K	16384	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3

Tabla 1.12. Características PIC18Fxxx.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Del listado de microcontroladores indicados en la tabla 1.12, el PIC18F4550, es uno de los más difundidos por sus altas prestaciones y recursos que dispone. Tiene 40 pines con 5 puertos numerados A, B, C, D y E. Cada puerto puede estar formado hasta con 8 líneas que se pueden configurar como entradas o salidas digitales. Además cada línea tiene más de una función como se ve en la figura 1.3.

Registros de los puertos

Cada puerto tiene tres registros para su funcionamiento. Estos registros son:

- **Registro TRISx.** Registro de la dirección de datos. El compilador CCS utiliza la instrucción **TRISx** para definir al puerto como salida (TRISx = 0) o como entrada (TRISx= 1).
- **Registro PORTx.** Lee los niveles en los pines del dispositivo. Los datos existentes en los pines son leídos a través del registro PORT. El compilador CCS a través de la instrucción **PORTx** escribe datos al puerto respectivo.
- **Registro LATx**. Latch de salida. Registro latch o cerrojo, almacena los estados lógicos que van a sacar las líneas de salida. No existe ninguna instrucción del compilador CCS asociada con el registro LAT.

Cada pin de los puertos que sea de entrada/salida tiene incorporado una línea al registro PORT, LAT y TRIS. La letra **x** en TRISx, PORTx y LATx corresponde al nombre del puerto usado (A, B, C, D, E).

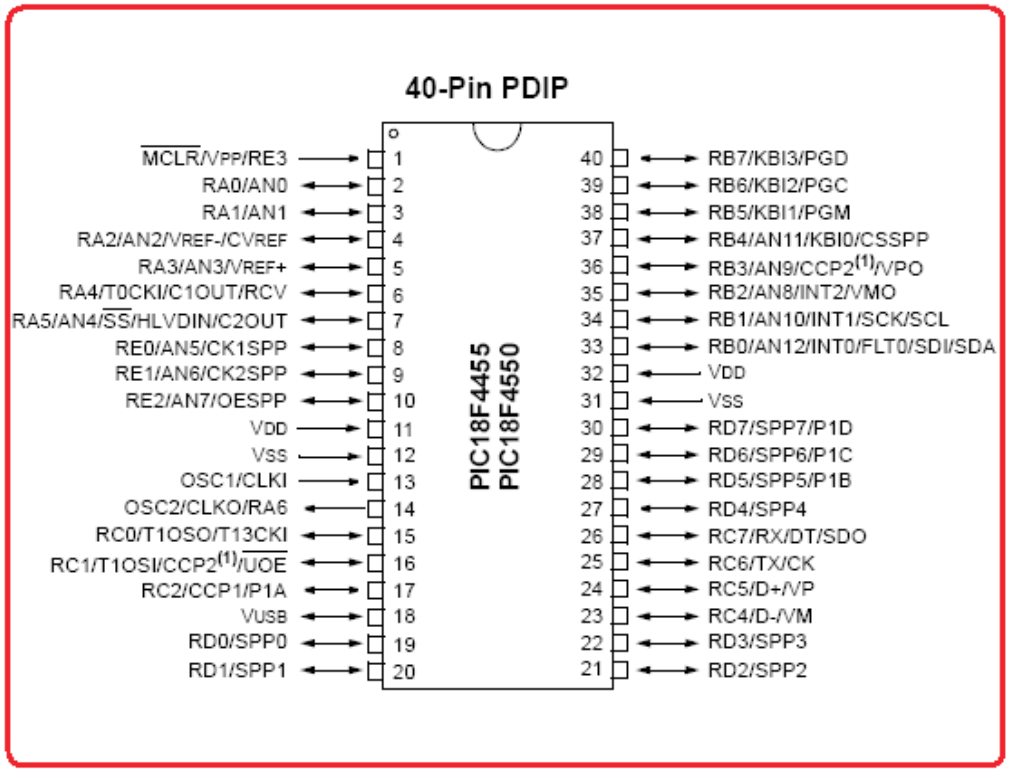


Figura 1.3. Distribución de pines del PIC18F4550, versión DIP.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

La figura 1.4, muestra el diagrama de conexiones de un puerto de entrada/salida para su operación.

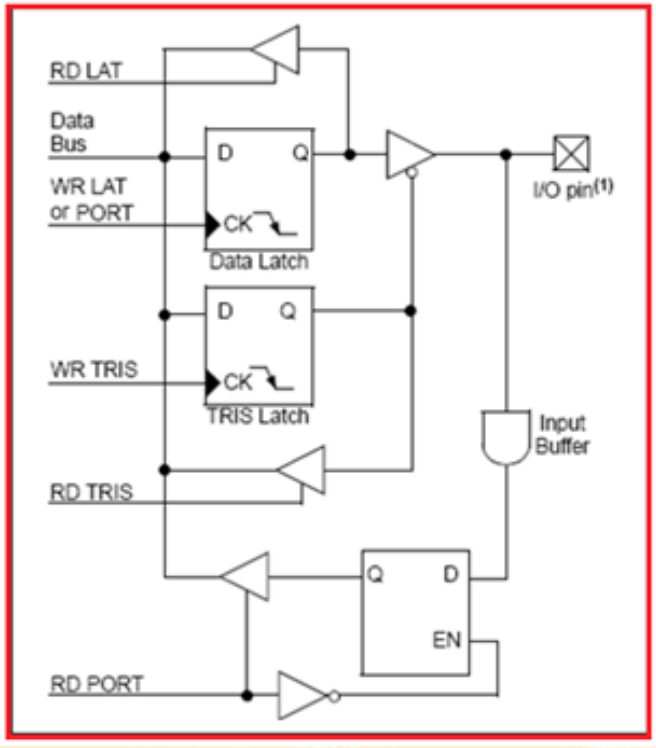


Figura 1.4. Diagrama de un puerto entrada/salida.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

PIC16: Principales características

Indudablemente la familia más numerosa Microcontroladores PIC de gama media de la familia son los PIC16, en sus diferentes series y tecnologías son las más utilizadas y exitosas.

Una de las series más famosas han sido los PIC de 18 pines como son el PIC16F84, PIC16F627A, PIC16F628A y actualmente el PIC16F88. Los micros de 40 pines el más destacado ha sido el PIC16F877 que está siendo reemplazado por la versión mejorada del PIC16F887.

De acuerdo a las hojas de datos proporcionados por Microchip, las características generales que presenta esta gama son:

- Arquitectura Harvard, RISC avanzada de 14 bits (instrucciones) con 8 bit de datos.
- 35 instrucciones a nivel de assembler.
- Desde 18 a 68 pines.

- Hasta 8K bytes de memoria de programa.
- Frecuencia máxima de reloj 20MHz.
- Múltiples fuentes de interrupción.
- Módulos de comunicación sincrónica y asincrónica.
- Módulos PWM/CCP/ECCP y comparadores.
- Circuitos temporizadores de 8 y 16 bits.
- Memoria RAM de hasta 368 bytes.
- Memoria EEPROM de hasta 256 bytes.
- Conversor con canales multiplexados.
- Resistencias pull-up programables en el puerto B.
- Dos puertos I/0 en los micros de 18 pines y 5 puertos en los PIC de 40 pines.

El PIC16F84, fue el que revolucionó el diseño de circuitos con microcontroladores y se utilizó en la mayoría de centros de enseñanza la programación en base a este PIC.

Los PIC 16F627A, 16F628A y 16F648A, ingresan al mercado con características mejoradas y reemplazan al PIC16F84A, siendo el PIC16F628A el de mayor acogida. Tienen una configuración para 8 osciladores. Un único pin para un circuito RC es la solución de más bajo coste. En modo LP, para mínimo consumo, XT para oscilador estándar y dos fuentes internas de precisión INTOSC. El modo HS para cristales de alta velocidad (hasta 20MHz) y el modo EC para una fuente externa de reloj. En estas series el pin RA4 es de drenador abierto y el pin RA5 que corresponde al MCLR (Master Clear), solo puede actuar como puerto de entrada.

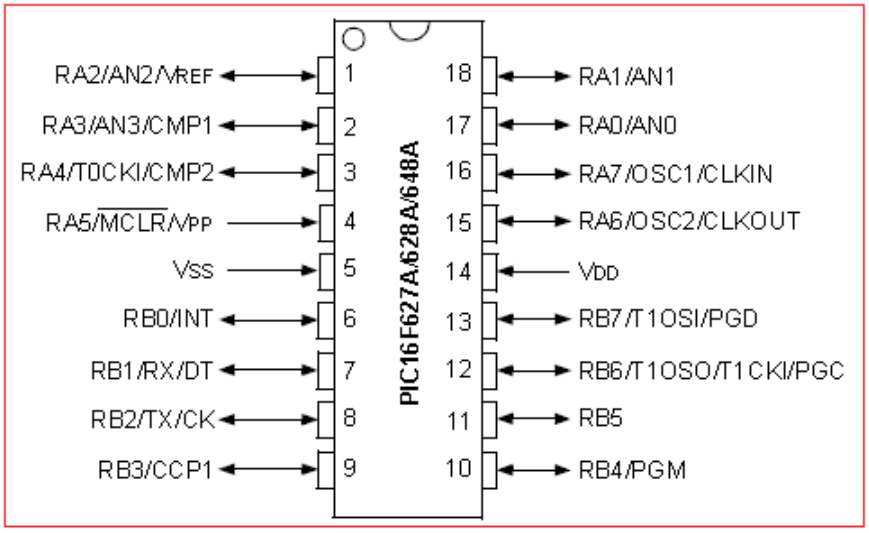
El microcontrolador PIC16F88, es 100% compatible con sus antecesores microcontroladores 16F628A/84A, pero incluye un módulo conversor análogo digital y el doble de la capacidad de la memoria FLASH. Además el PIC en mención, incluye un bloque de oscilador interno con 8 frecuencias seleccionables que van desde 31KHz hasta 8MHz, tres modos de osciladores de cristal externo LP, XT y HS hasta 20MHz, dos modos externos RC y un modo de reloj externo ECIO de hasta 20MHz. También, es un microcontrolador multifunciones, con capacidad de autoprogramación mediante firmware bootloader residente.

La tabla 1.13, presenta un resumen de las características más importantes de los microcontroladores mencionados en este punto.

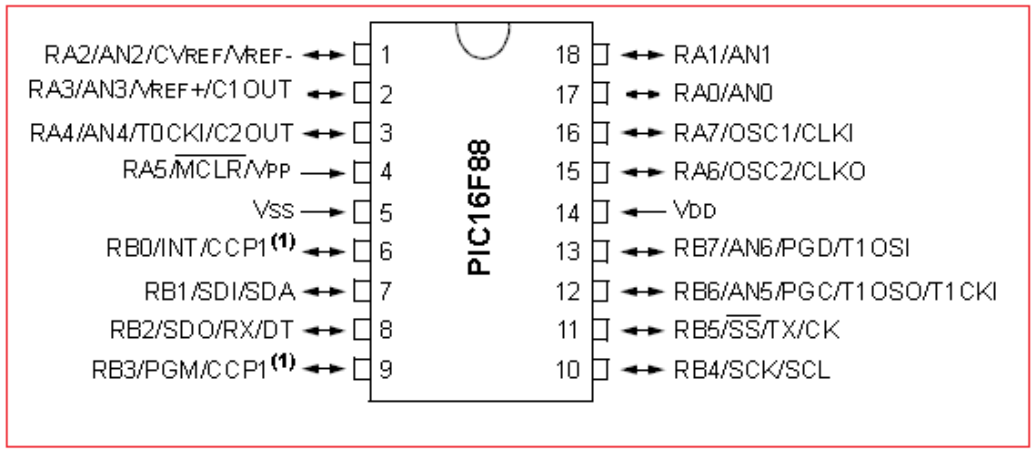
CARACTERÍSTICAS	PICF16F84A	PIC16F627A	PIC16F628A	PIC16F648A	PIC16F88
Memoria FLASH (Words)	1024	1024	2048	4096	4096
Memoria EEPROM (bytes)	64	128	128	256	256
Memoria SRAM (bytes)	68	224	224	256	368
Puertos I/O	2	2	2	2	2
Pines de I/O	13	16	16	16	16
Oscilador	Externo	Interno/Externo	Interno/ Externo	Interno/Externo	Interno/Externo
Fuentes de interrupción	10	10	10	10	12
Comunicación serial	USART	USART	USART	USART	AUSART, SSP (SPI e I2C)
Comparadores	-	2	2	2	2
Conversores AD	-	-	-	-	7 canales mulltiplexados

Tabla 1.13. Características principales de los microcontroladores PIC16F84/627A/628A/88.
Elaborado por: Los Autores.

La figura 1.5 a, muestra el diagrama del microcontrolador 16F627A/628A/648A y la figura 1.5 b, el diagrama del microcontrolador 16F88, en versiones DIP.



a. Diagrama de distribución de pines de los PIC16F627A/628A/648A.



b. Diagrama de distribución de pines de los PIC16F88.

Figura 1.5. Microcontroladores PIC gama media de 18 pines, series 16FXXX versión DIP.
Fuente: PIC16F87/88 Data Sheet, PIC16F627A/628A/648A Data Sheet.Microchip Technology INC.

En versiones de 28, 40 y 44 pines el microcontrolador de gama media 16F877A ha sido uno de los más exitosos. Entre las características más destacadas de este micro está que tiene 5 puertos (A, B, C, D, E). Posee un módulo de Comunicación Serial Universal Asincrónico USART, que soporta RS-485, RS-232 y LIN2.0 y un Puerto Serie Síncrono Maestro (MSSP), para modos SPI (Serial Peripheral Interface) e I2C (Inter-Integrated Circuit).

El PIC16F877A no incluye un sistema de reloj interno, por lo que Microchip para suplir esta necesidad introduce el PIC16F887, con varias mejoras. Este micro posee un sistema de selección de reloj interno de precisión desde 8MHz a 31KHz. Posee un registro postescaler de 8 bits, para dividir la fuente de 8MHz y obtener 8 frecuencias distintas (8, 4, 2, 1 MHz, 500, 250, 125 y 31.25 kHz). Los modos de oscilador externo son LP, XT, HS, RC, RCIO, EC, y en oscilador interno INTOSC y INTOSCIO.

El módulo CCP mejorado (ECCP) útil para el control PWM de puentes y semipuentes para circuitos inversores, es otra de las características relevantes de este PIC. El módulo EUSART (Enhanced USART), permite la autodetección de baudios.

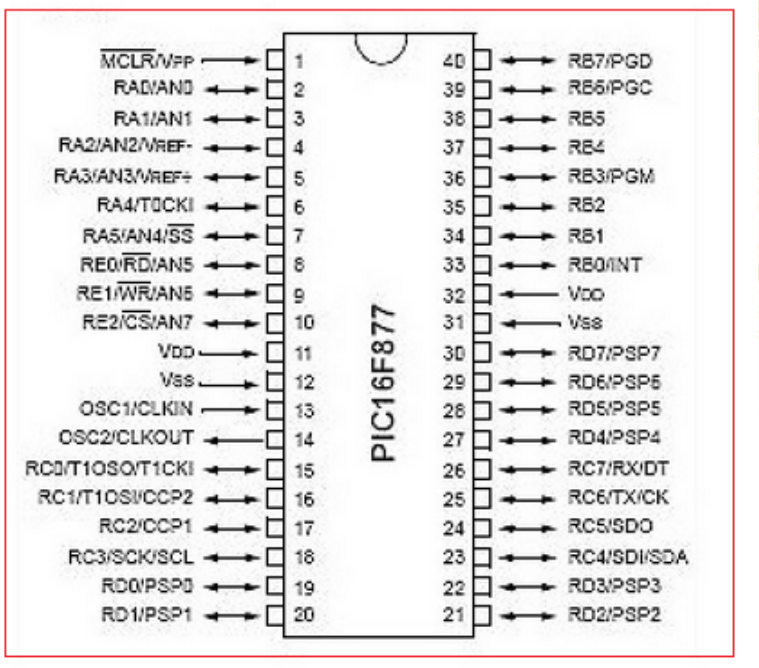
El módulo ADC, es de 10 bits y tiene hasta 14 canales multiplexados para ser utilizados simultáneamente. Los microcontroladores de esta gama poseen un amplio rango de voltaje de operación que va de entre 2 y 5.5V.

La tabla 1.14, muestra un resumen comparativo de las principales características que distinguen a estos dos microcontroladores.

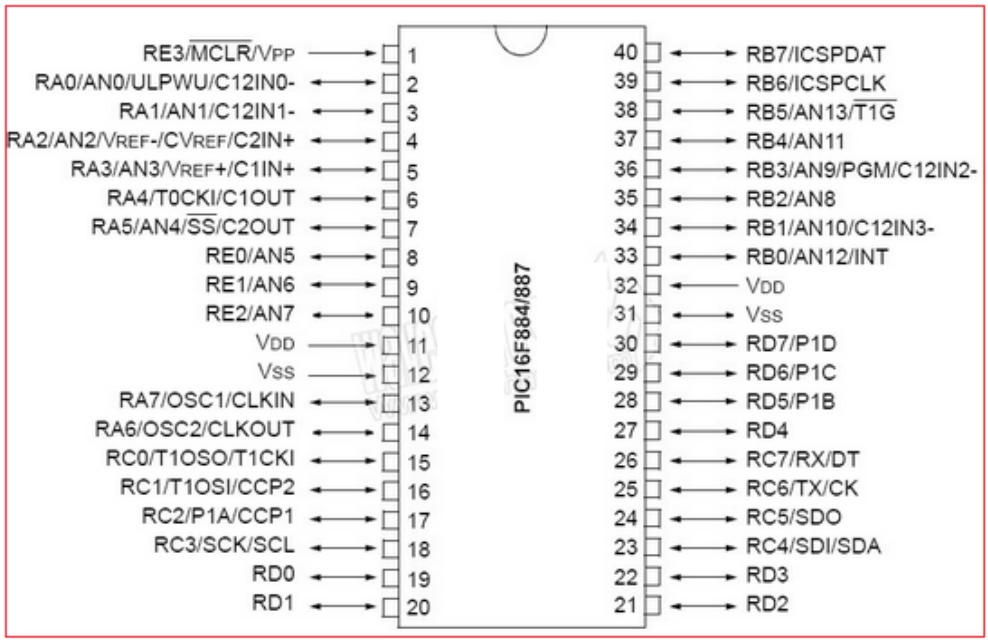
CARACTERÍSTICAS	PIC16F877A	PIC16F887
Frecuencia Máxima	20MHZ	20MHZ
Memoria FLASH (Words)	8192	8192
Memoria EEPROM (bytes)	256	256
Memoria SRAM (bytes)	368	368
Puertos I/O	33	35
Timer (8/16 bits)	½	½
Oscilador	Externo	Interno/Externo
Comparadores	2	2
Conversores AD	8 canales (10 bits)	14 canales (10 bits)
Frecuencias INTOSC	-	31 KHZ-8 MHZ
Fuentes de interrupción	15	15
ECCP/CCP	0/2	½
Comunicación serial	USART/MSSP	EUSART/MSSP
Modos de oscilador	4	8

Tabla 1.14. Características de los PIC 16F877A/887.
Elaborado por los autores.

Los diagramas de distribución de pines para los PIC16F877A/887, versión DIP, se observa en la figura 1.6.



a. Diagrama del PIC16F877.



b. Diagrama de distribución de pines del PIC16F887.

Figura 1.6. Microcontroladores PIC16F877/887 versión DIP.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

EJERCICIOS Y PREGUNTAS PROPUESTAS

1. ¿Cuál carácter se utiliza para hacer comentarios en una sola línea?

2. ¿Con qué carácter se separan instrucciones?

3. ¿Cuál es la diferencia entre retorno de línea y cambio de línea?

4. ¿Qué palabra se debe utilizar para declarar un variable entero de 1 bit?

5. Realice la declaración de la variable I = 2.536

6. ¿Cuáles son las restricciones que se deben tomar en cuenta al declarar una variable?

7. Encuentre el error en la declaración de la variable y corríjala: float 2 contadores = 5;

8. Una variable int16 ¿qué rango de valores contiene?

9. ¿En qué parte del programa puede ser utilizada una variable global?

10. Se realiza la operación: c= 10%4, ¿cuál es el valor de c?

11. En la operación x+=2; ¿cuánto es el resultado de x?

12. ¿Cuál es el resultado de la operación x= x>>2, si x= 25? Realice las operaciones en binario.

13. ¿Cuál es el resultado de la operación x= x<<4, si x= 63? Realice las operaciones en binario.

14. Explique la diferencia entre realizar la operación:
a. a&b y a&&b.
b. a|b y a||b.

15. En el siguiente código

```
if (x < 18)
        printf("Eres menor de edad.\n");
else
        printf("Eres un joven.\n");
```

Sea x= 25, ¿qué mensaje se imprime?

16. En el siguiente ejemplo:
a. ¿Para qué sirve break?
b. Si lista=0, ¿qué mensaje se imprime?

```
Switch( lista)
{
  case 0: printf("No hay elementos.");
```

```
  break;
  case 1: printf("Hay solo un elemento.");
  break;
  default: printf("Hay varios elementos");
}
```

17. Usando while escriba la forma para crear un bucle infinito.

18. Usando for escriba la forma para crear un bucle infinito.

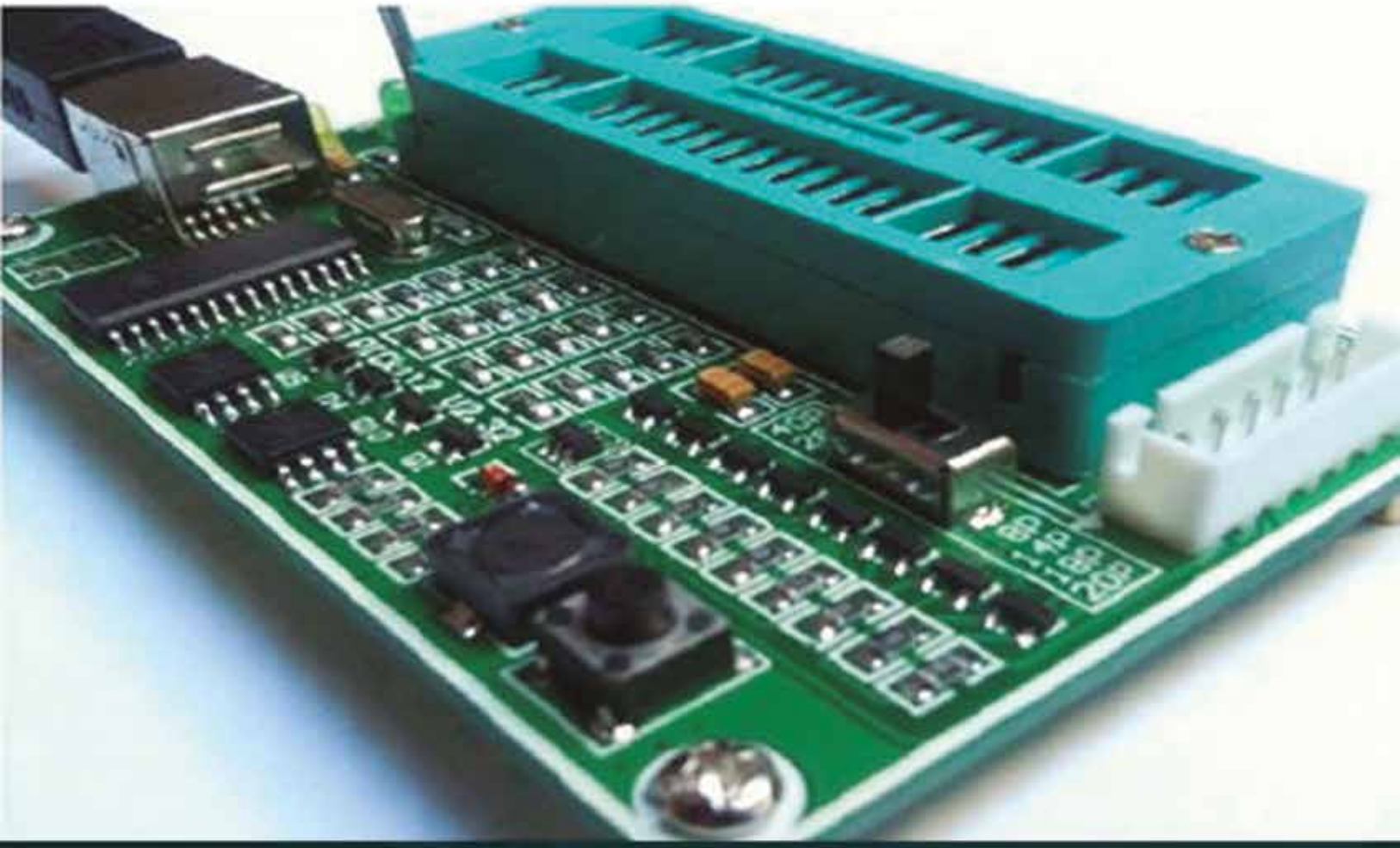
19. Escriba la instrucción usando for para que una variable entrada cuente del 0 al 99.

20. ¿Qué tiempo de retardo genera delay_ms(1500)?

21. De los microcontroladores citados en este capítulo, ¿cuáles no tienen reloj interno?

22. Cuál microcontrolador de 18 pines tiene memoria FLASH de 4096 words y posee un módulo conversor de AD de 7 canales.

23. ¿Cuáles series de microcontroladores PIC tienen un módulo de conexión USB?



CAPÍTULO 2

PROGRAMACIÓN DEL MICROCONTROLADOR PIC

Los fuses del microcontrolador PIC18

Todos los microcontroladores PIC tienen incorporados circuitos especiales que se pueden activar o desactivar según los requerimientos de la aplicación o algún caso en particular. Estos circuitos o características especiales se deben configurar mediante software o por el equipo programador al momento de grabar el programa en el microcontrolador. La configuración adecuada de estos circuitos determina en gran parte la funcionabilidad y eficiencia del programa desarrollado.

Los ***fuses*** permiten entre otras cosas seleccionar el oscilador: interno o externo que se va utilizar, la frecuencia de trabajo del sistema, el uso del Master Clear (reset del sistema), etc. Los *fuses* internamente en el microcontrolador son controlados por registros internos y que se pueden habilitar o deshabilitar mediante software o por medio del equipo programador. CCS provee de instrucciones de alto nivel que permiten de forma transparente configurar los FUSES (fusibles). En el PIC18F4550, los *fuses* que CCS configura se muestra en la tabla 2.1. Para no distorsionar la descripción del *fuse*, se ha dejado en el idioma original, que CCS proporciona en el IDE del programa.

FUSE	DESCRIPCIÓN
LP	Low power osc < 200 kHz
XT	Crystal osc <= 4mhz for PCM/PCH , 3mhz to 10 mhz for PCD
HS	High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
RC	Resistor/Capacitor Osc with CLKOUT
EC	External clock with CLKOUT
EC_IO	External clock
H4	High speed osc with HW enabled 4X PLL
RC_IO	Resistor/capacitor osc
RC	Resistor/Capacitor Osc with CLKOUT
INTRC_IO	Internal RC Osc, no CLKOUT
INTRC	Internal RC Osc
NOFCMEN	Fail-safe clock monitor disabled
FCMEN	Fail-safe clock monitor enabled
IESO	Internal External Switch Over mode enabled
NOIESO	Internal External Switch Over mode disabled
PUT	Power up timer
NOPUT	No power up timer
BROWNOUT	Reset when brownout detected

NOBROWNOUT	No brownout reset
BROWNOUT_SW	Brownout controlled by configuration bit in special file register
B R O W N O U T _ NOSL	Brownout enabled during operation, disabled during SLEEP
BORV45	Brownout reset at 4.5V
BORV42	Brownout reset at 4.2V
BORV27	Brownout reset at 2.7V
BORV20	Brownout reset at 2.0V
NOWDT	No watch dog timer
WDT	Watch dog timer
WDT128	Watch Dog Timer uses 1:128 Postscale
WDT64	Watch Dog Timer uses 1:64 Postscale
WDT32	Watch Dog Timer uses 1:32 Postscale
WDT8192	Watch Dog Timer uses 1:8192 Postscale
WDT16384	Watch Dog Timer uses 1:16384 Postscale
WDT16	Watch Dog Timer uses 1:16 Postscale
WDT8	Watch Dog Timer uses 1:8 Postscale
WDT256	Watch Dog Timer uses 1:256 Postscale
WDT512	Watch Dog Timer uses 1:512 Postscale
WDT1024	Watch Dog Timer uses 1:1024 Postscale
WDT2048	Watch Dog Timer uses 1:2048 Postscale
WDT4	Watch Dog Timer uses 1:4 Postscale
WDT32768	Watch Dog Timer uses 1:32768 Postscale
WDT4096	Watch Dog Timer uses 1:4096 Postscale
WDT1	Watch Dog Timer uses 1:1 Postscale
WDT2	Watch Dog Timer uses 1:2 Postscale
CCP2B3	CCP2 input/output multiplexed with RB3
CCP2C1	CCP2 input/output multiplexed with RC1
NOPBADEN	PORTB pins are configured as digital I/O on RESET
PBADEN	PORTB pins are configured as analog input channels on RESET
LPT1OSC	Timer1 configured for low-power operation
NOLPT1OSC	Timer1 configured for higher power operation
NOMCLR	Master Clear pin used for I/O
MCLR	Master Clear pin enabled
STVREN	Stack full/underflow will cause reset
NOSTVREN	Stack full/underflow will not cause reset
NOLVP	No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
LVP	Low Voltage Programming on B3(PIC16) or B5(PIC18)

NOXINST	Extended set extension and Indexed Addressing mode disabled (Legacy mode)
XINST	Extended set extension and Indexed Addressing mode enabled
NODEBUG	No Debug mode for ICD
DEBUG	Debug mode for use with ICD
PROTECT	Code protected from reads
NOPROTECT	Code not protected from Reading
NOCPB	No Boot Block code protection
CPB	Boot block code protected
CPD	Data eeprom code protected
NOCPD	No EE protection
WRT	Program memory write protected
NOWRT	Program memory not write protected
WRTC	Configuration registers write protected
NOWRTC	Configuration not registers write protected
NOWRTB	Boot block not write protected
WRTB	Boot block write protected
WRTD	Data EEPROM write protected
NOWRTD	Data EEPROM not write protected
NOEBTR	Memory not protected from table reads
EBTR	Memory protected from table reads
EBTRB	Boot block protected from table reads
NOEBTRB	Boot block not protected from table reads

Tabla 2.1. FUSES para el PIC184550 del compilador CCS.
Fuente: CCS Compiler V4.084.

Algunos ***fuses*** de interés significan:

LVP: Low Voltage programming. Programación por voltaje bajo.

- 1 = LVP habilitado, la terminal RB4/PGM tiene tal función.
- 0 = LVP deshabilitado, RB4/PGM es una terminal de I/O

BODEN: Brown Out Detect Reset Enable bit (Bit de reset por voltaje de alimentación bajo). En CCS el fusible es BROWNOUT NOBROWNOUT**.**

- 1 = Reset por BOD habilitado.
- 0 = Reset por BOD deshabilitado.

BOR: Brown-out Reset. Este circuito hace que el PIC se reinicie cuando se detecta una inestabilidad en la alimentación. El compilador proporciona

varios FUSES para configurar el BOR para diferentes niveles de tensiones (ver tabla 2.1). Por ejemplo: BORV45 reinicia el microcontrolador en nivel de 4.5V.

PBADEN y NOPBADEN: Cuando se produce un RESET en el Micro, con PBADEN los pines del puerto B se configuran como canales de entrada analógicos y con NOPBADEN los pines del puerto B se configuran como entradas o salidas digitales.

MCLR: Master Clear. Habilitación/deshabilitación del terminal de reset.

- 1 = Terminal de reset (MCLR) en RA5.
- 0 = MCLR conectado internamente a Vdd, RA5 es un pin de I/O.

PUT: Power up Timer enable bit. En el datasheet del microcontrolador es el bit que se conoce como PWRT. Es el bit de habilitación de temporizador al energizar. Al activar el POWER UP TIMER lo que hace es retardar el arranque del PIC. Este empieza a funcionar unos 65.5 ms., después de recibir la tensión de alimentación.

- 1 = PWRT habilitado.
- 0 = PWRT deshabilitado.

WDT: Watch Dog Timer Enable Bit. Bit de habilitación del Watch-Dog.

- 1 = WDT habilitado.
- 0 = WDT deshabilitado.

Para configurar los fusibles se debe incluir en el programa la directiva *#fuses,* seguido de los nombres y separados entre comas (,) de los fuses que se necesiten fijar en el programa. Si no se considera algunos de ellos el programa asume el valor por default que suministra el fabricante, mismo que podemos ver en el datasheet del microcontrolador. Ejemplo:

Suponer que se va a utilizar un cristal de 4 MHz, desactivar el Master Clear, No proteger el código para lectura, desactivar la programación a bajo voltaje; el código del compilador CCS para los FUSES mencionados es:

#fuses XT, NOWDT, NOPROTECT, NOLVP //Configuración de Fuses.

Nota: Se debe indicar que el compilador acepta mayúsculas o minúsculas al escribir las instrucciones, funciones u otras sentencias para el desarrollo de los programas.

Configuración del reloj

La mayoría de los microcontroladores PIC de la familia 16F y 18F tienen incorporado un sistema de reloj para ser utilizado según las necesidades del diseño. El hardware de control del oscilador es el registro OSCCON (Oscillator Control Register) y el registro oscilador de ajuste OSCTUNE (Oscillator Tuning Register), se utiliza para ajustar frecuencia interna vía software.

Tipos de osciladores

Según el datasheet del fabricante, los dispositivos PIC18F2455 / 2550/4455/4550 y PIC16F88, PIC16F887, pueden ser operados en distintos modos de oscilador, siendo estos los más usados:

- XT Crystal / Resonador (se usa para cristales <= 4 MHZ).
- HS alta velocidad Crystal / Resonador (>4MHZ).
- HSPLL alta velocidad Crystal / Resonador con PLL activado.
- EC reloj externo con F_{OSC} / 4 de salida.
- ECIO reloj externo con I / O en RA6.
- ECPLL Reloj externo con PLL Activado y F_{OSC} / 4 Salida en RA6.
- ECPIO Reloj externo con PLL habilitado, I/O en RA6.
- INTHS oscilador interno utilizado como
    - Microcontroladores fuente de reloj, HS
    - Oscilador utilizado como USB Fuente de reloj
- INTIO oscilador interno utilizado como:
    - Microcontrolador fuente de reloj, EC
    - Oscilador utilizado como fuente de reloj USB,
    - I/O digital en RA6
- INTCKO oscilador interno utilizado como
    - Microcontrolador fuente de reloj, EC
    - Oscilador utilizado como fuente de reloj USB,
    - F_{OSC} / 4 Salida en RA6

El PIC16F628A tiene los modos LP, XT, HS, ECIO, INTIO, INTCKO.

La tabla 2.2, indica los valores típicos del cristal y los capacitores que el fabricante sugiere utilizar con los diferentes cristales para modo XT y HS. Mayor información sobre los valores del oscilador de cristal y los capacitores

para los resonadores cerámicos se pueden encontrar en las hojas del datasheet del fabricante.

Osc Type	Crystal Freq	Typical Capacitor Values Tested:	
		C1	C2
XT	4 MHz	27 pF	27 pF
HS	4 MHz	27 pF	27 pF
	8 MHz	22 pF	22 pF
	20 MHz	15 pF	15 pF

Tabla 2.2. Selección de capacitores para el oscilador de cristal.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

El oscilador se conecta en los pines RA6/OSC2/CLKO y OSC1/CLK1. La figura 2.1, muestra el diagrama de conexiones del oscilador de cristal de 8MHz y los capacitores cerámicos de 22 pF, como recomienda el fabricante.

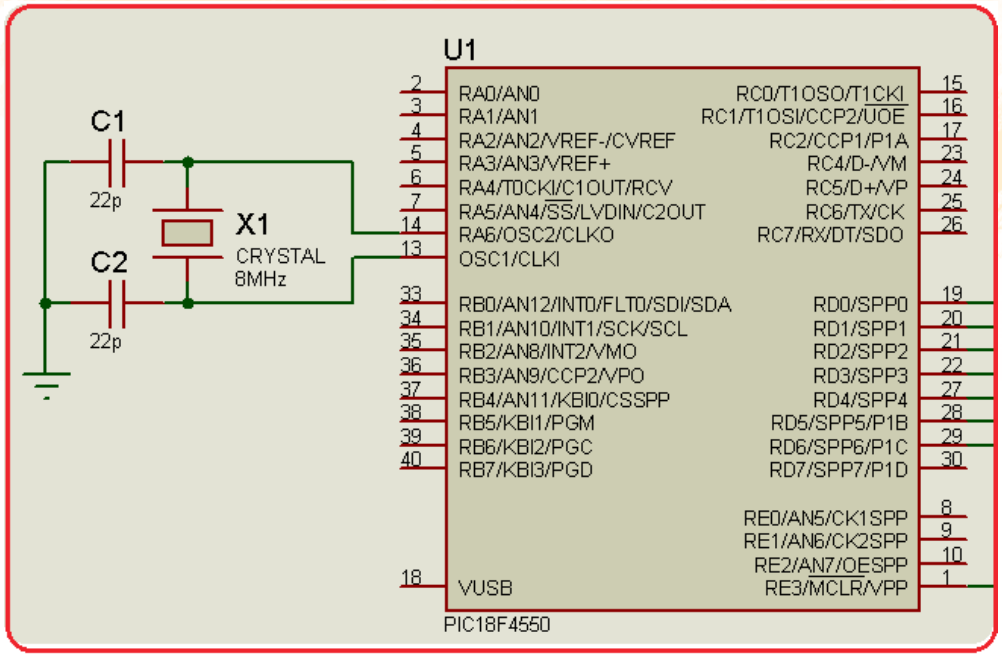


Figura 2.1. Conexión del oscilador de cristal externo.
Elaborado por los autores.

Todas las frecuencias se derivan de 8MHZ y con el registro POSTSCALER de 8 bits se obtienen el resto de frecuencias (8MHz/256= 31.25kHz) y son seleccionables por software. La figura 2.2, indica los valores de las frecuencias internas que pueden ser usadas.

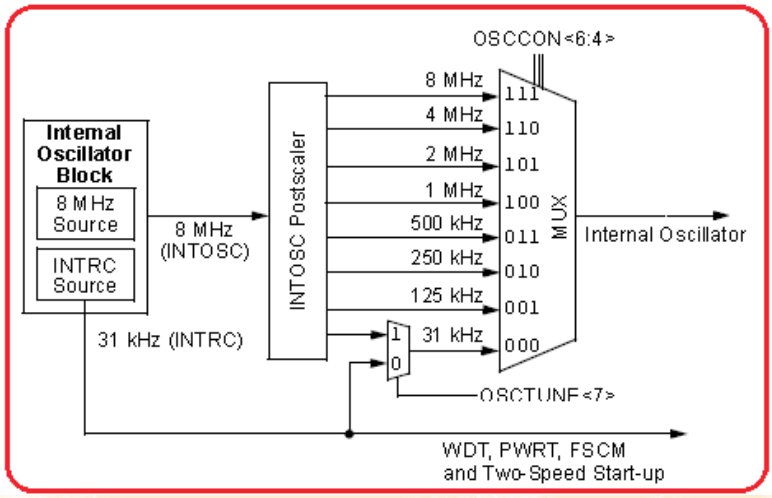


Figura 2.2. Diagrama de bloques del oscilador interno y la gama de frecuencias derivadas. *Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Tecnhology INC.*

Mediante la configuración de los FUSES se puede utilizar el oscilador externo o la fuente de frecuencia interna. El compilador provee una serie FUSES para configurar el reloj interno.

Configuración del oscilador

La instrucción para utilizar el tipo de oscilador es:

#fuses TIPO DE OSCILADOR

Donde el TIPO DE OSCILADOR hace referencia a los descritos en este punto. Ejemplo:

#fuses HS
#use delay (clock=12000000)

Se indica al microcontrolador que utilizará un oscilador externo de alta velocidad (HS) con un cristal de 12000000 Hz. Para indicar directamente la frecuencia en MHz, se puede utilizar la notación:

#use delay (clock=12M)

Para el caso que utilicemos la fuente de reloj interno del microcontrolador, se utilizaría así:

#fuses INTRC_IO
#use delay (clock=8M)

Se indica al compilador que se va a utilizar el oscilador interno RC y el pin RA6, se configura para que sea utilizado como entrada o salida digital. Es válido también en algunos Micros que tienen el puerto RA7, o sea RA6 y RA7, se configuran como I/O digitales.

La instrucción *#use delay (clock=8M),* indica al compilador usar frecuencia de reloj para el sistema de 8MHz. En la figura 2.2, se pueden observar los valores de reloj internos que se pueden utilizar.

Si se utiliza la configuración del fusible:

#fuses INTRC.

Se indica al compilador use el reloj interno y por el terminal RA6 se obtendrá una salida de reloj del sistema interno (F_{OSC} /4).

Otra forma de fijar el oscilador interno es mediante la función ***setup_oscillator().*** Esta función hace referencia al registro *OSCCON* del microcontrolador. La función controla y devuelve el estado del oscilador RC interno en los micros que disponen del registro *OSCCON*. Se debe tener en cuenta que si *INTRC* o *INTRC_IO* se especifica en *#fuses* y un retraso *#use* se utiliza para una opción de velocidad válido, el compilador hará esta configuración automáticamente al inicio del *main ()*.

La sintaxis y argumentos de la función son:

setup_oscillator(mode, finetune)

Dónde:

mode, es dependiente en el chip. Por ejemplo, algunos chips permiten ajuste de velocidad tales como OSC_8MHZ u OSC_32KHZ. Otros micros permite el cambio del origen como OSC_TIMER1.

finetune, es el ajuste preciso, es un variable entera que va desde -31 a +31. Solo se permiten en microcontroladores que disponen del registro OSCTUNE.

Algunos chips devuelven un estado tal como OSC_STATE_STABLE para indicar el oscilador es estable. La función descrita debe siempre colocarla dentro del *main ()* del programa. Ejemplo:

```
void main()
{
setup_oscillator(OSC_8MHz|OSC_INTRC|OSC_PLL_OFF);
}
```

Conexión de LEDS en los puertos

La figura 2.3, muestra la conexión de diodos LEDS al puerto A, B, C, D y E del microcontrolador. Hay que tomar en cuenta que la máxima corriente que puede entregar o recibir los puertos del microcontrolador es de 25 mA, según las características eléctricas que Microchip establece para el PIC18F4550. Por tanto, tomando en consideración el valor indicado, se ha colocado resistencias de 330 Ω, en las líneas del puerto para limitar la corriente y evitar dañar el puerto. Esta conexión es común para todos los puertos excepto para el pin RA4, que en algunos de los microcontroladores son de drenador abierto.

Dado que la caída de tensión en el LED cuando está conduciendo es de aproximadamente 1.3 V, la corriente de cada pin del puerto es:

$$I_{PIN} = \frac{V_{DD} - V_{LED}}{R}$$

$$I_{PIN} = \frac{5\,V - 1.3\,V}{330\Omega}$$

$$I_{PIN} = 11.21\ mA.$$

El valor obtenido es inferior a los 25 mA y está en el rango (5 a 15 mA) que necesita un LED para iluminar adecuadamente.

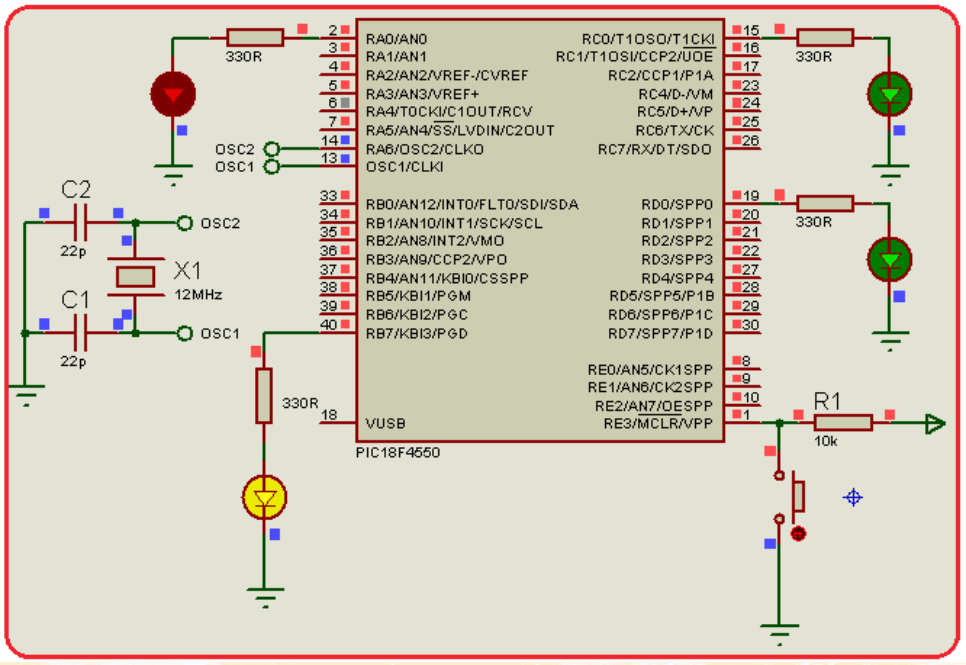


Figura 2.3. Conexión de LEDS en los puertos del microcontrolador.
Elaborado por los autores.

Gestión de puertos

En el compilador CCS se pueden gestionar los puertos de dos formas:

- Definiendo la posición de la memoria RAM como una variable C para los registros TRISX y PORTX.
- Usando la directivas específicas del compilador (#USE FAST_IO, #USE FIXED_IO, #USE STANDARD_IO)

Usando la memoria RAM

Las posiciones de memoria que utilizan los registros TRISX y PORTX del PIC18 se identifican así:

#BYTE tris_n = Dir

#BYTE port_n = Dir

Donde, ***n*** es el nombre del puerto (A, B, C, D, E) y la variable ***Dir*** la dirección en la memoria RAM.

Ejemplo:

#BYTE tris_b = 0xF93

#BYTE port_b = 0xF81

La tabla 2.3, muestra las direcciones de memoria ocupadas por los registros TRIS en la familia de PIC18FXXX. En todo caso, siempre es necesario consultar el datasheet del fabricante para verificar la dirección exacta que ocupa los registros TRIS en la memoria RAM.

REGISTRO TRIS	DIRECCIÓN DE MEMORIA
A	0xF92
B	0xF93
C	0xF94
D	0xF95
E	0xF96

Tabla 2.3. Localidades de memoria ocupados por los registros TRIS, para los PIC18FXXX.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Tecnhology INC.

La tabla 2.4, indica las direcciones de la memoria RAM que son asignadas para el registro PORT de los PIC de la serie PIC 18F4550. Esta información se encuentra en las hojas de datos técnicos que proporciona el fabricante de microcontroladores PIC, Microchip Technology INC.

REGISTRO PORT	DIRECCIÓN DE MEMORIA
A	0xF80
B	0xF81
C	0xF82
D	0xF83
E	0xF84

Tabla 2.4. Localidades de memoria ocupados por los registros PORT, para los PIC18FXXX.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Tecnhology INC.

Para las series de microcontroladores PIC 16F8XX y 16F6XX, las direcciones del registro TRIS y PORT son:

REGISTRO TRIS	DIRECCIÓN DE MEMORIA
A	0x85
B	0x86
C	0x87
D	0x88
E	0x89

Tabla 2.5. Localidades de memoria ocupados por los registros TRIS, para los PIC16F.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Tecnhology INC.

REGISTRO PORT	DIRECCIÓN DE MEMORIA
A	0x05
B	0x06
C	0x07
D	0x08
E	0x09

Tabla 2.6. Localidades de memoria ocupados por los registros PORT, para los PIC16F.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Tecnhology INC.

Definidas las variables se pueden controlar los puertos por comandos de asignación. Ejemplos:

- ***tris_a = 0xFF ;*** Define todo el **puerto a** como entrada
- ***tris_b = 0x00;*** Define todo el **puerto b** como salida
- ***tris_d = 0x0F;*** Define los 4 primeros bits del **puerto d** como entrada y los 4 bits (MSB) como salida.

Luego se escribe o lee los datos en el puerto.
Escritura del puerto:
port_b = 0x0B; El dato 1011, se coloca en el puerto b.
Lectura del puerto:
Contador = port_a; El dato del puerto a se asigna a la variable Contador.

Ejercicio 2.1. Prender y apagar un LED conectado en el puerto B0.

El primer programa de computadora que enseñan en programación y en la mayoría de lenguajes es imprimir en la pantalla *"Hola Mundo"*. El primer programa semejante que se realiza con un microcontrolador es prender y apagar un LED indefinidamente.

En el programa primero debemos definir el microcontrolador a utilizarse, para lo cual se utiliza la directiva:

#include <18F4550.h> //Librería para usar el PIC18F4550.

Ahora se procede a configurar los fusibles. La habilitación o deshabilitación de los fusibles dependerá de la aplicación que se esté desarrollando, pero el fusible que siempre se debe incluir es el reloj que se utilice. En este caso definimos un cristal de alta velocidad *HS*.

#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.

Es importante también definir la frecuencia del oscilador:

#use delay (clock=12000000) // F_{OSC} =12MHz.

En este programa el LED se conecta en el puerto RB0. Se define el bit a utilizar mediante la instrucción *#bit b0= 0xF81.0.* La variable bit b0 se puede asignar valores de 0 y 1 para obtener valores lógicos de tensión de 0 y 5V respectivamente en el puerto RB0. Por tanto, en el programa las asignaciones de *b0= 0;* establecen que el puerto RB0 sea igual a 0V y *b0= 1;* en el puerto RB1 tendrá el nivel lógico de 5V.

Cuando no se utiliza las interrupciones, es conveniente deshabilitar todas las interrupciones, sin que sea una exigencia de alta prioridad. La instrucción *disable_interrupts (GLOBAL);* realiza esta función.

El registro *TRISB* está definido en la localidad 0xF93 como indica en las hojas técnicas del microcontrolador PIC18F4550.

Otro aspecto importante que se debe tomar en cuenta dentro de la programación es la definición del puerto como salida. La instrucción *tris_b = 0x00;* configura el puerto como salida, para que permita establecer los niveles lógicos de 0 y 1.

PROGRAMA:

```
#include <18F4550.h> //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000) // F_OSC =12MHz.
#byte tris_b = 0xF93          //Define el registro tris b en la localidad 0xF93.
#bit b0= 0xF81.0              //Define el bit b0 en la localidad 0xF81.0.

void main()                   //Función principal main.
{
disable_interrupts (GLOBAL);  //Deshabilita las interrupciones globales.
tris_b = 0x00;                //Configura el puerto B como salida.

do{                           //Inicio del bucle infinito.
  b0= 0;                      //El puerto RB0 = 0.
  delay_ms( 500 );            //Retardo de 500 ms.
  b0 =1;                      //El Puerto RB0 = 1.
  delay_ms(500);              //Retardo de 500 ms.
  } while ( TRUE );           //Bucle infinito.
}                             //Fin del bucle main.
```

El circuito para comprobar el funcionamiento se indica en la figura 2.4. Como en la configuración de los FUSES no está definido el uso del MCLR, es necesario conectar este terminal a través de una resistencia a V_{CC}, el pulsador permitirá tener un botón de RESET.

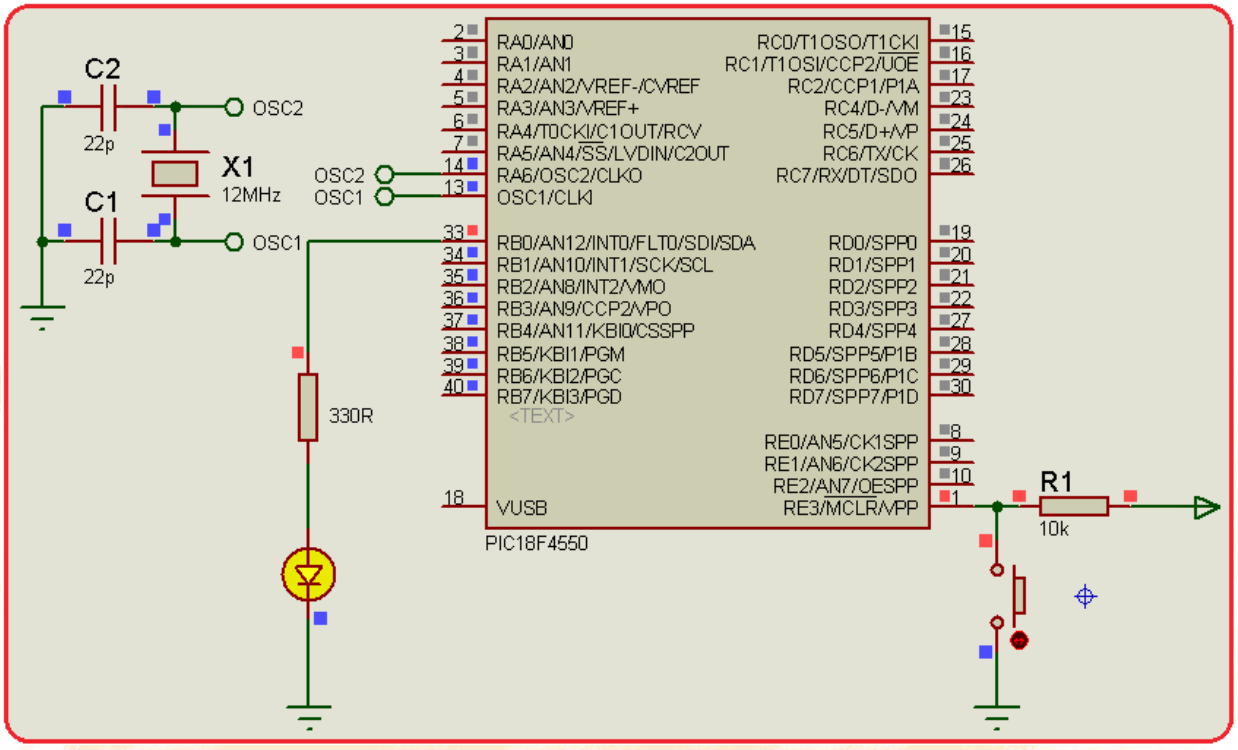


Figura 2.4. Conexión de un LED en el puerto RB0.
Elaborado por los autores.

Ejercicio 2.2. Prender y apagar los LEDS del puerto B.

En el ejercicio 2.2, los LEDS conectados al puerto B se prenden y apagan indefinidamente, como indica la figura 2.5. En este caso en el programa todos los ocho bits de la localidad de memoria *0xF81* se asignan a la variable *port_b*, de igual manera ocurre con el registro *tris_b*, con la localidad de memoria *0xF93*.

PROGRAMA:

```
#include <18F4550.h> //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)  / / F_OSC = 12MHz.
#byte port_b = 0xF81          //Identificador para el puerto b en  la localidad 0xF81.
#byte tris_b = 0xF93          //Identificador para el registro tris b en la localidad 0xF93.

void main()                   //Función principal main.
{
  disable_interrupts(GLOBAL); //Deshabilita todas las interrupciones globales.
  tris_b = 0x00;              //Configura todo el puerto B como salida.
  port_b = 0x00;              // Asigna a todo el Puerto B cero (0).
do{                           // Hace o empieza el bucle infinito.
  port_b = 0xFF;              //Coloca todo el puerto B en 1.
  delay_ms( 500 );            //Retardo de 500 ms.
  port_b = 0x00;              //Coloca todo  el puerto B en 0.
  delay_ms(500);              //Retardo de 500 ms.
  } while ( TRUE );           //Bucle infinito.
}                             //Fin del bucle main.
```

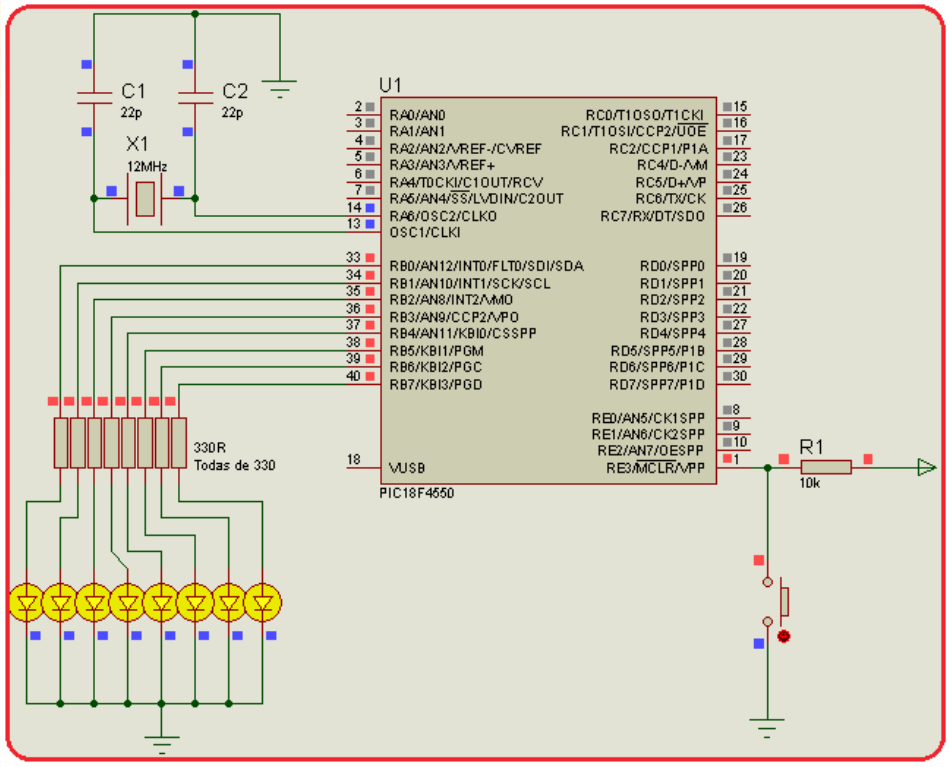


Figura 2.5. LEDS conectados en el puerto B.
Elaborado por los autores.

Con el programa propuesto y el circuito indicado los LEDS se prenden por 500 ms y se apagan por un tiempo igual. La ejecución de la secuencia de prendido y apagado es indefinido, el bucle **DO – WHILE** en la función principal ***main()*** se encarga de ejecutar las instrucciones en bucle infinito.

Otra forma de generar un bucle infinito es utilizando la sentencia ***while (TRUE)*** o ***while(1)*** como se indica en el ejercicio 2.3.

Ejercicio 2.3. Prender y apagar los LEDS que están conectados al puerto B indefinidamente.

La mayoría de programas que se desarrollan en C, para generar los bucles infinitos se utiliza la sentencia *while(TRUE)*, por lo que en este programa se utilizará esta forma.

PROGRAMA:

```
#include <18F4550.h> //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000) // F_OSC = 12MHz.
#byte port_b = 0xF81          //Identificador para el puerto b en  la localidad 0xF81.
#byte tris_b = 0xF93          //Identificador del registro tris b en la localidad 0xF93.

void main()                   //Función principal main.
{
 disable_interrupts(GLOBAL); //Deshabilita todas las interrupciones globales.
 tris_b = 0x00;               // Configura el puerto B como salida.
 port_b = 0x00;               // Asigna a todo el Puerto B cero (0).
 while (TRUE){                //Bucle infinito.
   port_b = 0xFF;             //Coloca todo el puerto B en 1.
   delay_ms( 500 );           //Retardo de 500 ms.
   port_b = 0x00;             //Coloca todo  el puerto B en 0.
   delay_ms(500);             //Retardo de 500 ms.
 }                            //Fin del bucle infinito.
}                             //Fin del bucle main.
```

A través de directivas

El compilador ofrece funciones predefinidas para trabajar con los puertos. Estas directivas son:

- ***output_x (value);*** por el puerto indicado saca el valor correspondiente a 0 a 255.

- ***input_x();*** se obtiene el valor del puerto.
- ***set_tris_x(value);*** configura el puerto ***x*** como entrada o salida dependiendo de la variable ***value***. Ejemplo:

  set_tris_b(0x00); el puerto b es de salida.

  set_tris_b(0xFF); el puerto b es de entrada.
- ***get_tris_x();*** devuelve el valor del registro tris_x.

  x, es el símbolo correspondiente a los puertos A, B, C, D, E.

Las funciones asociadas al terminal o pin son:

- ***output_low (PIN_n);*** pin a 0.
- ***output_high (PIN_n);*** pin a 1.
- ***output_bit (PIN_n,value);*** *p*in al valor especificado.
- ***output_toggle (PIN_n);*** complemento el valor del pin.
- ***output_float (PIN_n);*** pin de entrada, quedando a tensión flotante. Simula salida de drenador abierto.
- ***output_state (PIN_n);*** lee el valor del pin.
- ***input_state(PIN_n);*** lee el valor del PIN sin cambiar el sentido del terminal.
- ***input(PIN_n);*** lee el valor del PIN.

  n, se refiere al nombre del puerto y/o del PIN.

Directiva #use standard_io (puerto)

Es la directiva por defecto. Las funciones *output_x e input_x,* modifican el registro TRIS asegurando que los terminales usados sean de salida o entrada respectivamente.

Si no se incluye ninguna directiva, el programa tomará por defecto *#use standard_io(puerto)*, que reprogramará cada vez al pin como entrada o salida aumentando el código generado por el compilador.

Al usar las directivas no es necesario identificar las direcciones que ocupan los registros *PORT* y *TRIS,* porque estos ya están definidos en las directivas.

Ejercicio 2.4. Prender y apagar un LED por el puerto RB0.

Este programa trabaja de la misma forma del ejercicio 2.1. En este caso se utiliza las funciones de la directiva *#USE STANDARD_IO(PUERTO)* para

definir el puerto a usarse. Nótese que ya no se Define *port_x* de entrada o salida con *TRIS*. El circuito de figura 2.4, se puede utilizar para demostrar el funcionamiento.

PROGRAMA:

```
#include <18f4550.h>   /          /Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)   //FOSC=12MHz.
#use standard_io(b)           //Directiva usado por el compilador por defecto.
                              //Es opcional llamarla en el programa.
void main(void)               //Función principal main.
{
disable_interrupts(GLOBAL);   //Todas las interrupciones desactivadas.

while(TRUE);                  //Bucle infinito.
{                             //Inicio del bucle.
  output_low(PIN_B0);         //LED off.
  delay_ms(500);              //Retardo de 500 ms.
  output_high(PIN_B0);        //LED on.
  delay_ms(500);              //Retardo de 500 ms.
 }                            //Fin del bucle infinito.
}                             //Fin del bucle main.
```

b. Directiva #use fast_io(port)

Esta directiva define a los puertos como entradas o salidas digitales. La variable ***port*** puede tomar los valores de ***a***, ***b***, ***c***, ***d***, ***e***, o ***all*** para señalar que todos los puertos se utilizarán como entradas y salidas digitales. Las funciones *output_x e input_x,* no modifican el registro *TRIS,* por lo cual se debe colocar necesariamente la instrucción para definir que los terminales usados sean de salida o entrada respectivamente.

Al utilizar esta directiva, el compilador optimiza el código generado y ahorra memoria de programa. El uso de las funciones *output* e *input* no reprograman los puertos ya que quedan definidos en la instrucción *TRIS*. Por tanto, la respuesta de los puertos para conmutaciones rápidas se ve mejorado con el uso el uso de la directiva #*use fast_io(port)*.

Ejercicio 2.5. Prende y apaga LED por el puerto RB0.

En el programa para definir el estado en alto del puerto se utiliza la función *output_high(PIN_B0)* y el estado en bajo la función *output_low(PIN_B0)*. El registro *TRIS* no identifica con la dirección de memoria respectiva y se hace referencia directamente para fijar al puerto B como salida con la instrucción *set_tris_b (00x00)*.

PROGRAMA:

```
#include <18f4550.h>   /          /Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)   //FOSC=12MHz.
#use fast_io(b)               //Define el puerto b como i/o digital.

void main(void)               //Función principal main.
{
 set_tris_b(0x00)             //Define port_b como salida.
 disable_interrupts(GLOBAL);  //Todas las interrupciones desactivadas.
 while(TRUE);                 //Bucle infinito.
   {                          //Inicio del bucle.
   output_low(PIN_B0);        //LED off.
   delay_ms(500);             //Retardo de 500 ms.
   output_high(PIN_B0);       //LED on.
   delay_ms(500);             //Retardo de 500 ms.
    }                         //Fin del bucle infinito.
}                             //Fin del main.
```

c. Directiva #use fixed_io(puerto_outputs=pinx)

Con la directiva #*use fixed_io(puerto_outputs=pinx)*, cuando se utilizan las funciones *input* y *output* los puertos son reprogramados como entradas y salidas por lo que el código generado es mayor. Sin embargo, se ahorra una posición de memoria RAM por cada puerto utilizado.

Ejemplo: #use fixed_io(b_outputs=pin_b0, pin_b1)

Ejercicio 2.6. Prender y apagar un LED.

PROGRAMA:

```
/*  MANEJO DEL PUERTO  USANDO DIRECTIVA #USE FIXED  */
#include <18F4550.h>   /          /Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)    //FOSC=12MHz.
#use fixed_io(b_outputs=pin_b0)          //Define el puerto b0 como salida.

void main()                              //Función principal main.
{
disable_interrupts(GLOBAL);   //Desactiva todas las interrupciones.

while (TRUE)                  //Bucle infinito.
  {                           //Inicio del bucle.
  output_high(pin_b0);        //Puerto RB0 en alto.
  delay_ms( 500 );            //Retardo de 500 ms.
  output_low(pin_b0);         //Puerto RB0 en bajo.
  delay_ms(500);              //Retardo de 500 ms.
  }                           //Fin del bucle infinito.
}                             //Fin del bucle main.
```

Secuencias con LEDS

En muchas aplicaciones como en luces de discotecas, decoraciones de locales, etc., se necesita controlar luces que deben un efecto visual atrayente. Utilizando el microcontrolador se puede realizar una serie de proyectos de este tipo.

Ejercicio 2.7. Secuencia de encendido y apagado de LEDS.

El ejercicio siguiente, consiste en activar y desactivar los LEDS que están conectados al puerto uno a uno empezando desde el puerto RB0 hasta llegar al RB7 en forma indefinida.

Para clarificar el funcionamiento y la secuencia de los bits que se desea implementar, la tabla 2.7, muestra el estado del puerto b, en cada uno de los bits que forman el puerto.

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

Tabla 2.7. Secuencia de encendido y apagado del puerto B.
Elaborado por los autores.

Para realizar el programa, se utiliza la siguiente lógica de programación:

- Inicializar el puerto B con (00000001), para activar el LED que está conectado en RB0.
- Mostrar este valor durante un tiempo (100 ms).
- Desplazar el valor del puerto una posición a la izquierda *(port_b = port_b<<1)*.
- Repetir este proceso hasta que llegue el 1 al puerto RB7. Mediante el bucle for se controla que el bit 1 que está en puerto RB0, se desplace hasta llegar al puerto RB7. Como el puerto tiene ocho bits, entonces el desplazamiento se debe repetir 7 veces. La instrucción que realiza esta función es: *for (i=1;i<=8;++i)*.
- Al llegar el bit a la posición 7 el valor del puerto será 128 (2^7=128). A partir de este valor, en este punto, el desplazamiento se realiza a la derecha *(port_b=port_b>>1)* hasta que el bit llegue a RB0.
- Repetir el proceso.

PROGRAMA:

```
#include <18f4550.h>  //Librería para usar el PIC18F4550
#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)   //F_OSC=12MHz
#BYTE port_b= 0xF81  //Identificador para el puerto b en  la localidad 0xF81.
int i;                        //Variable i tipo int para el bucle FOR.

void main(void) {             //Función principal main
set_tris_b(0x00);             //Puerto B como salida
disable_interrupts(GLOBAL);   //Todas las interrupciones desactivadas

while(TRUE)                   //Bucle infinito
{                             //Inicio del bucle.
  port_b=0x01;                //Puerto RB0 = 1
  for (i=1;i<=8;++i)          //Realiza el bucle para i = 0 hasta que sea menor o igual 8 en
                              //incrementos de 1.
  {
    delay_ms(100);            //Retardo de 100 ms
    port_b= port_b<<1;        //Desplaza el valor del puerto a la izquierda en 1
  }
   port_b= 128;               //port_b = 128.
   for (i=1;i<=8;++i)         //Realiza el bucle para i = 0 hasta que sea menor o igual
                              //8 en incrementos de 1.
  {
    delay_ms(100);            //Retardo de 100 ms
    port_b= port_b>>1;        //Desplaza el valor del puerto a la derecha en 1

  }
 }                            //Fin del bucle infinito.
}                             //Fin del bucle main.
```

Conexión en el Puerto A

En algunas familias de los PIC16F y 18F el puerto RA4, es de drenador abierto, lo cual significa que en la salida se debe conectar una resistencia en serie con el LED y luego a V_{DD}. Para los demás pines del puerto A la conexión es normal, como indica en la figura 2.6.

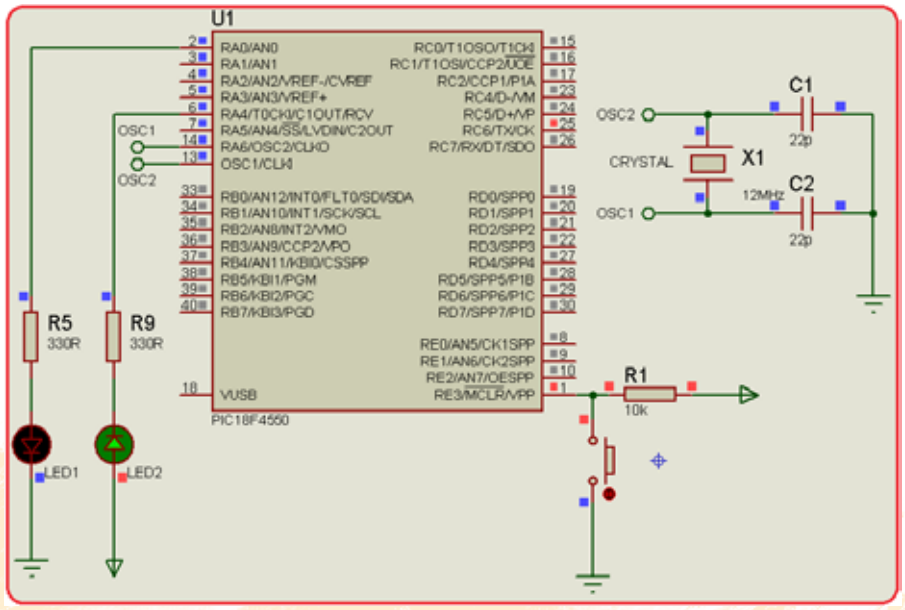


Figura 2.6. Conexión de LED en puerto RA4.
Elaborado por los autores.

Debido a la conexión, cuando el puerto RA4 tenga un nivel alto el LED2 se apaga y en un nivel bajo se prende.

Ejercicio 2.8. Manejo de LEDS con el puerto A.

PROGRAMA:

```
#include <18f4550.h>  //Librería para usar el PIC18F4550
#fuses HS,NOWDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000) // F_OSC =12MHz
#BYTE port_a= 0xF80  //Identificador para el puerto a en  la localidad 0xF80.

void main()                    //Función main.
{
set_tris_a(0x00);              //Puerto A como salida.
disable_interrupts(GLOBAL);    //Todas las interrupciones desactivadas.
port_a=0x00;           //Puerto A = 0.
while(TRUE);           //Bucle infinito
 {
  output_low(PIN_A0);  //LED conectado en el Puerto RA0 off.
  output_low(PIN_A4);  //LED conectado en el Puerto RA4 on.
  delay_ms(500);       //Retardo 500 ms.
  output_high(PIN_A0); //LED conectado en el Puerto RA0 on.
  output_high(PIN_A4); //LED conectado en el Puerto RA4 off.
  delay_ms(500);       //Retardo 500 ms.
 }                     //Fin del bucle infinito.
}                      //Fin del main
```

Conexión de pulsadores

Todos los ejercicios realizados, hasta aquí, los puertos del PIC se han utilizado como salidas. A continuación se utilizaran como entradas. Se conectan pulsadores para cambiar los niveles de tensión en los pines del puerto.

La figura 2.7, muestra las conexiones para manejar pulsadores. Con esta conexión los pines del puerto (RD0 y RD1), están directamente en estado lógico de 1. Por consiguiente, el accionamiento del pulsador determina que pasen los puertos indicados a bajo.

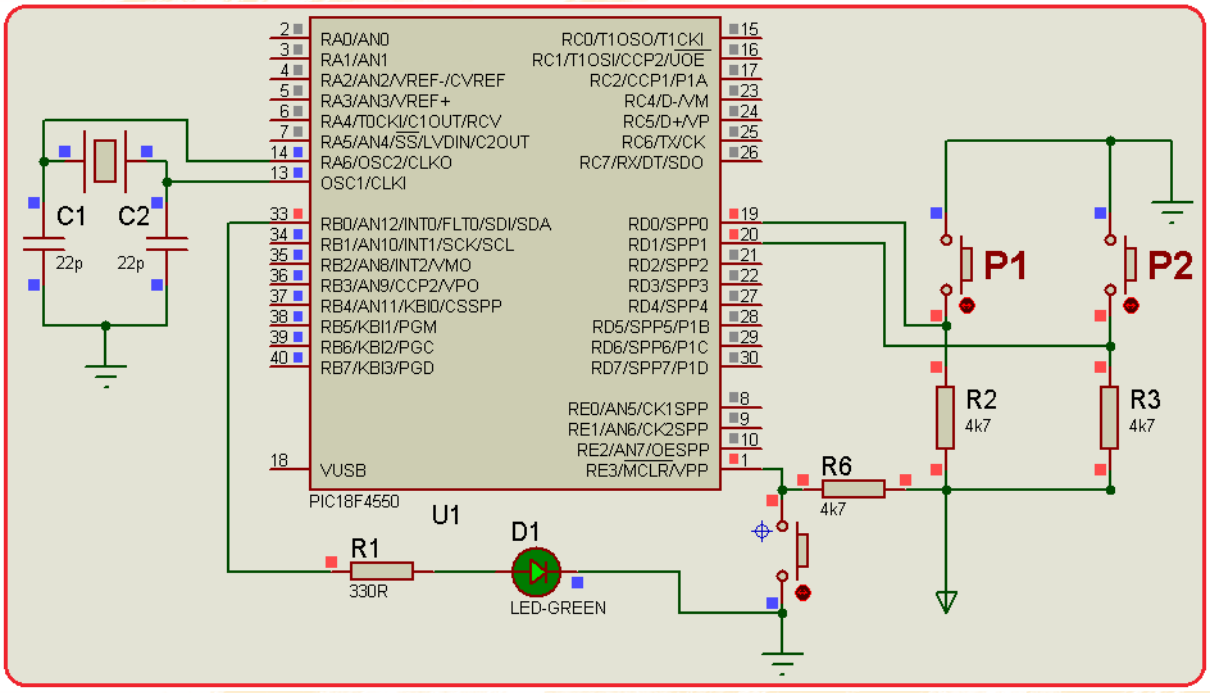


Figura 2.7. Conexión de pulsadores.
Elaborado por los autores.

Ejercicio 2.9. Manejo de pulsadores.

El ejercicio 2.9, consiste en prender un LED al accionar un pulsador 1 (P1) y apagar el LED al accionar el pulsador 2 (P2).

Los pulsadores P1 y P2 se han conectado al puerto D en los pines RD0 y RD1 respectivamente. En el programa se definen las variables prender y apagar mediante las directivas ***#define prender bit_test(port_d,0), #define apagar bit_test(port_d,1),*** las cuales permiten testear el estado de las entradas.

PROGRAMA:

```
#include <18f4550.h>    /                  /Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,PUT, NOPBADEN      //Configuración de fusibles.
#use delay (clock=12000000)   // F_OSC =12MHz.
#BYTE port_b= 0xF81  //Identificador para el puerto b en la localidad 0xF81
#BYTE port_d= 0xF83  //Identificador para el puerto d en la localidad 0xF83.
#define prender bit_test(port_d,0)       //Define el pin RD0 para prender.
#define apagar bit_test(port_d,1)        //Define el pin RD1 para apagar.

void main(void)                  //Función principal main.
{
set_tris_b(0x00);                //Define puerto b como salida.
set_tris_d(0xff);                //Define puerto d como entrada.
disable_interrupts(GLOBAL);      //Todas las interrupciones desactivadas.

 while(TRUE){                    //Inicio del bucle.
   if (prender == 0)             //Censa si P1= 0 .
   {
      bit_set(port_b,0);         //LED on.
   }
    if (apagar==0)               //Censa si P2  = 0.
   {
     bit_clear(port_b,0);        //LED off.
   }
 }          /                     /Fin de bucle infinito.
}                                //Fin del main.
```

Uso de la función input

Otra forma de censar el estado de los pines es utilizando la función ***input(PIN_D0) == 0.*** Para este caso si el pin del puerto D0 (o cualquier otro) que está en alto, si cambia de estado será detectado por la instrucción indicada.

Ejercicio 2.10. Manejo de pulsadores mediante la función input.

PROGRAMA:

```
#include <18f4550.h>              //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,PUT, NOPBADEN      //Configuración de fusibles.
#use delay (clock=12000000)  / /FOSC =12MHz
#use standard_io(B)               //Usa librería estándar para el puerto B.
#use standard_io(D)               //Usa librería estándar para el puerto D.

void main(void)                   //Función principal main.
{
disable_interrupts(GLOBAL);  //Todas las interrupciones desactivadas.
output_low(PIN_B0);               //LED off.
 while(TRUE){                     //Inicio del bucle.
   if (input(PIN_D0) == 0)        //Detecta si se accionado el pulsador P1.
     output_high(PIN_B0);         //LED on.
   if (input(PIN_D1) == 0)        //Detecta si se accionado el pulsador P2.
     output_low(PIN_B0);          //LED off.
   }        /                      Fin de bucle infinito.
}                                 //Fin del main.
```

Uno de los problemas que presentan los pulsadores mecánicos al momento de su accionamiento es el rebote de los contactos, lo que genera que se produzcan varios transitorios de voltaje (pulsos) que afectan el funcionamiento normal del circuito.

Una forma práctica de averiguar el anti rebote, es mediante el siguiente ejercicio, que consiste, en prender y apagar un LED con un mismo pulsador. En este caso al accionar el pulsador se prende el LED (si está apagado), y al volver a accionar el mismo pulsador se apaga el LED.

Ejercicio 2.11. Efecto del rebote mecánico de un pulsador.

PROGRAMA:

```
#include <18f4550.h>              //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)  / / FOSC =12MHz.
#use fast_io(b)                   //Directivas de i/o para puerto b.
#use fast_io(d)                   //Directivas de i/o para puerto d.
#BYTE port_b= 0xF81  //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83  //Identificador para el puerto d en  la localidad 0xF83.
#define pulsador bit_test(port_d,0);      //Define el pin RD0 para prender.
int k;                                    //Variable k para determinar el estado del LED.

void main()                       //Función  principal main.

{
set_tris_b(0x00);                 //Fija el puerto B como salida.
set_tris_d(0xff);                 //Fija el puerto D como entrada.
disable_interrupts(GLOBAL);  //Todas las interrupciones desactivadas.
port_b = 0;                       //Port =0.
 k= 0;                            //k=0.
  while(TRUE){                    //Inicio del bucle.
    if (pulsador== 0 && k==0) //Si pulsador = 0 y k = 0, realiza.
      {
        k= 1;                     //k=1.
        bit_set(port_b,0);        //LED on.
      }
     if (pulsador==0 && k==1) //Si pulsador = 0 y k = 1, realiza.....
      {
        k=0;                      //k=0.
        bit_clear(port_b,0);      //LED off.
      }
   }                              //Bucle infinito.
}                                 //Fin del main.
```

La variable *k*, permite en conjunto con el estado del pulsador prender el LED (k=0) y apagar el LED (k=1). Observe también que se efectúa una operación lógica *AND* con el estado de la entrada *(pulsador == 0 && k==0)*.

Al simular el programa o implementar en protoboard, el circuito de la figura 2.8, se observa como el LED titila varias veces hasta que se estabiliza:

quedando prendido o apagado. Claramente existe una situación de ambigüedad porque el LED no se sabe, qué estado tomará por la condición del pulsador.

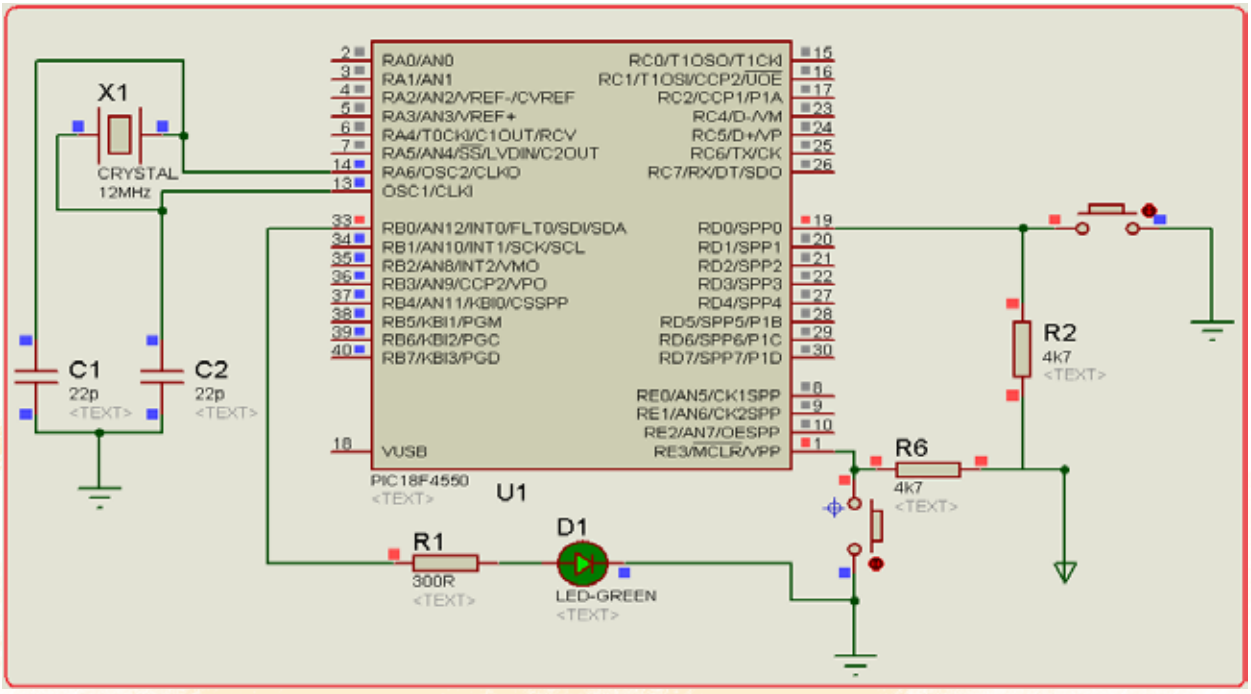


Figura 2.8. Circuito para demostrar el rebote mecánico del pulsador.
Elaborado por los autores.

Una forma de eliminar este inconveniente es mediante la inclusión de compuertas Smitch Trigger, o por software, como se presenta a continuación.

Ejercicio 2.12. Eliminación del rebote del pulsador.

PROGRAMA:

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)     // F_OSC =12MHz.
#use fast_io(b)                 //Directivas de i/o para puerto b.
#use fast_io(d)                 //Directivas de i/o para puerto d.
#BYTE port_b= 0xF81  //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83  //Identificador para el puerto b en  la localidad 0xF83.
#define pulsador bit_test(port_d,0);      //Define el pin RD0 para prender.
int k;                                    //Variable k para determinar el estado del LED.
void antirebote();                        //Función anti rebote.
```

```
void main()                        //Función  principal main.

{
set_tris_b(0x00);                  //Fija el puerto B como salida.
set_tris_d(0xff);                  //Fija el puerto D como entrada.
disable_interrupts(GLOBAL);        //Todas las interrupciones desactivadas.
port_b = 0;                        //Port =0.
 k= 0;                             //k=0.
  while(TRUE){                     //Inicio del bucle.
   if (pulsador== 0 && k==0)       //Si pulsador = 0 y k = 0, realiza.
     {
       antirebote();               //Si el pulsador = 0 llama a subrutina anti rebote.
       k= 1;                       //k=1.
       bit_set(port_b,0);          //LED on.
     }
    if (pulsador==0 && k==1)       //Si pulsador = 0 y k = 1, realiza.....
     {
      antirebote();                //Si el pulsador = 0 llama a subrutina anti rebote.
       k=0;                        //k=0.
       bit_clear(port_b,0);        //LED off.
     }
  }                                //Bucle infinito.
}                                  //Fin del main.

void antirebote(void)              //Función anti rebote.
  {
  while(pulsador == 0) { }         //No realiza nada hasta que el pulsador esté inactivo.
  delay_ms(30);                    //Retardo de 30 ms.
  return;                          //Retorna al programa principal.
  }                                //Bucle infinito.
```

Cada vez que se accione un pulsador se averigua si el estado del puerto sigue siendo 0, de ser afirmativo se llama a una función *"antirebote"*, para que de mantenerse en este estado no realice ninguna acción hasta que el pulsador este inactivo. La instrucción usada para el efecto es *while(pulsador == 0) { }*. Una vez que el pulsador este inactivo, de la función retornará *(por return)* al programa a siguiente instrucción desde donde fue llamado a seguir ejecutando el programa.

Al simular o implementar el circuito se observa que el LED inmediatamente se prende o se apaga, según el caso, eliminando el efecto del rebote del pulsador mecánico.

Generación de sonido

Otro de los elementos de Multimedia es el sonido, que vienen incorporados en la mayoría de equipos de comunicación y de procesamiento de información, razón por la cual, en este punto nos centraremos a la generación de sonido usando el microcontrolador, ya que los instrumentos y otros equipos electrónicos como pianos, guitarras eléctricas, ecualizadores, sintetizadores, alarmas, reloj digital, celulares, etc., basan el tratamiento y la generación de sonido en forma electrónica.

Las canciones, voz y sonido audible se componen de notas musicales: DO, RE, MI, FA, SOL, LA, SI. En términos de electrónica, cada nota musical equivale a una frecuencia. La tabla 2.8 muestra la nota musical y la correspondiente frecuencia.

NOTA	FRECUENCIA (Hz)
DO – C	262
RE – D	294
MI – E	330
FA – F	349
SOL- G	392
LA – A	440
SI – B	494

Tabla 2.8. Notas musicales y su correspondiente frecuencia.
Fuente: Electrónica, Microcontroladores y Psicología: Internet.

Las letras al lado de cada nota pertenecen a la nomenclatura inglesa. La librería TONES.C que maneja el sonido utiliza la nomenclatura inglesa.

Las notas tienen una duración, que está determinada por las figuras musicales, indicadas en la tabla 2.9.

FIGURA MUSICAL	TIEMPO DE DURACIÓN
Fusa	62 ms
Semicorchea	125 ms
Corchea	250 ms
Negra	500 ms
Blanca	1 segundo
Redonda	2 segundos

Tabla 2.9. Duración de una figura musical.
Fuente: Electrónica, Microcontroladores y Psicología: Internet.

El intervalo de frecuencias sonoras audibles se suele dividir en 10 intervalos de frecuencia (10 octavas), como muestra la tabla 2.10.

OCTAVA	FRECUENCIA (Hz)
Primera	15 – 30
Segunda	30 – 60
Tercera	60 – 125
Cuarta	125 – 250
Quinta	250 – 500
Sexta	500 – 1000
Séptima	1000 – 2000
Octava	2000 – 4000
Novena	4000 – 8000
Décima	8000 – 16000

Tabla 2.10. Octavas y sus equivalencias en frecuencia.
Fuente: Electrónica, Microcontroladores y Psicología: Internet.

El compilador CCS trae incorporado la librería llamada TONES.C para generar las notas musicales. Esta librería incluye la función:

generate_tone(frequency, duration); que permite crear las notas musicales. Los argumentos de la función son:

Frecuency, es una variable que toma el valor de la nota que se requiere generar, es decir la frecuencia de la nota.

Duration, es el tiempo que dura la nota y está dada en milisegundos. Cada nota trae en la librería predefinida las frecuencias, lo cual permite elegir en que octava tocarla; hasta una tercera octava.

Una parte del código de la librería TONES.C, se indica en la figura 2.9.

```
#ifndef  MUSIC_NOTES
#define  MUSIC_NOTES

//            NOTE                    FREQUENCY
//                        Octave0  Octave1  Octave2  Octave3
const long C_NOTE[4]  ={ 262,     523,     1047,    2093};
const long Db_NOTE[4] ={ 277,     554,     1109,    2217};
const long D_NOTE[4]  ={ 294,     587,     1175,    2349};
const long Eb_NOTE[4] ={ 311,     622,     1245,    2489};
const long E_NOTE[4]  ={ 330,     659,     1329,    2637};
const long F_NOTE[4]  ={ 349,     698,     1397,    2794};
const long Gb_NOTE[4] ={ 370,     740,     1480,    2960};
const long G_NOTE[4]  ={ 392,     784,     1568,    3136};
const long Ab_NOTE[4] ={ 415,     831,     1661,    3322};
const long A_NOTE[4]  ={ 440,     880,     1760,    3520};
const long Bb_NOTE[4] ={ 466,     923,     1865,    3729};
const long B_NOTE[4]  ={ 494,     988,     1976,    3951};
#endif

#define TONE_PIN  PIN_B0
```

Figura 2.9. Parte del código de la librería TONES.C, compilador CCS, versión 4.084.
Elaborado por los autores.

Tomando en cuenta esto (octavas y sus frecuencias), lo único que se necesita son las partituras de las canciones que deseamos generar o combinar las notas para crear diversos sonidos.

En la librería TONES.C, está definido el pin del puerto B.0 por donde genera las señales del sonido, como se ve en la figura 2.10. Basta modificar esta línea para utilizar cualquier otro pin del PIC.

```
#define TONE_PIN  PIN_B0


void do_delay(int ms_delay, int num_ms, int us_delay, int num_us)  {
 int i;
```

Figura 2.10. Definición del pin por donde se genera la señal del sonido. Compilador CCS versión 4.084
Elaborado por los autores.

Ejercicio 2.13. Generación de sonido.

En el siguiente ejercicio se van a generar las notas musicales DO, RE, MI, FA, SOL y con octavas alternadas de 0 a la 3 que proporciona la librería TONES.C

PROGRAMA:

```
#include <18f4550.h>   /          /Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)    //F_OSC =12MHz.
#include <TONES.c>             //Librería para gestión de sonido.
void main(void) {              //Función principal main.

int fusa=62;                   //Tiempo de duración fusa.
int semicorchea=125;           //Tiempo de duración semicorchea.
int corchea=250;               //Tiempo de duración corchea.

while(TRUE) {                  //Bucle infinito.
generate_tone(C_NOTE[0],fusa);
generate_tone(C_NOTE[1],fusa);
generate_tone(Eb_NOTE[2],corchea);
generate_tone(C_NOTE[3],fusa);
generate_tone(C_NOTE[0],fusa);
generate_tone(Eb_NOTE[1],corchea);
generate_tone(C_NOTE[3],fusa);
generate_tone(Eb_NOTE[3],fusa);
generate_tone(Ab_NOTE[3],semicorchea);
generate_tone(G_NOTE[3],semicorchea);
generate_tone(F_NOTE[3],semicorchea);
generate_tone(F_NOTE[3],semicorchea);
generate_tone(Eb_NOTE[3],semicorchea);
delay_ms(100);

 }                             //Fin del bucle infinito.
}                              //Fin del main.
```

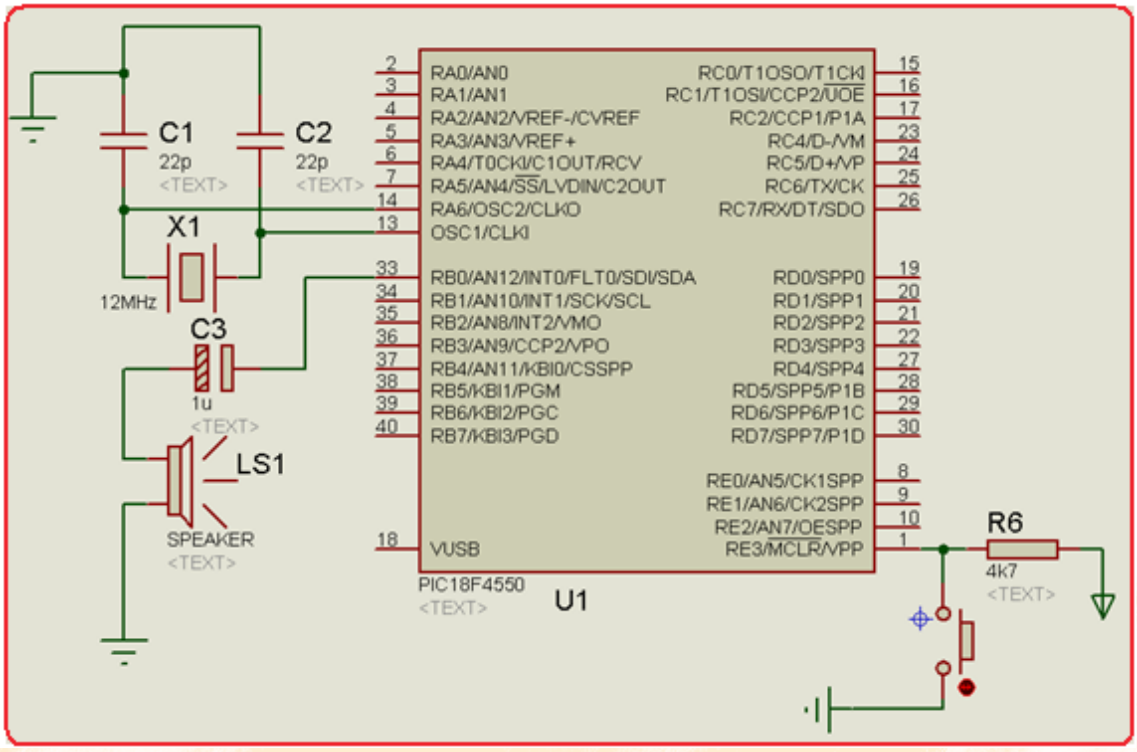


Figura 2.11. Circuito para generar sonido.
Elaborado por los autores.

En la figura 2.12, se muestra el circuito para la generación del sonido. Se debe acotar que el microcontrolador genera pulsos de voltaje cuadrados, es necesario filtrar esta señal. La inclusión de un capacitor como filtro a la salida ayuda a mejorar la señal audible. En todo caso, esto no garantiza la calidad del sonido. No es del alcance del texto realizar un análisis de filtros o del filtrado de la señal, por lo que el lector interesado puede consultar textos especializados para el manejo de filtros digitales o analógicos.

EJERCICIOS ADICIONALES

Ejercicios resueltos

Encender y apagar un LED con un interruptor.

En este ejercicio, cuando el interruptor está abierto el LED se apaga y cuando el interruptor está cerrado el LED se prende. Además se activan las resistencias PULL_UP para conectar internamente al puerto B a 5V.

PROGRAMA:

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,PUT, NOPBADEN       //Configuración de fusibles.
#use delay (clock=12000000)   //F_OSC =12MHz
#byte tris_b = 0xF93  //Identificador para el registro TRIS B en  la localidad 0xF93.
void main() {                           //Función principal main.
  set_tris_b(0xFF);             //Fija el Puerto B como salida.
  port_b_pullups(TRUE);         //Habilitación del  Pull-up.
  while(TRUE){                  //Bucle Infinito.
    if (input(PIN_B3) == 0)     //Detecta si se accionado el pulsador P1….
      output_high(PIN_D0);      //….y activa al LED.
   else                         //caso contrario…..
    output_low(PIN_D0);         //….el LED permanece apagado.
  }                             //Fin del bucle infinito.
}         /                      /Fin del main.
```

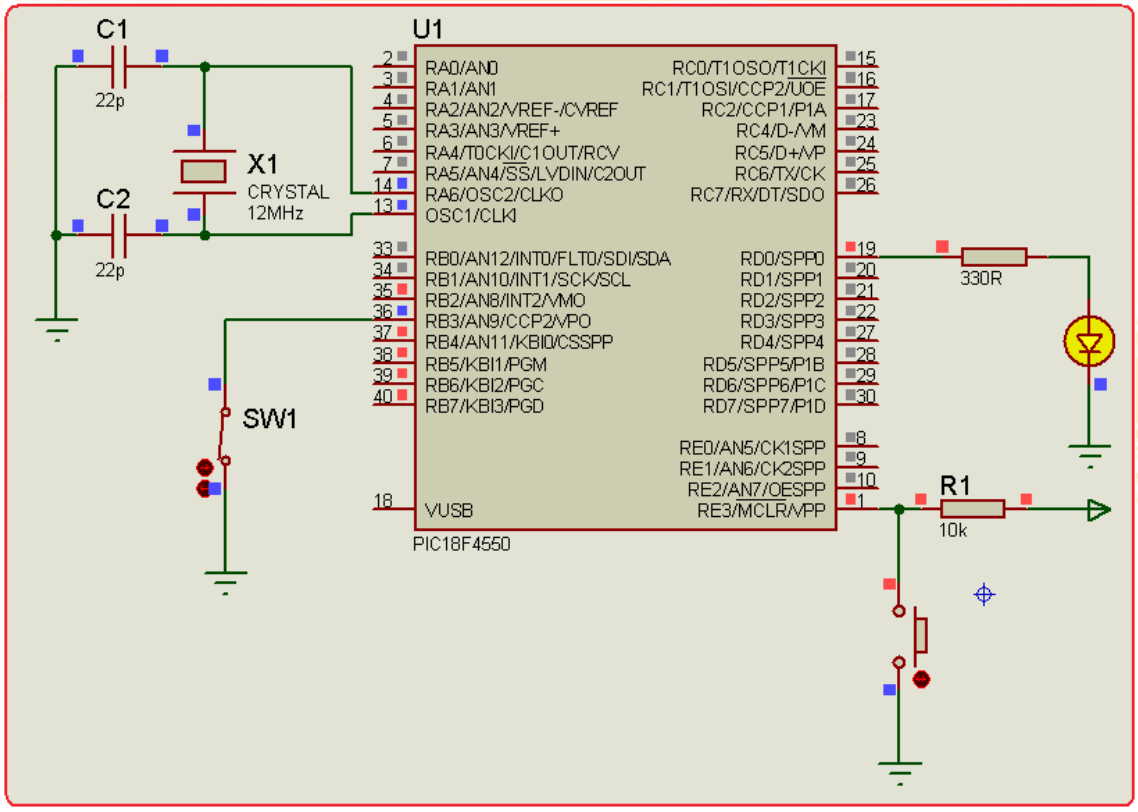


Figura 2.12. Conexión de un interruptor con resistencia PULL-UP interna.
Elaborado por los autores.

Secuencia de LEDS.

En este programa para que los LEDS conectados al puerto B se activen empezando desde RB0 hasta RB7. Luego empiezan a desactivarse uno por uno desde el último que se activó. La tabla 2.11, ilustra el funcionamiento.

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1

Tabla 2.11. Secuencia de encendido de LEDS.
Elaborado por los autores.

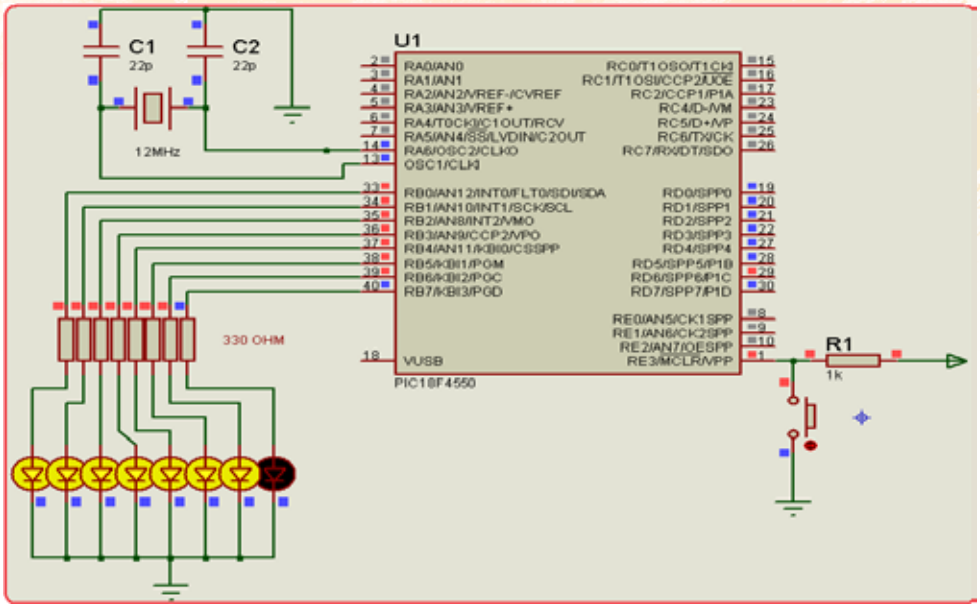


Figura 2.13. Encendido y apagado secuencial de LEDS.
Elaborado por los autores.

Uso de la resistencias internas Pull Up.

En el siguiente ejercicio un pulsador se conecta al puerto RB0 y un LED al puerto RA0. Al accionar el pulsador el LED se prende, caso contrario el LED permanece apagado. Se activa las resistencias internas Pull Up del puerto B mediante la instrucción **port_b_pullups (TRUE);** con el propósito de conectar el pulsador a V_{CC} y al accionar el mismo se conecte a GND.

PROGRAMA:

```
#include <16f88.h>                //Librería para usar el PIC16F88.
#fuses INTRC,NOWDT,NOMCLR,NOPROTECT,NOPUT //Configuración de fusibles.
#use delay (clock=4000000)        //Fosc=4MHz

void main(void){                  //Función principal main.
port_b_pullups (TRUE);            //Activa las resistencias Pull Up del puerto B.
disable_interrupts(GLOBAL);       //Todas las interrupciones desactivadas.
output_low(PIN_A0);               //Pin A0= 0.

 while(TRUE){                     //Bucle infinito.
   if (input(PIN_B0) == 0)        //Si pin RB0= 0....
     output_high (PIN_A0);        //Pin RA0= 1.
    else                          //Caso contrario...
     output_low(PIN_A0); /        /Pin RA0=0.

  }                               //Fin del bucle infinito
 }                                //Fin del main.
```

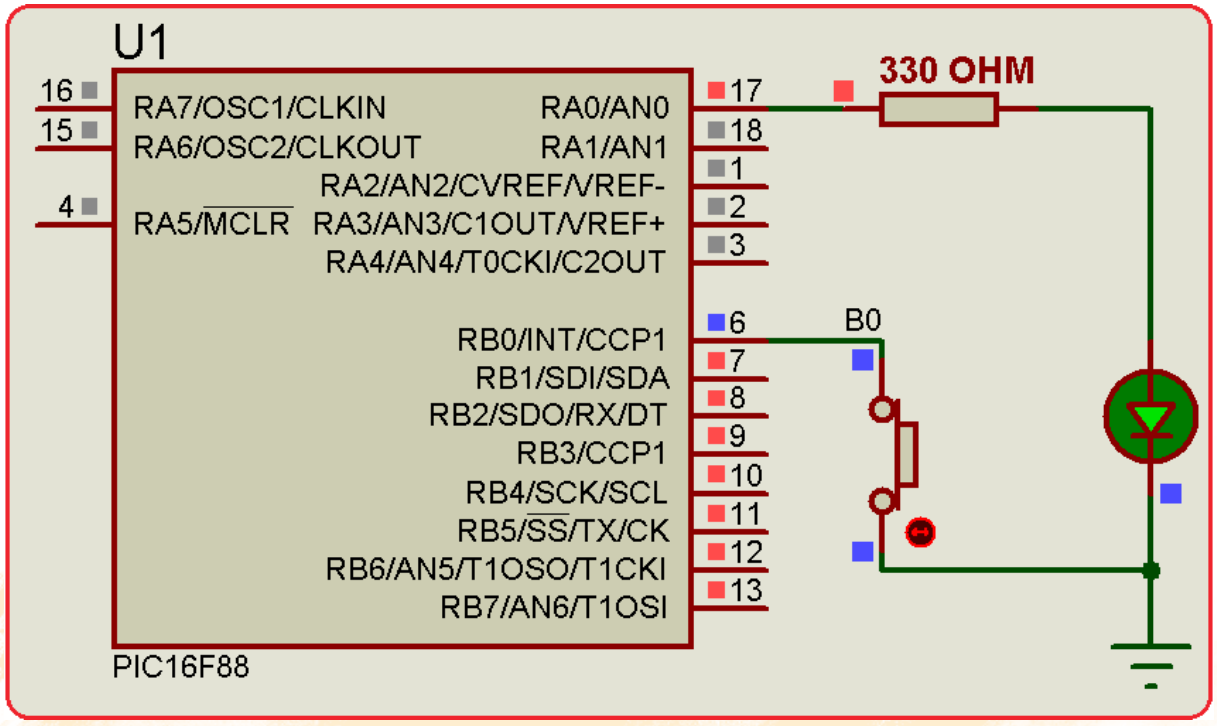


Figura 2.14. Demostración del funcionamiento de las resistencias PULL UP del puerto RB.
Elaborado por: Los Autores.

Ejercicios propuestos

Realice un programa para controlar dos semáforos.

Desarrolle el programa para que los LEDS conectados al puerto B se activen y desactiven según indica la secuencia en la tabla 2.12.

Nota: Optimizar la programación sin utilizar instrucciones repetitivas, en base a las instrucciones de desplazamiento elaborar una ecuación matemática para mover los bits.

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

1	1	1	1	1	1	1	0
1	1	1	1	1	1	0	0
1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Tabla 2.12. Secuencia de encendido para el ejercicio 2.5.
Elaborado por los autores.

Codifique la tabla de datos para manejar directamente a través del puerto B un display de 7 segmentos tipo cátodo común y realice el programa para un contador descendente.

Realice un programa para manejar los LEDS conectados a los puertos B y D. La secuencia de la tabla 2.13, se realiza en el puerto D y la secuencia de la tabla 2.14, en el puerto B. Optimizar la programación sin utilizar instrucciones repetitivas, utilizar instrucciones de desplazamiento.

RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

Tabla 2.13. Secuencia de encendidos de LEDs, para el ejercicio 2.7.
Elaborado por los autores.

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0
1	1	1	1	1	1	0	0
1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Tabla 2.14. Secuencia de encendidos de LEDs, para el ejercicio 2.7.
Elaborado por los autores.

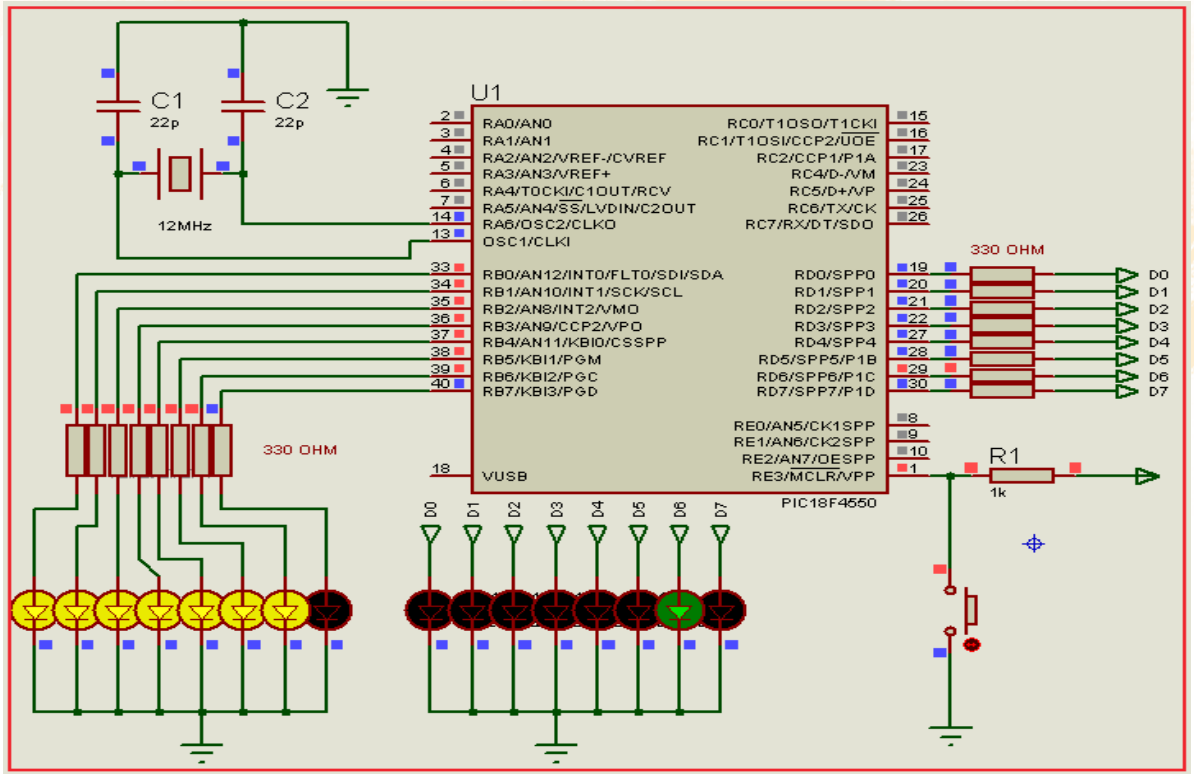


Figura 2.15. Circuito para el ejercicio 2.7.
Elaborado por los autores.

En el circuito de la figura 2.16, el LED D1 se activa al pulsar P1, D2 con P2 y D3 con P3. Cualquier LED puede activarse si los tres están apagados. El pulsador P4, sirve para apagar el LED que esté prendido.

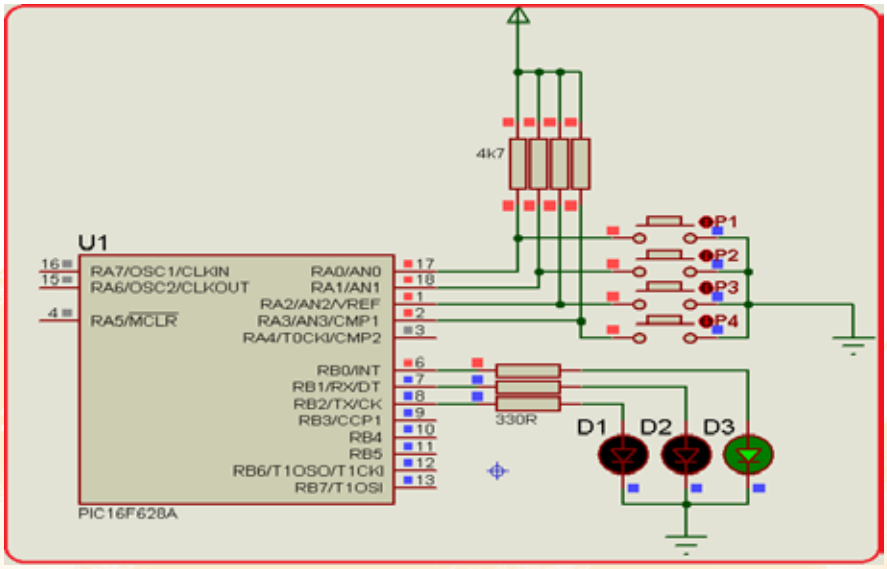


Figura 2.16. Ejercicio 2.8.
Elaborado por los autores.

Realizar el programa para simular el funcionamiento de dos conmutadores de tres vías. Al pulsar cualquiera de los pulsadores el estado del LED cambia.

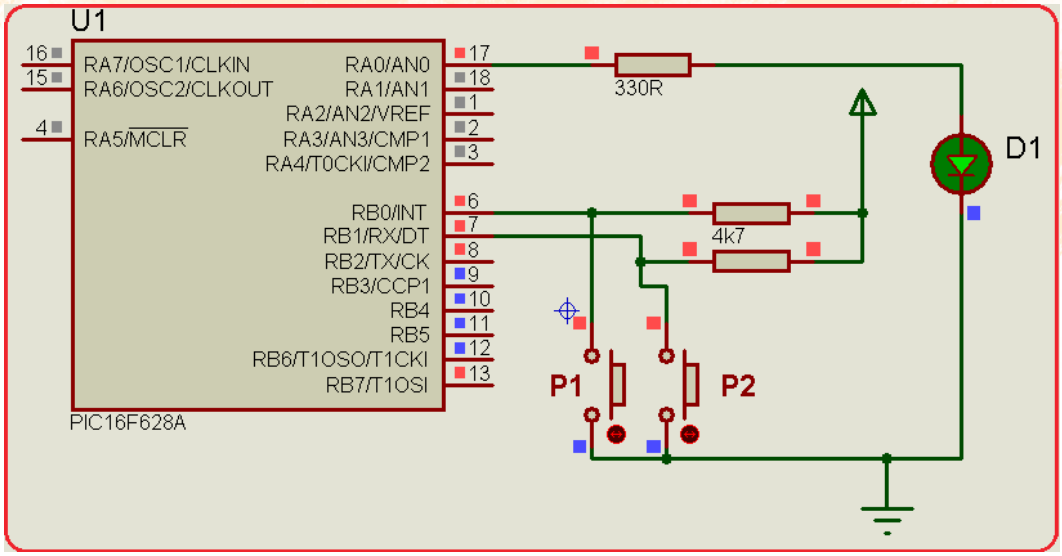


Figura 2.17. Ejercicio 2.9.
Elaborado por los autores.

Realizar el programa para prender 3 LEDS en secuencia al ir pulsando P1 y apagarlos con P2, si los tres LEDS están prendidos.

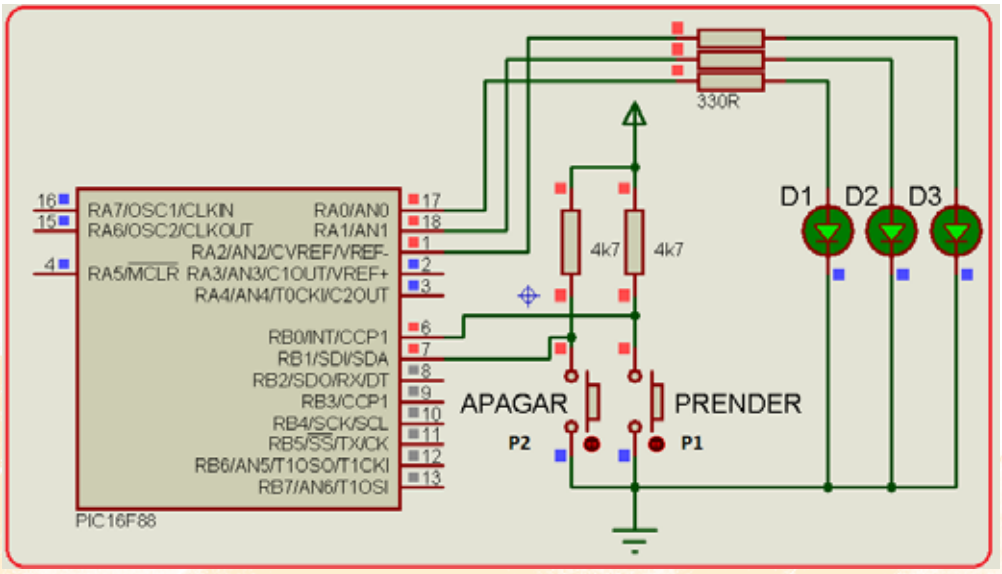
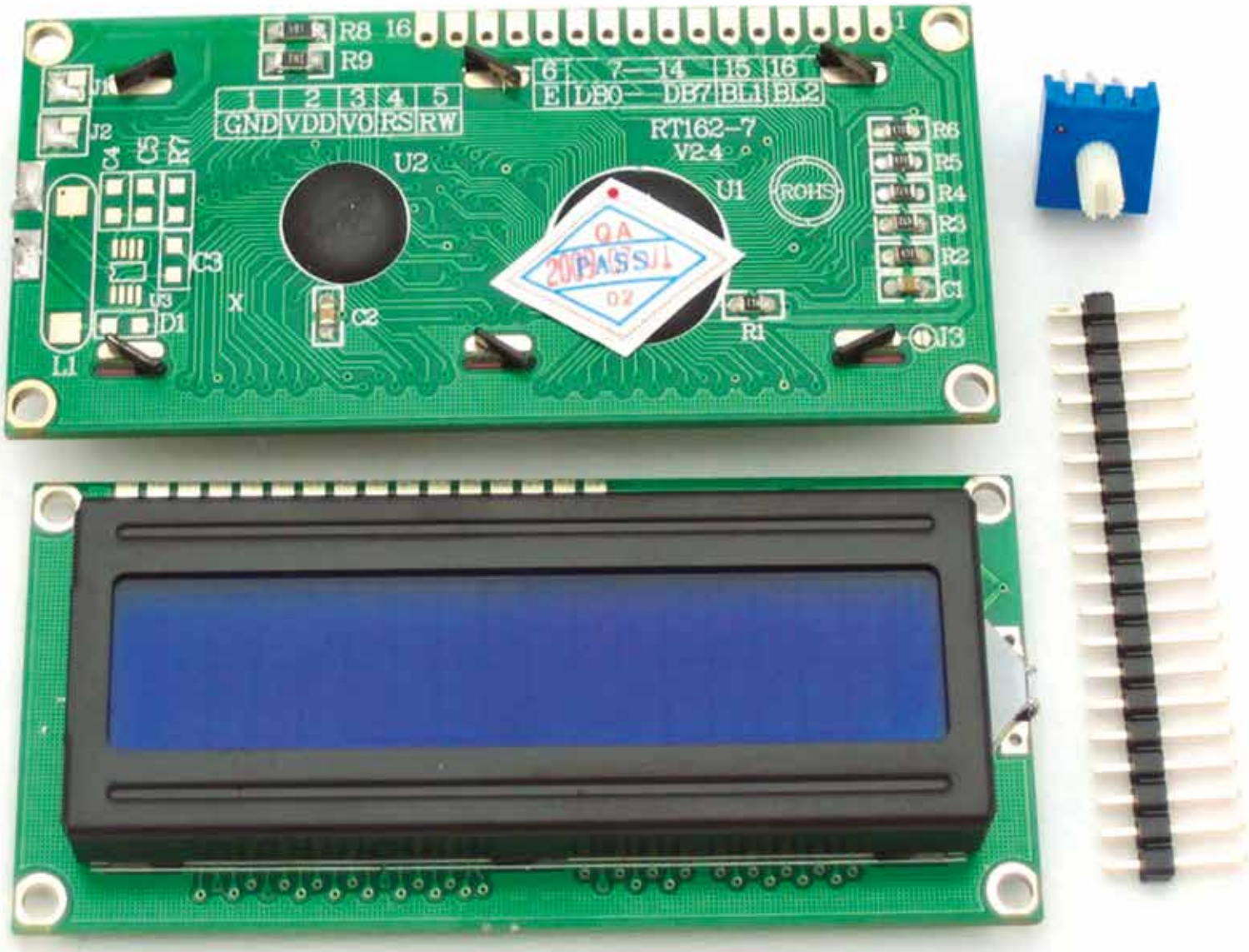


Figura 2.18. Encendido secuencial de LEDS.
Elaborado por los autores.



CAPÍTULO 3

MANEJO DE DISPLAYS, MÓDULOS LCD, GLCD Y TECLADOS

Para la visualización de datos es muy común el uso de displays de 7 segmentos, módulos LCD y GLCD, por lo que en este capítulo revisaremos las diferentes formas de manejo y control de estos dispositivos, con el propósito de que los estudiantes tengan las herramientas suficientes para programar y controlar los puertos del Microcontrolador. El manejo de los teclados hexadecimales es otra que las aplicaciones que abordamos en este capítulo.

Manejo de Display de 7 segmentos

Para visualizar los datos en un display de 7 segmentos, existen dos posibilidades.

- Conectar un decodificador de BCD a 7 segmentos,
- Conectar el display directamente al puerto.

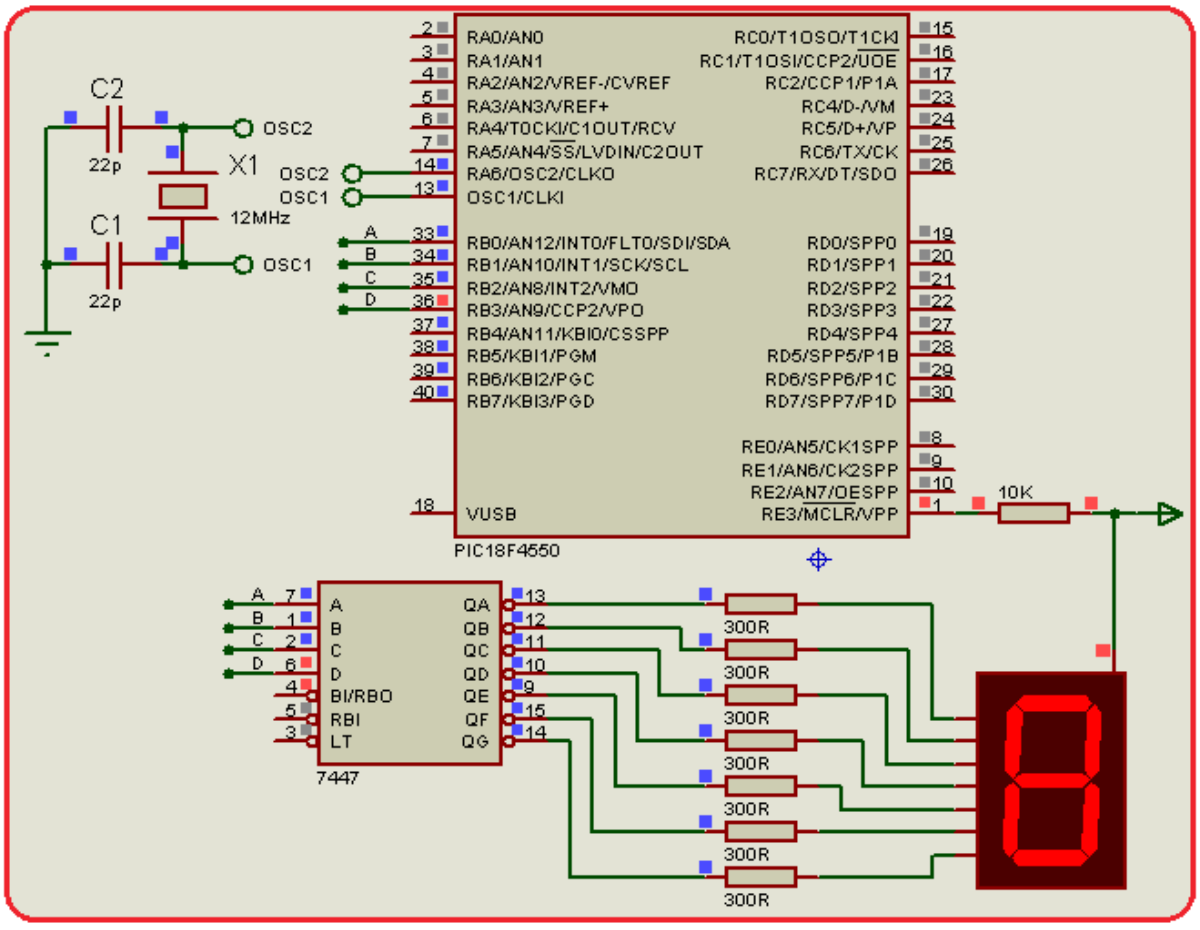


Figura 3.1. Conexión de display de 7 segmentos tipo ánodo común
Elaborado por los autores.

Conexión de un decodificador de BCD a 7 segmentos

Un decodificador de BCD a 7 segmentos tiene cuatro líneas de entrada y siete salidas, el código binario generado por el microcontrolador es transformado a 7 segmentos para ser visualizado en el display. Si el display es de ánodo común se utiliza el decodificador TTL 7447, para un display cátodo común el decodificador TTL 7448. La figura 3.1, muestra las conexiones de un decodificador 7447 y un display ánodo común. Se debe tener en cuenta de limitar la corriente de los segmentos mediante resistencias. El uso de resistencias de valores entre 220Ω y 330Ω garantiza el funcionamiento normal del display.

Ejercicio 3.1. Contador de 0 a 9 utilizando un decodificador y display.

En este caso los datos de un contador realizado en un bucle FOR, se pueden sacar directamente al puerto y el decodificador se encarga de convertirlos en 7 segmentos. La variable i se incrementa en 1 cada 500 ms y este valor es asignado al puerto b (*port_b= i;*)

PROGRAMA:

```
#include <18f4550.h>              //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,PUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)       // F_OSC = 12MHz.
#BYTE port_b= 0xF81               //Identificador para el puerto b en  la localidad 0xF81.
int i;                            //Variable i para el bucle FOR.

void main(void)                   //Función principal main.
{
set_tris_b(0x00);                 //Define port_b como salida
disable_interrupts(GLOBAL);       //Todas las interrupciones desactivadas.
 port_b = 0;                      //Puerto b = 0.

 while(TRUE)                      //Bucle infinito.
  {                               //Inicio del bucle.

    for (i=0; i<=9;++i){          //Repite mientras i sea menor o igual a 9.
        delay_ms(500);            //Retardo de 500 ms.
        port_b= i;                //El valor de i se saca al Puerto B. i varia de 0 a 9.
        }
  }                               //Fin del bucle de inicio.
}                                 //Fin del bucle main.
```

Manejo directo de display de 7 segmentos

En ciertas aplicaciones donde se disponen de suficientes líneas del puerto del microcontrolador y reducir el espacio físico es la prioridad, el display se puede conectar directamente a cualquier puerto.

En la figura 3.2, se muestra el circuito del display conectado al puerto B del microcontrolador. Se utiliza un display de ánodo común por lo que el terminal común del display se conecta a V_{CC}. También se podría utilizar un display cátodo común.

Así mismo, como cuando se utiliza un driver decodificador de 7 segmentos, se debe tener cuidado de utilizar las resistencias limitadores de corriente para conectar los segmentos del display al puerto y evitar que se produzcan posibles sobrecorrientes que puedan afectar al puerto o al segmento del display. Resistencias con valores entre 220Ω y 330Ω, son comunes utilizar para activar los segmentos del display.

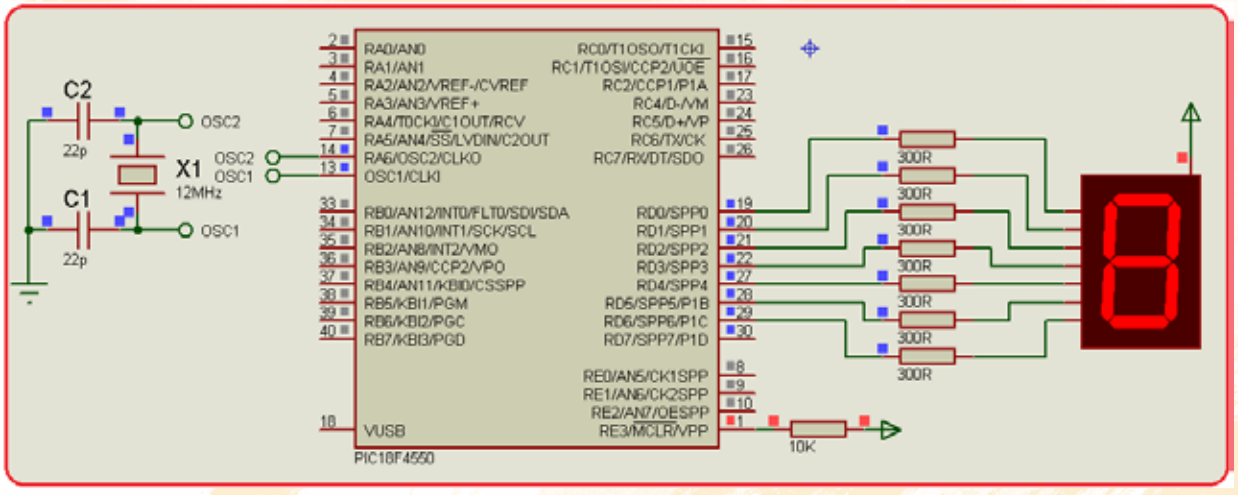


Figura 3.2. Conexión directa del display de 7 segmentos al microcontrolador.
Elaborado por los autores.

Ahora considerando que los datos que salen al puerto deben estar decodificados en 7 segmentos, se debe elaborar la tabla de datos decodificados para manejar por software y obtener los números respectivos.

La tabla 3.1, muestra los valores para activar los segmentos de un display ánodo común, recuerde que el display de ánodo común con un nivel alto (1) el segmento se desactiva y con un nivel bajo (0) el segmento se activa.

g	f	e	d	C	B	a	Equivalencia decimal	Número en el display
1	0	0	0	0	0	0	64	0
1	1	1	1	0	0	1	121	1
0	1	0	0	1	0	0	36	2
0	1	1	0	0	0	0	48	3
0	0	1	1	0	0	1	25	4
0	0	1	0	0	1	0	18	5
0	0	0	0	0	1	1	3	6
1	1	1	1	0	0	0	120	7
0	0	0	0	0	0	0	0	8
0	0	1	0	0	0	0	16	9

Tabla 3.1. Datos decodificados para activar al display de 7 segmentos.
Elaborado por los autores.

Ejercicio 3.2. Contador MOD 10, manejando directamente el display de 7 segmentos con el microcontrolador.

Los datos mostrados en la tabla 3.1 deben ser enviados directamente al puerto para activar los segmentos según el número indicado. Para esto en el programa en una constante tipo array se almacenan los valores codificados:

const int x[]= {64,121,36,48,25,18,2,120,0,16};

Mediante el lazo *for* se van sacando los valores desde el 0 hasta el 9. La variable *i* del bucle *for* va asignando al puerto el valor que ocupa en el vector *const*. Así, por ejemplo si *i* vale 3, el puerto tendrá el valor de 48 que corresponde al número 3. Observe la instrucción *port_d= x[i].*

PROGRAMA:

```
#include <18f4550.h>  //Librería para usar el PIC18F4550
#fuses HS,WDT,NOPROTECT,PUT, NOPBADEN        //Configuración de fusibles
#use delay (clock=12000000)   //F_OSC =12MHz
#BYTE port_d= 0xF83           //Identificador para el puerto d en la localidad 0xF83.
int i;                        //Variable i, contador para el bucle FOR.
const int x[]= {64,121,36,48,25,18,2,120,0,16};//Tabla de datos decodificados a 7 segmentos.
void main(void)               //Función principal main.
{
set_tris_d(0x00);             //Identificador para el puerto d como salida.
disable_interrupts(GLOBAL);   //Todas las interrupciones desactivadas.
 port_d = 64;                 //Asigna el valor de 64 al Puerto para ver el número 0.

 while(TRUE)          //Bucle infinito.
  {                   //Inicio del bucle.
   for (i=0; i<=9;++i) //Repite mientras i sea menor o igual a 9.
     {
       delay_ms(500);  //Retardo de 500 ms. Necesario para mantener el dato en el
                       // puerto.
       port_d= x[i];   //Asigna al puerto el valor apuntado por la variable i en el
                       //array x.
     }
  }                    //Fin del bucle infinito.
}                      //Fin del main.
```

MULTIPLEXACIÓN DE DISPLAYS DE 7 SEGMENTOS

En ciertos proyectos es necesario el ahorro de las líneas de los puertos para utilizarlos en otras aplicaciones, en estos casos, se recurre a manejar los displays por el mismo puerto. Esto se consigue usando el principio de la multiplexación.

Para el ejemplo se va a diseñar un contador módulo 100 (00-99). Los datos de las unidades se visualizan en el display 1 y las decenas en el display 2. Para controlar los dos displays simultáneamente, el proceso consiste en prender el display 1 y apagar el display 2 durante un corto tiempo y mostrar los datos de las unidades, luego prender el display 2 y apagar el display 1 para indicar los datos de las decenas. El encendido y apagado de los displays se realiza controlando la corriente de la base de dos transistores que trabajan en corte y saturación los mismos que manejan a los displays como se indica en la figura 3.3.

Los datos en binario para los displays son enviados por el puerto RB0:RB3, el decodificador 7447 convierte de BCD a 7 segmentos. Las dos líneas de control para los transistores PNP, son del puerto RD0 y RD1 (líneas S1 y S2). Los transistores permanecen en corte y saturación durante 20 ms, con lo cual para ojo humano "casi" es imperceptible el apagado del display y apenas existirá un leve parpadeo. Esta acción se repite por 50 veces en la función ***display()*** obteniendo un tiempo de retardo de 1s.

En la figura 3.3, RD0 controla al display de las decenas y RD1 al display de las unidades, por tanto, cuando se ejecuta la instrucción *port_d= 1;* RD0=1 y RD1= 0, con lo cual el transistor Q1 se apaga y el transistor Q2 se prende, los datos del puerto corresponden a las unidades *port_b= unidad*. En el caso de *port_d= 2*; RD0=0 y RD1= 1, en este caso el transistor Q1 se prende y el transistor Q2 se apaga, los datos del puerto corresponden a las decenas *port_b= decenas*. Recuerde que en los transistores PNP con un 0 lógico en la base conduce y con un 1 lógico en la base el transistor está abierto.

La figura 3.3, muestra las conexiones para manejar por el mismo puerto y un solo decodificador dos display de 7 segmentos simultáneamente.

Ejercicio 3.3. Manejo de dos display por principio de multiplexación

PROGRAMA:

```
#include <18f4550.h>              //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,PUT, NOPBADEN      //Configuración de fusibles.
#use delay (clock=12000000)   //F_OSC =12MHz.
#BYTE port_b= 0xF81  //Identificador para el puerto b en la localidad 0xF81.
#BYTE port_d= 0xF83  //Identificador para el puerto d en la localidad 0xF83.

int unidad, decena, x;            //Variables.

//Función display. Muestra los datos en los display y genera un retardo de 1segundo.
void display()
{
  for (x= 1; x<= 25; ++x){        //Repite 25 veces.
     port_d = 1;                  //Habilita display unidad y apaga display decena.
     port_b= unidad;              //Asigna al Puerto b el dato de unidad.
     delay_ms(20);                //El dato de unidad permanece el Puerto b 20 ms.
     port_d = 2;                  //Habilita el display decena y apaga display unidad.
     port_b= decena;              //Asigna al Puerto b el dato de decena.
     delay_ms(20);                //El dato de decena permanece el Puerto b 20 ms.
```

```
  }
  return;                    //Retorna al programa principal.
}

void main(void)              //Función principal main.
{
set_tris_b(0x00);            //Define port_b como salida.
set_tris_d(0x00);            //Identificador para el puerto d como salida.
disable_interrupts(GLOBAL);  //Todas las interrupciones desactivadas.
 port_b = 0;                 //Puerto b =0.
 while(TRUE){
    for (decena=0; decena<=9;++decena){     //Datos para el display decena 0 a 9.
       for (unidad=0; unidad<=9;++unidad){  //Datos para el display unidad 0 a 9.
          display();                        //Función display.
      }
    }
  }                          //Fin del bucle infinito.
}                            //Fin del bucle main.
```

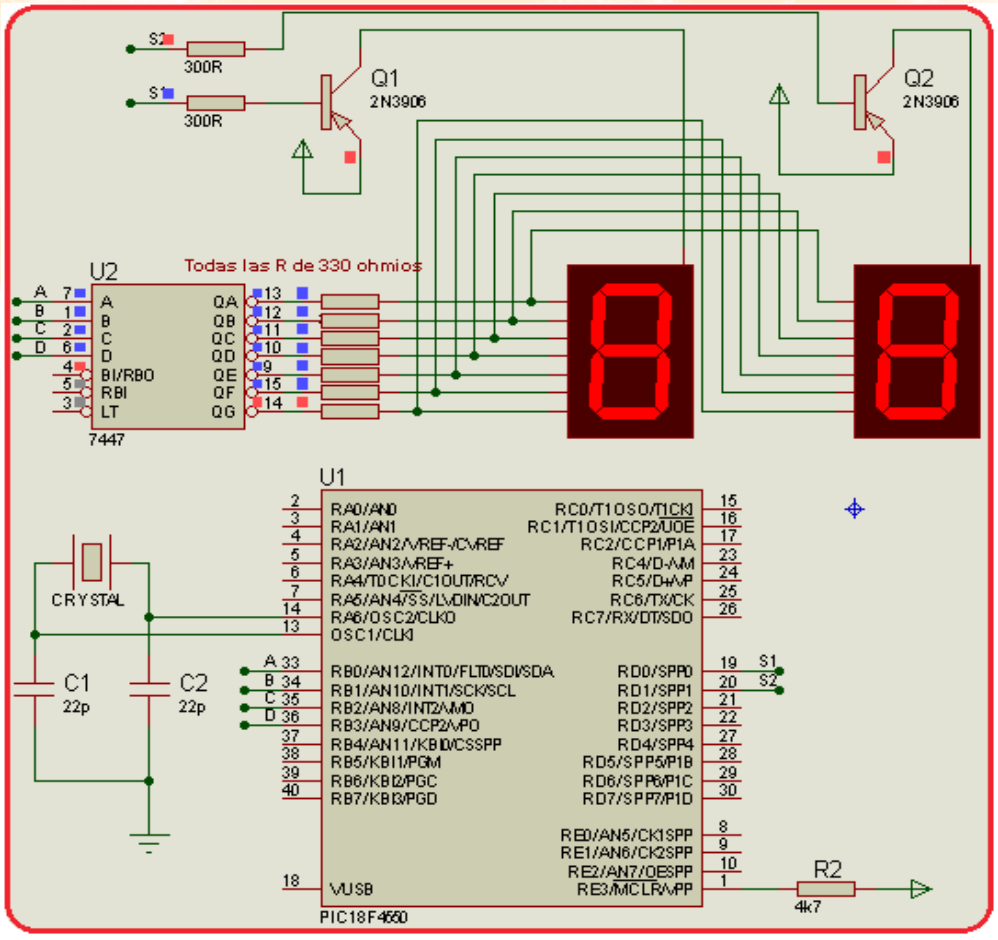
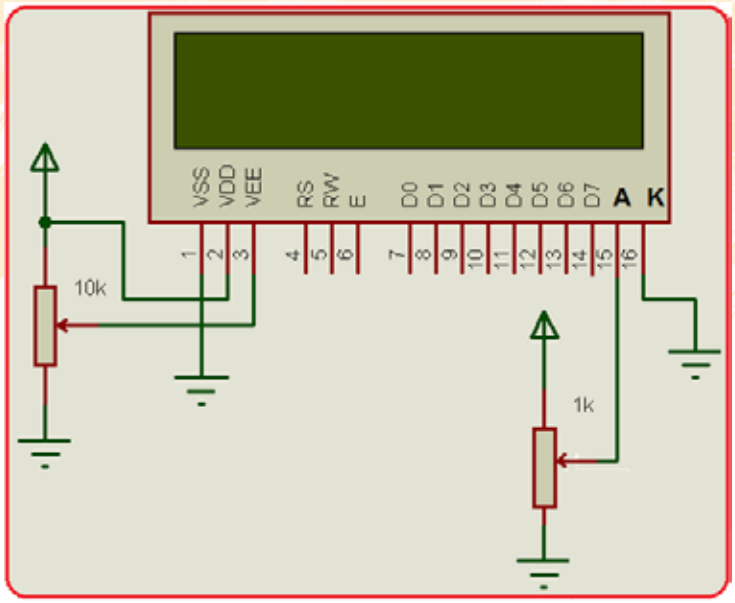


Figura 3.3. Conexión de dos display con el mismo decodificador**.**
Elaborado por los autores.

Módulo LCD

Los módulos LCD son muy útiles para ver cualquier tipo de información. La pantalla de cristal líquido está conformada por una, dos o cuatro líneas de 8, 16, 20, 24 o 40 caracteres de 5x7 pixels c/u. Los más comunes son los módulos LCD de 16x2, 16x4, con y sin backlight). La figura 3.4 a, muestra una vista de módulo LCD 16 x2 de 16 pines (los pines 15 y 16 corresponden al backlight, control de luz de fondo). Las conexiones para controlar el contraste y el backlight del LCD se indican en la figura 3.4 b. Dependiendo el puerto que se utilice los demás pines se conectan al microcontrolador como veremos más adelante.

Imagen de un módulo LCD típico.



Conexiones de las señales de contraste y backlight para LCD que disponen de esos terminales.

Figura 3.4. Módulo LCD.
Elaborado por los autores.

La descripción de pines del módulo LDC, se indica en la tabla 3.2.

Pin número	Símbolo	Función
1	Vss	Tierra o Masa
2	Vdd	Alimentación + 5 VDC
3	Vo	Voltaje de ajuste de contraste (5V no visible, 0v más visible)
4	R/S	Selección de Dato / Comando
5	R/W	Lectura / Escritura
6	E	Habilitador
7	D0	1a línea de datos (LSB)
8	D1	2a línea de datos
9	D2	3a línea de datos
10	D3	4a línea de datos
11	D4	5a línea de datos
12	D5	6a línea de datos
13	D6	7a línea de datos
14	D7	8a línea de datos (MSB)
15	A	Alimentación Backlight +3.5 V a +5V
16	K	GND del Backlight

Tabla 3.2. Funciones de los pines del módulo LCD.
Elaborado por los autores

El compilador CCS incluye un archivo (driver) *lcd.c*. El driver *lcd.c* está configurado para trabajar con el puerto d. Modificando este fichero es posible usarlo para cualquier puerto. El fichero se llama con #include <lcd.c> y algunas funciones que presenta son:

Las instrucciones básicas para manejar el LCD son las siguientes:

lcd_init();

Es la primera función que debe ser llamada. Borra el LCD y lo configura en formato de 4 bits con dos líneas y con caracteres de 5x8 puntos, en modo encendido, cursor apagado y sin parpadeo. Configura el LCD con un autoincremento del puntero de direcciones y sin desplazamiento del display real.

lcd_gotoxy(byte x, byte y);

Indica la posición de acceso al LCD. Por ejemplo (1,2) indica la primera posición de la segunda línea; (2,1) segunda posición de la primera línea (El LCD tratado tiene 16 posiciones en cada línea).

lcd_getc (byte x, byte y);

Lee el carácter de la posición (x,y).

lcd_putc (char s);

Escribe la variable en la posición correspondiente, s es una variable tipo char. Además se puede incluir:

f se limpia el LCD.

n el cursor cambia de línea.

\b el cursor retrocede una posición.

Existe la función **printf** que es más versátil para trabajar en el LCD.

printf (string)

printf (cstring, valores,,)

printf(fname, cstring, valores,,)

string es una cadena o array de caracteres, *valores* es una lista de variables separadas por comas, y *fname* es una función.

El formato es ***%nt,***

Dónde:

n es opcional y puede ser:

1-9 especifica el número de caracteres.

01-09 indica la cantidad de ceros a la izquierda.

1.1-9.9 para punto flotante.

t puede indicar un: carácter **c**, cadena o caracteres, entero sin signo **u**, entero con signo **d**, entero largo sin signo **Lu**, Entero largo con signo **Ld**, entero hexadecimal (minúsculas) **x**, entero hexadecimal (mayúsculas) **X**, entero largo hexadecimal (minúsculas) **Lx**, entero largo hexadecimal (mayúsculas) **LX**, flotante con truncado **f**, flotante con redondeo **g**, flotante el forma exponencial **e**, entero sin signo con decimales insertados **w**.

Manejo del módulo LCD usando Puerto D

El siguiente programa envía el conocido mensaje "HOLA MUNDO" al LCD. Abran el archivo C:\Archivos de programa\PICC\Drivers\lcd.c, y ahí les muestra la forma de conectar el LDC a los pines del puerto D, como se indica en la figura 3.5. Se debe aclarar que el archivo utilizado corresponde al CCS versión v4.084.

```
// As defined in the following structure the pin connection is as follows
//      D0  enable
//      D1  rs
//      D2  rw
//      D4  D4
//      D5  D5
//      D6  D6
//      D7  D7
//
//   LCD pins D0-D3 are not used and PIC D3 is not used.

// Un-comment the following define to use port B
```

Figura 3.5. Conexión de pines del PIC con el LCD.
Elaborado por los autores.

Ejercicio 3.4. Manejo del módulo LCD por el Puerto D.
Algunos aspectos importantes para manejar el LCD son:

- Incluir la librería que maneja el LCD: #*include* <*lcd.c*>
- Inicializar el LCD: *lcd_init();* esta instrucción "arranca" el LCD.
- Situar el cursor en la posición columna y fila desde donde se visualice los caracteres: *lcd_gotoxy(4,1);* en este caso está indicando que empiece en la columna 4 fila 1.
- Escribir el mensaje: *lcd_putc("HOLA MUNDO!!\n LCD en CCS")*. En la instrucción el argumento *\n*. Por tanto, en la primera fila a partir de la columna 4 en el LCD se indicará la palabra *HOLA MUNDO!!*, y en la segunda fila a partir de la columna 4 se mostrará *LCD en CCS*. Cada espacio en blanco que separa los caracteres o leras (***\n LCD***) ocupan una posición en las columnas del LCD.

PROGRAMA:

```
#include <18f4550.h>          //Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOWRT, NOPUT, NOWDT //Configuración de fusibles.
#use delay (clock=12000000)   //F_OSC =12MHz.
#include <lcd.c>              //Librería lcd.c
set_tris_d(0x00);             //Identificador para el puerto d como salida

void main(void)               //Función principal main.
{
  lcd_init();                 // Inicializa el LCD.
  lcd_gotoxy(4,1);            //Sitúa el cursor en columna 4, fila 1.
  lcd_putc("HOLA MUNDO!!\n   LCD en CCS");    //Escribe en el LCD.
}
```

Para cambiar el cursor de una línea a otra o posicionar en una columna distinta la instrucción *lcd_gotoxy(x,y);* es de mucha utilidad. En el presente ejercicio se tiene el mismo resultado si se escribe el código así:

```
lcd_gotoxy(4,1);              //Sitúa el cursor en columna 4, fila 1.
lcd_putc("HOLA MUNDO!! ");    //Escribe en el LCD.
lcd_gotoxy(4,2);              //Sitúa el cursor en columna 4, fila 2.
lcd_putc(LCD en CCS");        //Escribe en el LCD.
```

Todo dependerá de las necesidades de diseño y habilidades del programador. El diagrama de conexiones para realizar las pruebas de funcionamiento de los programas del módulo LCD conectado al puerto D, se muestra en la figura 3.6. No están consideradas las conexiones del control de contraste y el Bach light. Al implementar el circuito físico las conexiones mencionadas se indican en la figura 3.4 b.

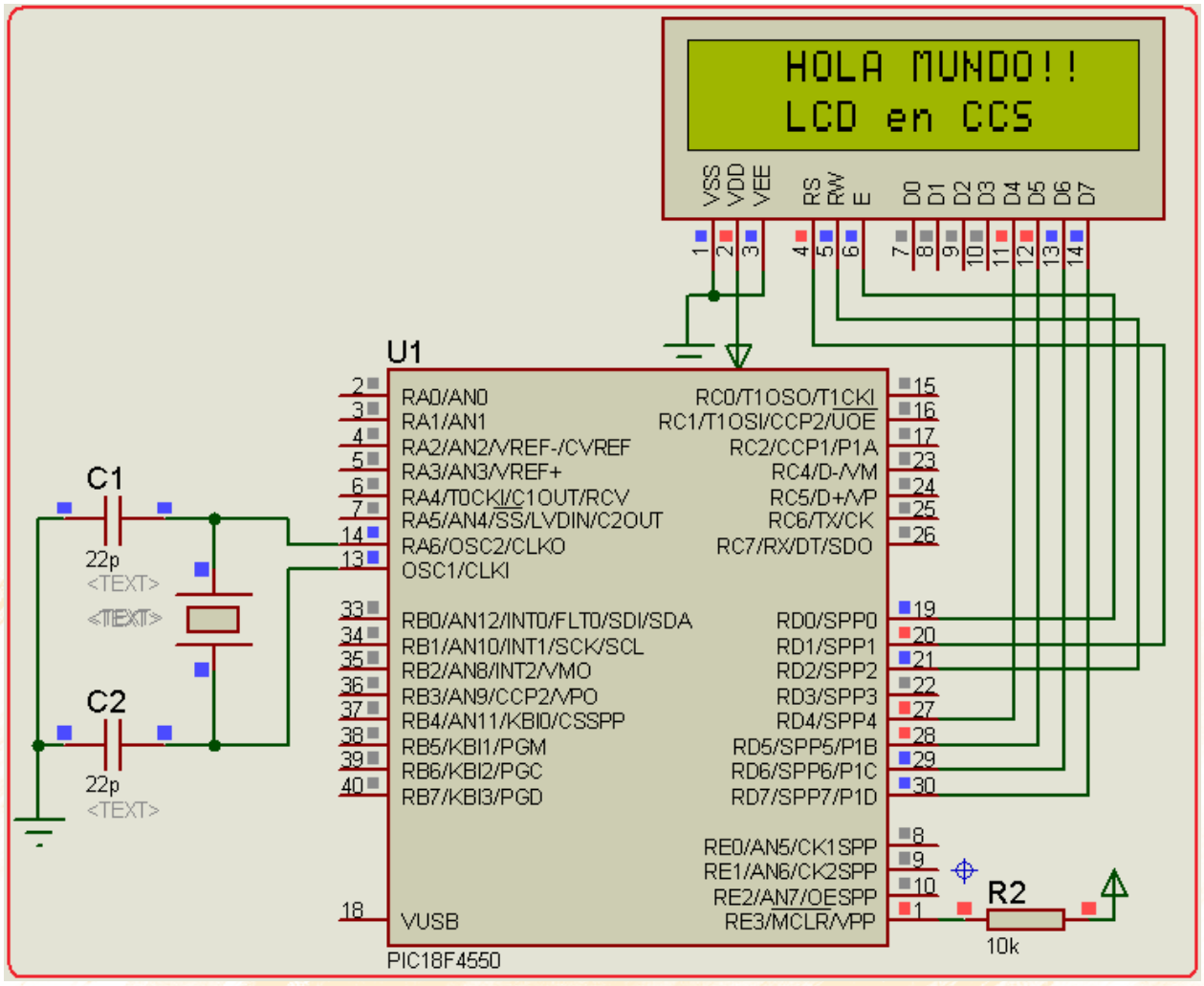


Figura 3.6. Conexión del LCD en el puerto D.
Elaborado por los autores.

Para indicar el valor de una variable se debe utilizar la función *printf(lcd_putc," %d constante ",variable);*

Ejercicio 3.5. Contador con visualización en LCD.

El ejercicio 3.5, muestra un contador ascendente de 0 a 9 con visualización de datos en el LCD. Para mostrar el valor numérico se utiliza la instrucción *printf(lcd_putc," %d ",i)*. El argumento *%d* de la función printf indica que el valor de la variable *i* es entero decimal.

PROGRAMA:

```
#include <18f4550.h>          //Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOWRT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)   //FOSC =12MHz.
#include <lcd.c>              //Librería lcd.c.
int i;                        //Variable i.

void main(void)               //Función principal main
{
lcd_init();                   // Inicializa el LCD.
while (TRUE){                 //Inicio del bucle.
  lcd_gotoxy(5,1);            //Sitúa el cursor en 5 columna, 1 fila.
  lcd_putc("CONTADOR");       //Escribe CONTADOR.
 for (i=0; i<=9;++i)          //La variable empieza desde 0 hasta 9.
    {
    lcd_gotoxy(8,2);          //Sitúa el cursor en 8 columna, 2  fila.
      printf(lcd_putc," %d ",i);  //Escribe en el LCD el valor actual de i.
      delay_ms(500);          //Retardo de 500 ms.
    }
 }                            //Fin del bucle infinito.
}                             //Fin del bucle main.
```

La figura 3.7, indica el resultado obtenido. Se ha capturado la pantalla cuando el valor de la variable i = 2.

Figura 3.7. Vista parcial del resultado en el LCD.
Elaborado por los autores.

Manejo del LCD por el puerto B

Para usar el LCD en el puerto B o cualquier otro se debe modificar el fichero del lcd.c o definir los pines de conexión según las necesidades.

Para el puerto B, utilizaremos las conexiones como se indica en la tabla 3.3.

PUERTO	LCD
BO	ENABLE
B1	RS
B2	RW
B3	NO UTILIZADO
B4	D4
B5	D5
B6	D6
B7	D7

Tabla 3.3. Conexión de pines entre el puerto B y el LCD.
Elaborado por los autores.

Los pines pueden se definen de la siguiente manera:

```
#define LCD_ENABLE_PIN  PIN_B0
#define LCD_RS_PIN      PIN_B1
#define LCD_RW_PIN      PIN_B2
#define LCD_DATA4
#define LCD_DATA5
#define LCD_DATA6
#define LCD_DATA7
```

De esta forma se puede utilizar cualquier pin de los puertos del Microcontrolador.

Ejercicio 3.6. Módulo LCD usando el puerto B.

PROGRAMA:

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)   //F_OSC =12MHz.

#define LCD_ENABLE_PIN  PIN_B0      //Definición de pines para la conexión del LCD al puerto B.
#define LCD_RS_PIN      PIN_B1
#define LCD_RW_PIN      PIN_B2
#define LCD_DATA4       PIN_B4
#define LCD_DATA5       PIN_B5
#define LCD_DATA6       PIN_B6
#define LCD_DATA7       PIN_B7

#include <lcd.c>                //Librería para el manejo del LCD.

void main(void)                 //Función principal main.
{
  lcd_init();                   //Inicializa el LCD.
  lcd_putc("HOLA MUNDO\n LCD en CCS");          //Escribe en el LCD.
}                               //Fin de programa.
```

El mismo resultado se obtiene si se define el uso del puerto B desde el programa incluyendo la instrucción #*define use_portb_lcd TRUE*.

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)   //F_OSC =12MHz.
#include <lcd.c>                //Librería para el manejo del LCD.
#define use_portb_lcd TRUE    //Define uso del puerto B para manejar el LCD.

void main(void)                 //Función principal main.
{
  lcd_init();                   //Inicializa el LCD.
  lcd_putc("HOLA MUNDO\n LCD en CCS");          //Escribe en el LCD.
}                               //Fin de programa.
```

La figura 3.8, muestra las conexiones de un LCD usando el puerto B.

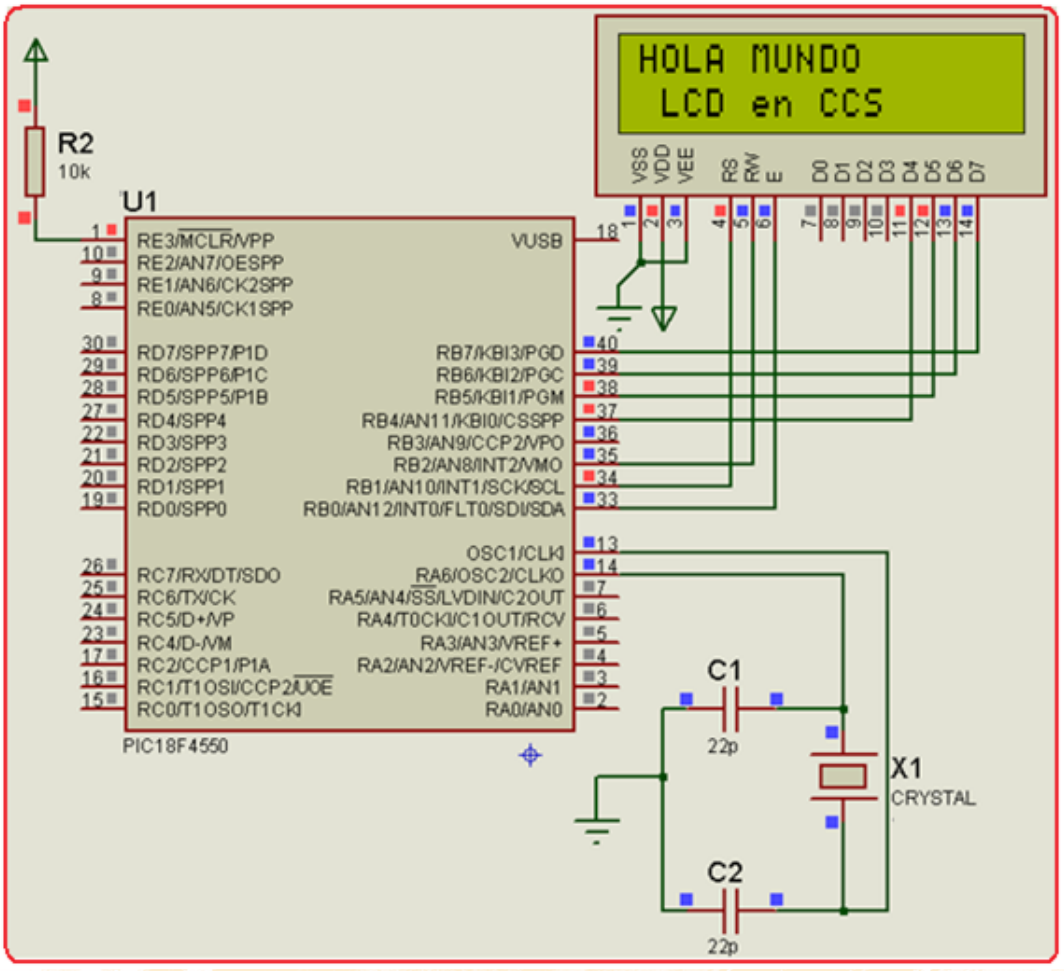


Figura 3.8. Conexión del LCD por el puerto B.
Elaborado por los autores.

Movimiento de texto a la izquierda

En muchas ocasiones para mejorar presentación o llamar la atención, se necesita que el mensaje se mueva. Dado la gran versatilidad del lenguaje C, se puede desarrollar programas para estas situaciones. El siguiente programa el texto se desplaza a la izquierda. El primer carácter aparece en la fila 1, columna 16, transcurrido un corto tiempo (150 ms), se mueve a la columna 15 y aparece el segundo carácter. La secuencia continua de esa forma hasta completar toda la palabra "electromicrodigital". El desplazamiento continúa en la fila 2, repitiéndose el ciclo.

Ejercicio 3.7. Desplazamiento texto a la izquierda.

PROGRAMA:

```
        //MOVIMIENTO DE TEXTO HACIA LA IZQUIERDA
#include <18f4550.h>              //Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOWRT, NOPUT, NOWDT //Configuración de fusibles.
#use delay (clock=12000000)   //F_OSC = 12 MHz.
#include <lcd.c>          /         /Librería para el manejo del LCD.
const char mensaje[]= {"electromicrodigital"}; //Mensaje a desplegarse en el LCD.

void main(void)                   //Función principal main.
{
  char y=1;       /                /Índice fila.
  signed char x=16;               //Índice columnas (posiciones).
  lcd_init();                     //Inicializa LCD.

  while(TRUE){                    //Bucle inicio.
    lcd_gotoxy(x,y) ;             //Ubica cursor para escribir mensaje.
    lcd_putc(mensaje ) ;          //Muestra por pantalla el mensaje.
    delay_ms(150);                //Retardo 150 ms.
    x--;                          //Decremento índice de columnas.
    if(x>-16){                    //¿Se ha mostrado mensaje entero en la primera fila?.
      x=16;       /               /SI, índice columna x=-16, inicia el cursor en la primera posición.
      y++;                        //Incremento índice fila (mostrar texto en segunda columna).
      if(y>2)     /               /¿Se ha mostrado mensaje por segunda fila?.
      y=1;        /               /SI, restaurar índice de fila.
    }
    lcd_putc("\f" ) ;       /     /Borra pantalla.
  }                               //Bucle infinito.
}                                 //Fin del main.
```

Figura 3.9. Captura parte del resultado del desplazamiento del texto a la izquierda.
Elaborado por: Los Autores.

Movimiento de texto a la derecha

Aunque el movimiento de texto a la derecha no es muy común dejar escrito el código para realizar este desplazamiento. El lector puede realizar el análisis del código y comprobar el funcionamiento ya sea en el simulador o armando el circuito con los componentes reales.

Ejercicio 3.8. Desplazamiento del texto hacia la derecha.

PROGRAMA:

```
//MOVIMIENTO DE TEXTO HACIA LA DERECHA
#include <18f4550.h>              //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)   // F_OSC = 12 MHz.
#include <lcd.c>          /         /Librería para el manejo del LCD.
 const char mensaje[]= {"electromicrodigital"};          //Mensaje a desplegarse en el LCD.

void main(void)                   //Función principal main.
{
  char y=1;       /                /Índice fila.
  signed char x=-16;              //Índice columnas (posiciones).

  lcd_init();                     //Inicializa LCD.

  while(TRUE){                    //Inicio del bucle.
    lcd_gotoxy(x,y) ;             //Ubica cursor para escribir mensaje.
    lcd_putc(mensaje):            //Muestra por pantalla el mensaje.
    delay_ms(150);                //Retardo de 150 ms.
    x++;                          //Incremento índice de columnas.
    if(x>16){      /              /¿se ha mostrado mensaje entero en la primera fila?
      x=-16;       /              /SI, índice columna x=-16, inicia el cursor en la primera posición.
      y++;         /              /incremento índice fila (mostrar texto en segunda columna).
      if(y>2)      /              /¿se ha mostrado mensaje por segunda fila?.
      y=1;         /              /SI, restaurar índice de fila.
    }
    lcd_putc("\f" ) ;       /     /Borra pantalla.
  }                               //Bucle infinito.
}                                 //Fin del bucle main.
```

Módulo GLCD

Las nuevas tecnologías de la información han permito que los equipos electrónicos incluyan sistemas multimedia, por lo que el manejo audio, video, gráficos, imágenes, texto, se ha convertido en algo cotidiano. En esta

sección se trata el manejo de gráficos e imágenes utilizando los módulos LCD gráficos.

Los módulos GLCD (Graphic LDC), permiten visualizar caracteres alfanuméricos y gráficos con una gran calidad. La pantalla del GLCD más común tiene las dimensiones 128x64 (X,Y). Hay que tener en cuenta que en la esquina superior izquierda se encuentra ubicado el punto 0,0.

Definición de pines

Para que la librería pueda ser adaptada a las necesidades y definir los pines que manejan el GLCD, se puede definir las conexiones en el programa como se indica:

```
#define GLCD_CS1      PIN_B0
#define GLCD_CS2      PIN_B1
#define GLCD_RW       PIN_B4
#define GLCD_E        PIN_B5
#define GLCD_RESET    PIN_C0
```

Dividir la pantalla del GLCD en dos

El display está dividido en dos mitades de 64x64 pixeles. Al momento de escribir en el GLCD se debe seleccionar cuál mitad vamos a utilizar. Las líneas de control CS1 y CS2, permiten manejar las secciones correspondientes.

Funciones CCS para el GLCD

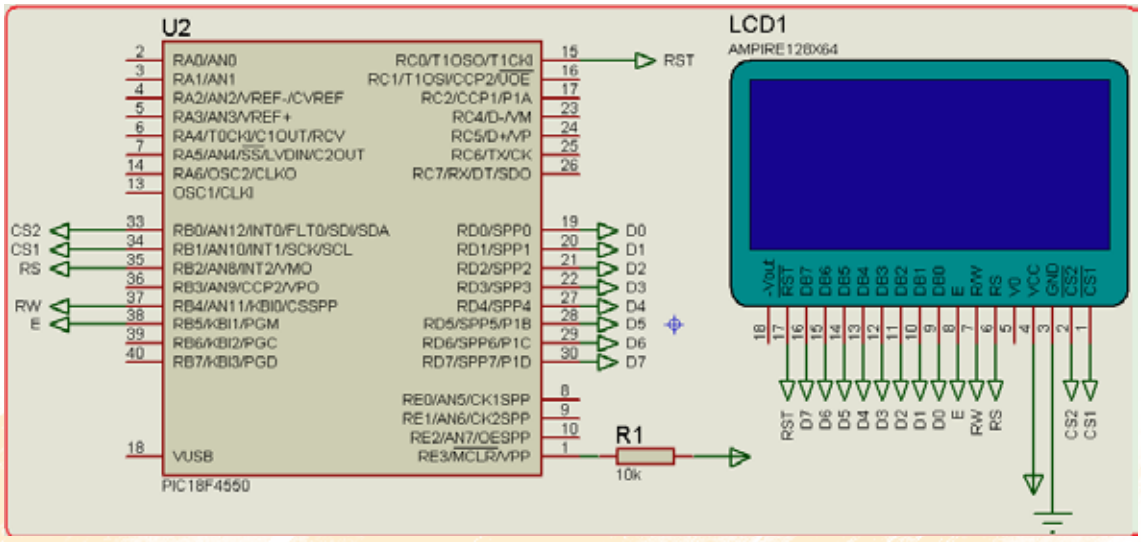
CCS proporciona la librería GLCD_K0108.C y GRAPHICS.C, que presenta una serie de funciones para manejar los módulos GLCD. Las más utilizadas son:

GLCD_init(mode)

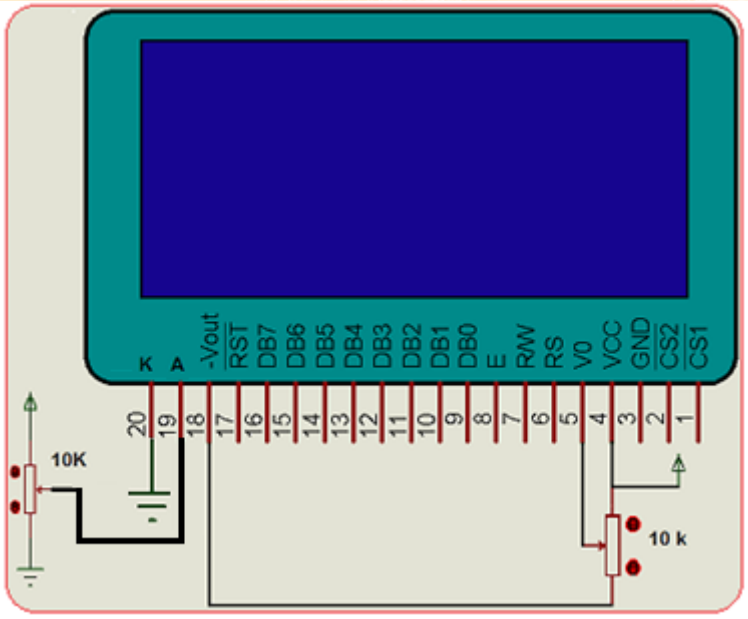
Inicializa el GLCD. Es la primera función del programa para usar el GLCD. El argumento **mode** define el estado del GLCD: **on** (**1)** activado, **off (0)** apagado.

La figura 3.10 a, muestra las conexiones del GLCD con el microcontrolador usado para comprobar el funcionamiento en el simulador. Algunos GLCD vienen con dos terminales para el control del backlight y son el pin 19 corresponde al A (ánodo), y el pin 20 al K (cátodo) para este caso se conecta como indica la figura 3.10 b. El contraste se puede controlar utilizando los

pines –Vout y Vo conectados como se indica en la misma figura, los demás pines se conectan como se indica en la figura 3.10 a.



Conexiones MC - GLCD para pruebas en el simulador.



Conexiones MC – GLCD que dispone terminales de backlight y contraste de imagen.

Figura 3.10. Conexiones del GLCD con el MC.
Elaborado por: Los Autores.

GLCD_fillscreen(color)

Esta es la función que cambia de color la pantalla para indicar que está encendida o pagada. El único argumento (color) determina el estado activado o desactivado de la pantalla.

Con **color** OFF (0); limpia la pantalla (color blanco) y con color ON (1); la pantalla se pone en negro completamente. Se debe aclarar que se obtiene el mismo resultado escribiendo **on** o **1**, u **off** o **0.**

En los programas es importante incluir las librerías <HDM64GS12.c> y <graphics.c> para utilizar las funciones que se están describiendo.

Ejercicio 3.9. En el siguiente programa se utiliza las funciones de inicialización del GLCD y de cambio del color de la pantalla.

PROGRAMA:

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)     // F_OSC = 12 MHz.
#include <HDM64GS12.c> /        /Librería para el manejo del GLCD.
#include <graphics.c>           //Librería gráfica para funciones del GLCD.

void main(){
  glcd_init(1);                 //Inicializa el GLCD
  while(1){
    glcd_fillScreen(off);       //Pinta la pantalla completamente de blanco.
    delay_ms(2000);             //Espera dos segundos.
    glcd_fillScreen(on);        //Pinta la pantalla completamente de negro.
    delay_ms(2000);             //Espera dos segundos.
  }
}
```

La figura 3.11, indica el resultado de la instrucción glcd_fillscreen(0).

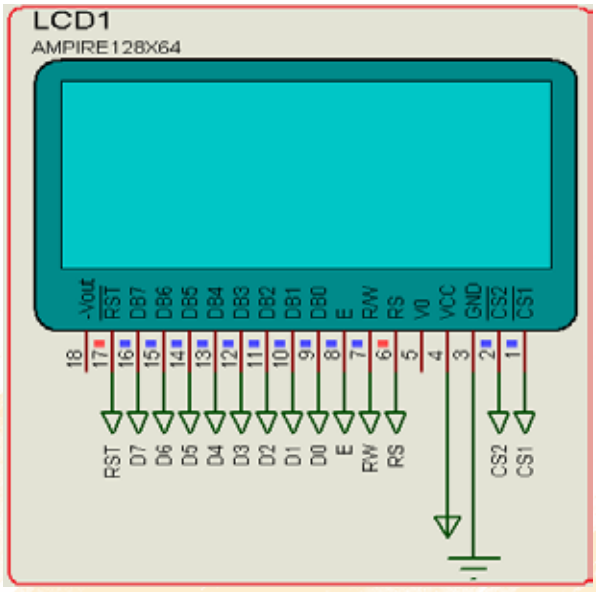


Figura 3.11. Pantalla en blanco o limpia del GLCD.
Elaborado por: Los Autores.

La figura 3.12, indica el resultado de la instrucción glcd_fillscreen(1).

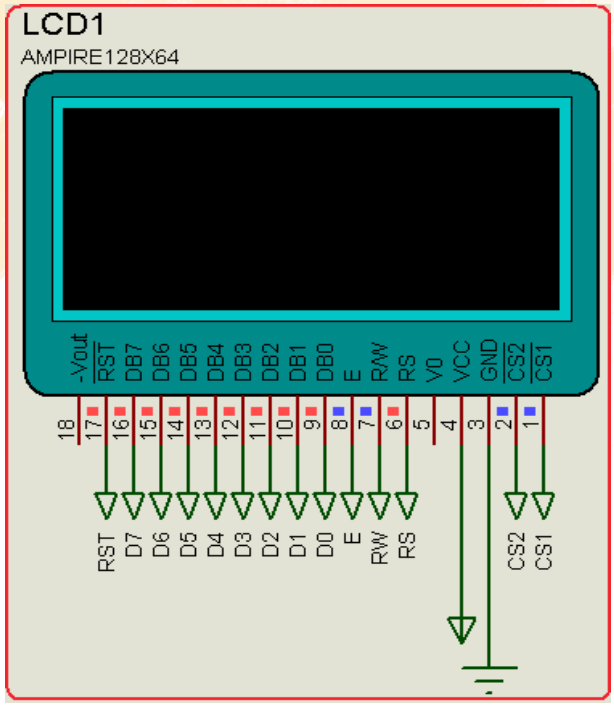


Figura 3.12. Pantalla en negro del GLCD.
Elaborado por: Los Autores.

Dibujo de entidades y figuras primitivas

Las funciones que se detallan sirven para dibujar entidades y figuras primitivas como son puntos, líneas, rectángulos, círculos y textos.

GLCD_pixel(x,y,color)

Dibuja o "pinta" un punto en la pantalla del GLCD. A partir de **GLCD_pixel(x, y, color)** se escribe las funciones gráficas restantes. Los parámetros de ésta función son:

- **x**: un int8, es la coordenada "x" (horizontal), con valores válidos de 0 a 127 (izquierda a derecha).
- **y**: un int8, es la coordenada "y" (vertical), con valores válidos de 0 a 63 (arriba a abajo).
- **color**: activa o desactiva un bit, "0" = apagado, "1" = encendido. También se puede activar con "on" y desactivar con "off".

Ejercicio 3.10. Graficar cuatro puntos en los límites de la esquinas de la pantalla del GLCD.

PROGRAMA:

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)     //F_OSC = 12 MHz.
#include <HDM64GS12.c> /        /Librería para el manejo del GLCD.
#include <graphics.c>           //Librería gráfica para funciones del GLCD.

void main()                     //Función principal.
{
        GLCD_pixel(0,0,1);      //Punto esquina superior izquierda
        GLCD_pixel(0,63,1);     //Punto esquina inferior izquierda
        GLCD_pixel(127,0,1);    //Punto esquina superior derecha
        GLCD_pixel(127,63,1);   //Punto esquina inferior derecha
 }
```

La figura 3.13, presenta el resultado del programa

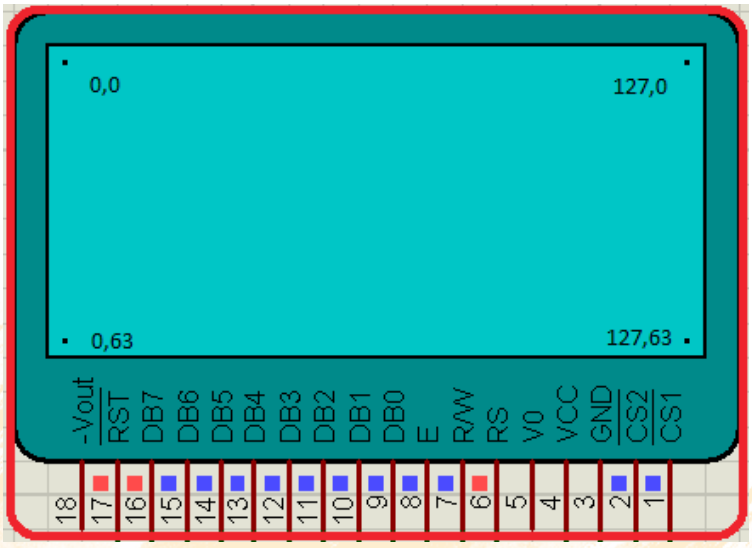


Figura 3.13. Puntos límites de la pantalla del GLCD.
Elaborado por: Los Autores.

Ejercicio 3.11. Graficar una matriz de puntos en toda la pantalla del GLCD.

PROGRAMA:

```
#include <18f4550.h>              //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)       //F_OSC = 12 MHz.
#include <HDM64GS12.c> /          /Librería para el manejo del GLCD.
#include <graphics.c>             //Librería gráfica para funciones del GLCD.

void main()                       //Función principal.
{
   int x,y;                       //Variables para las coordenadas de los pixel.

   GLCD_init(1);                  //Inicializa el GLCD, encendido.
   GLCD_fillscreen(0);            //Limpia la pantalla.
   for(x=0; x<=127; x=x+5){       //Los puntos se pintan cada 5 pixel en el eje x.
      for(j=0; y<=63; y=y+5){     //Los puntos se pintan cada 5 pixel en el eje y.
        GLCD_pixel(x,y,1);        //Pinta el punto.
      }
   }
}          /                      Fin del main.
```

La figura 3.14, muestra el resultado del programa.

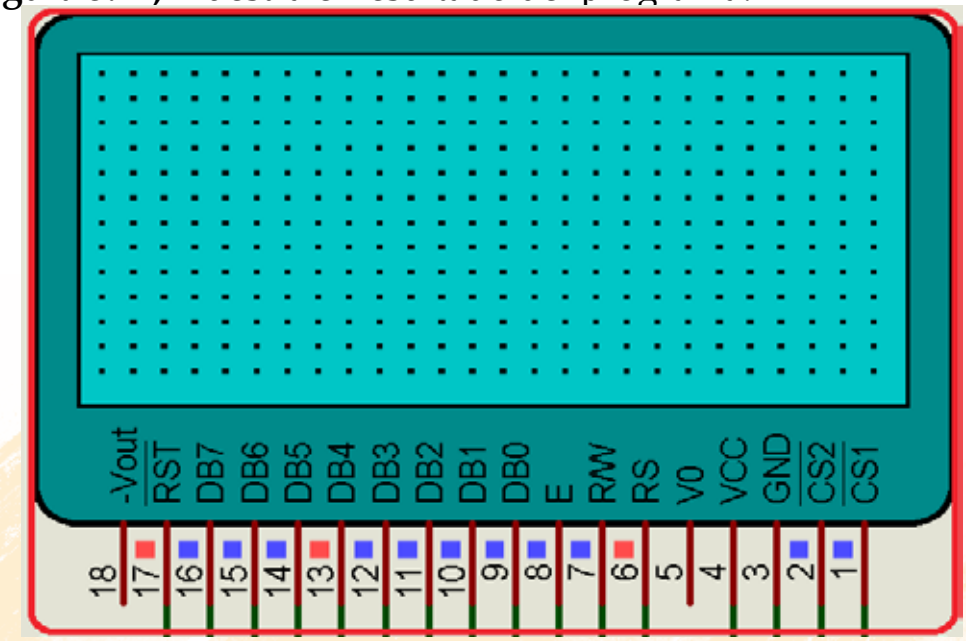


Figura 3.14. Matriz de puntos en la pantalla GLCD
Elaborado por: Los Autores.

GLCD_line(x1,y1,x2,y2,color)

Ésta función sirve dibujar una línea. Los argumentos de la función son:

- **x1,y1,** indica las coordenadas dónde va empezar la línea;
- **x2,y2,** son las coordenadas para el otro extremo de la línea, y
- **color ON (1)** para activar o dibujar la línea y **OFF (0)** para desactivar la línea.

Ejercicio 3.12. Graficar dos líneas diagonales en el LCD.

PROGRAMA:

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)   //F_OSC = 12 MHz.
#include <HDM64GS12.c> /     /Librería para el manejo del GLCD.
#include <graphics.c>          //Librería gráfica para funciones del GLCD.

void main()                    //Función principal.
{
   GLCD_init(1);               //Inicializa el GLCD, encendido.
   GLCD_fillscreen(0);//Limpia la pantalla. Trazo varias líneas diagonales, empieza en la esquina superior.
   GLCD_line(0,0,127,63,1);  // Trazo varias líneas diagonales, empieza en la esquina inferior.
   GLCD_line(0,63,127,0,1);
}
```

GLCD_rect(x1,y1,x2,y2,fill,color)

Ésta función sirve para dibujar un rectángulo. Los argumentos son:

- **x1, y1.** Define la primera esquina del rectángulo y;
- **x2, y2** el extremo opuesto del rectángulo;
- **fill** es para rellenar el rectángulo, colocando **yes (1)** se rellena el rectángulo y **no (0)** sin relleno;
- **color** es para dibujar el rectángulo con **ON (1)**; y con **OFF (0)** no se dibuja el rectángulo.

Ejercicio 3.13. Dibujar un rectángulo relleno en toda la pantalla del GLCD.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)             //F_OSC = 12 MHz.
#include <HDM64GS12.c>                  //Librería para el manejo del GLCD.
#include <graphics.c>                   //Librería gráfica para funciones del GLCD.

void main()                             //Función principal.
{
   GLCD_init(1);                        //Inicializa el GLCD, encendido.
   GLCD_fillscreen(0);                  //Limpia la pantalla.
   GLCD_rect(0,0,127,63,yes,1);         //Dibuja el rectángulo.
}
```

Para dibujar un rectángulo sin relleno se debe modificar en la instrucción *GLCD_rect(0,0,127,63,**yes**,1);* por *GLCD_rect(0,0,127,63,**no**,1);*. El resultado se muestra en la figura 3.15.

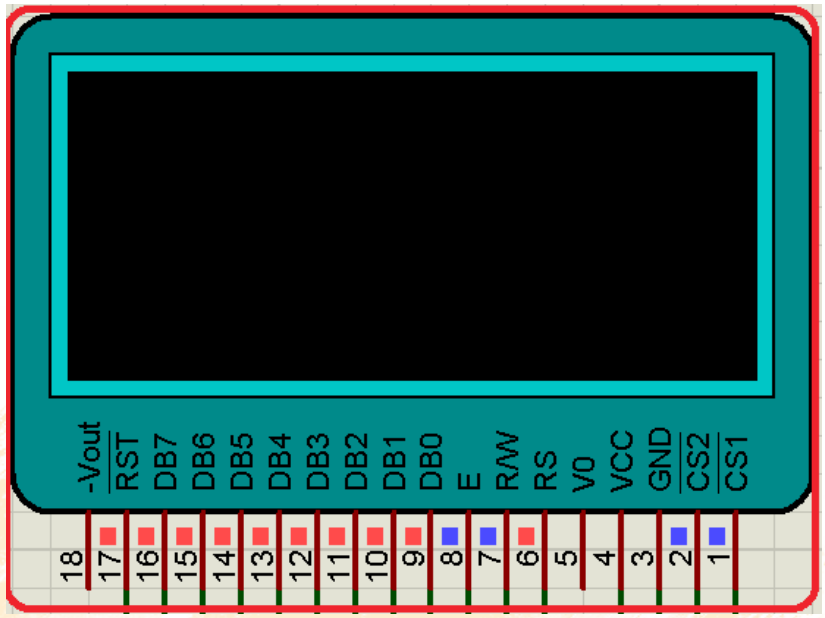


Figura 3.15. Rectángulo con relleno en toda la pantalla del GLCD.
Elaborado por: Los Autores.

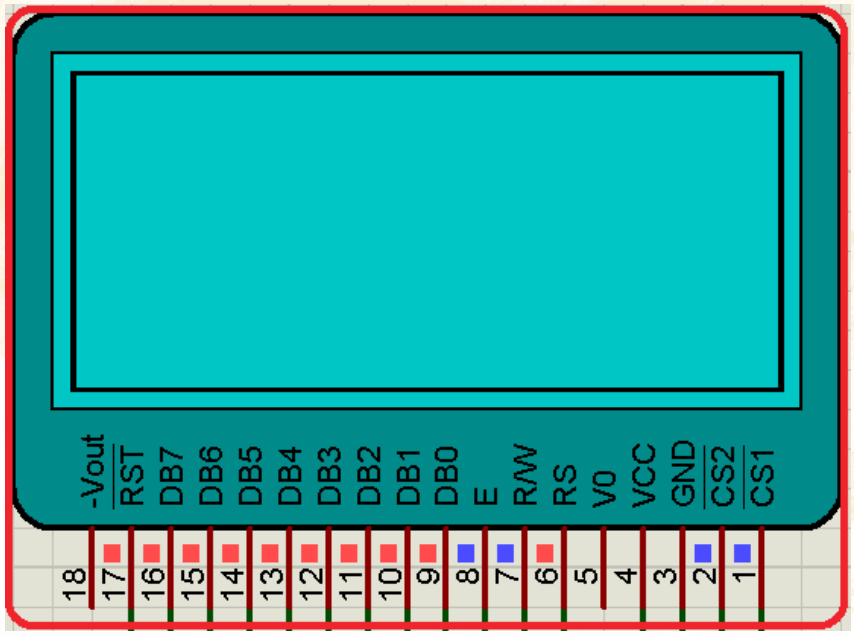


Figura 3.16. Rectángulo sin relleno en toda la pantalla del GLCD.
Elaborado por: Los Autores.

GLCD_bar(x1,y1,x2,y2,width,color)

Esta función dibuja una barra. Los argumentos de la función GLCD_bar son:

- ***x1,y1,*** indica el inicio de la barra***;***
- ***x2,y2***, el fin;

- **width** el ancho de la barra, un valor en pixel, el ancho de la resolución, este valor, depende del GLCD y la posición de la barra y,
- **color** para pintar la barra (**yes**, **1**, pinta; **no**, **0**, no pinta).

Ejercicio 3.14. Graficar una barra en el centro de la pantalla.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)             //F_OSC = 12 MHz.
#include <HDM64GS12.c>                  //Librería para el manejo del GLCD.
#include <graphics.c>                   //Librería gráfica para funciones del GLCD.

void main()                             //Función principal.
{
   GLCD_init(1);                        //Inicializa el GLCD, encendido.
   GLCD_fillscreen(0);                  //Limpia la pantalla.
   GLCD_bar(63,0,63,31,20,1);           //Dibuja el rectángulo.
}
```

La figura 3.17. Indica el gráfico de la barra graficada en el GLCD.

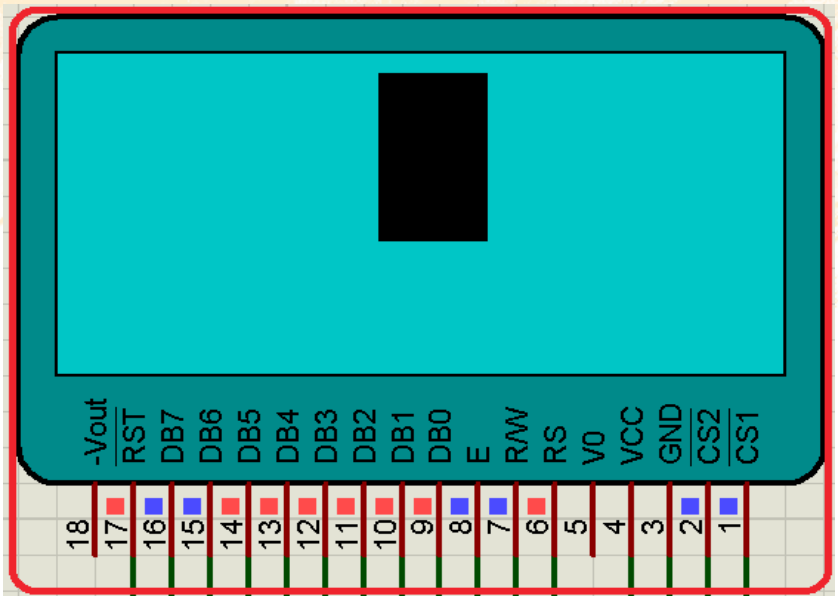


Figura 3.17. Barra graficada en un GLCD.
Elaborado por: Los Autores.

GLCD_circle(x,y,radius,fill,color)

Grafica un círculo. Dónde:

- **x, y** son las coordenadas del centro del círculo,
- **radius,** el radio del círculo,
- **fill** como en los casos anteriores indica si se rellena o no el círculo y
- **color**, para dibujar o no el círculo (**yes**, **1**, pinta; **no**, **0**, no pinta).

Ejercicio 3.15. Graficar un circulo en el centro de la pantalla del GLCD.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)             //F_OSC = 12 MHz.
#include <HDM64GS12.c>                  //Librería para el manejo del GLCD.
#include <graphics.c>                   //Librería gráfica para funciones del GLCD.

void main()                             //Función principal.
{
   GLCD_init(1);                        //Inicializa el GLCD, encendido.
   GLCD_fillscreen(0);                  //Limpia la pantalla.
   GLCD_circle(63,31,20,1,1);           //Dibuja el círculo.
}
```

Para que el círculo se grafique sin relleno la instrucción:

GLCD_circle(63,31,20,1,1);

se debe modificar por:

GLCD_circle(63,31,23,20,0,1);

La figura 3.18, muestra la imagen del círculo con relleno y la figura 3.19, la imagen del circulo sin relleno.

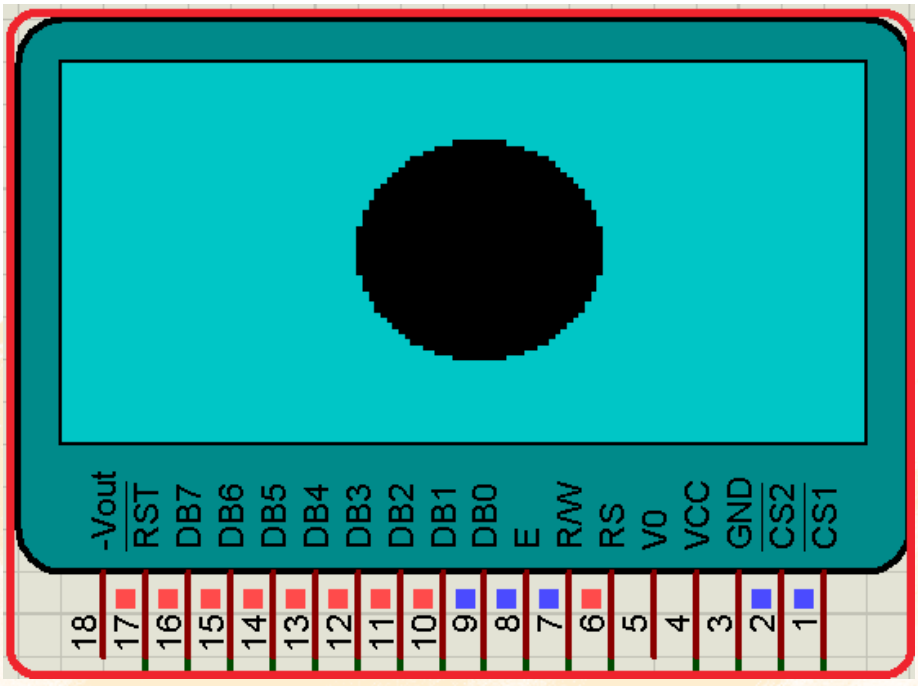


Figura 3.18. Circulo con relleno graficado en el GLCD.
Elaborado por: Los Autores.

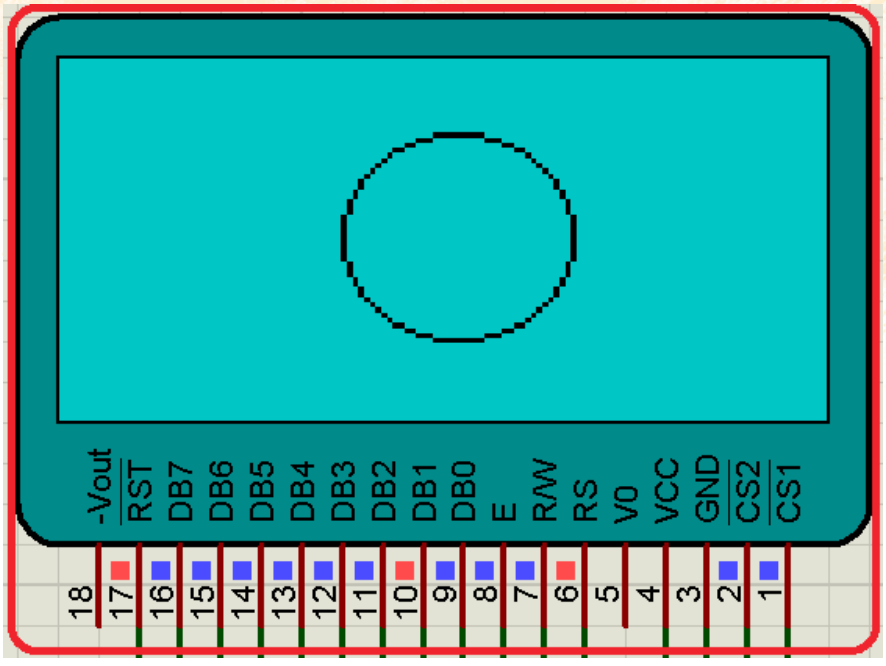


Figura 3.19. Circulo sin relleno graficado en el GLCD.
Elaborado por: Los Autores.

GLCD_text57(x,y,textptr,size,color)

Con ésta función se escribe texto. Los argumentos que vienen incorporados en la función son:

- **x, y,** indican el punto donde se empieza a escribir.
- **textptr**, el texto a mostrar.
- **textptr** es una variable tipo **char** que puede cambiarse usando la función **sprintf.**
- **size** el tamaño de la letra,
- **color** para mostrar el texto: **(on, 1)** o no **(off, 0)**

Ejercicio 3.16. Escribir en el centro de la pantalla del GLCD la palabra PIC MICRO.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)             //F_OSC = 12 MHz.
#include <HDM64GS12.c>                  //Librería para el manejo del GLCD.
#include <graphics.c>                   //Librería gráfica para funciones del GLCD.

void main()                             //Función principal.
{
char TEXTO[] = "PIC MICRO";
  GLCD_init(1);                         //Inicializa el GLCD, encendido.
  GLCD_fillscreen(0);                   //Limpia la pantalla.
  GLCD_text57(31,31,TEXTO, 1,1);        //Escribe el texto desde la posición 31,31 con ancho de 1.
}
```

La figura 3.20, muestra el resultado obtenido.

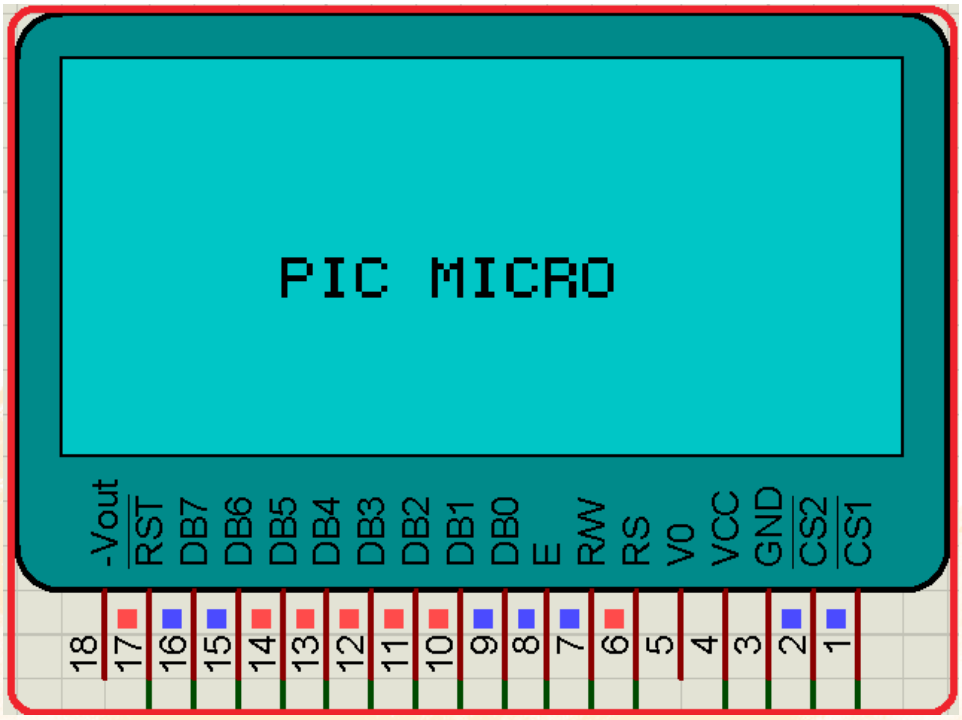


Figura 3.20. Texto con ancho 1 en la pantalla del GLCD.
Elaborado por: Los Autores.

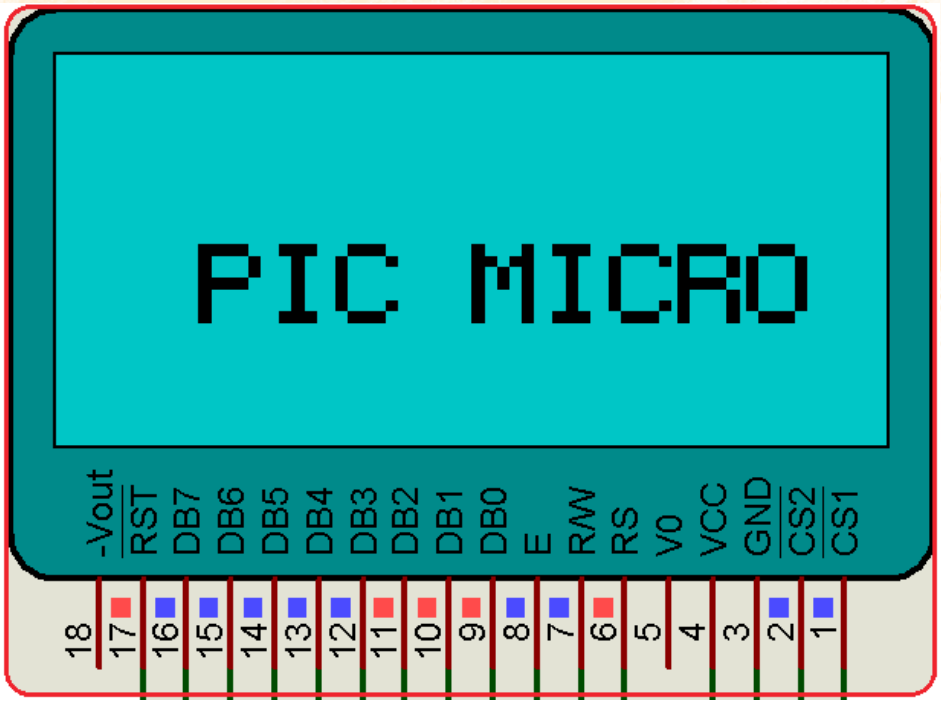


Figura 3.21. Texto con ancho 2 en la pantalla del GLCD.
Elaborado por: Los Autores.

Es muy común que se requiera enviar información numérica. Para este caso se debe realizar una conversión a texto para visualizar en el LCD. La función **sprint** permite realizar esta conversión. El formato de la función sprintf es:

SPRINTF(string, cstring, values...);

Donde, los argumentos son:

- **string,** array de caracteres,
- **cstring**, una constante o un array,
- **values,** lista de variables, separadas con comas.

Ejercicio 3.17. Enviar un dato numérico a la pantalla del GLCD, simulando que es una variable eléctrica.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)             //F_OSC = 12 MHz.
#include <HDM64GS12.c>                  //Librería para el manejo del GLCD.
#include <graphics.c>                   //Librería gráfica para funciones del GLCD.
#include <stdlib.h>                     //Librería para búsqueda y ordenamiento de datos
#include <math.h>                       //Librería para cálculos matemáticos.

char Voltaje[9];                        //Array de 10 elementos tipo char
 float x= 50.123457;                    //Variable x definida con valor.
 char Magnitud[] = "Voltaje="           //Magnitud eléctrica a desplegarse en la pantalla.

void main()                             //Función principal.
{
  GLCD_init(1);                         //Inicializa el GLCD, encendido.
  GLCD_fillscreen(0);                   //Limpia la pantalla.
  sprintf(Voltaje, "%f", (float)x);     //Convierte el valor numérico a texto.
  Voltaje[5] = '\0';                    //Limita a 4 dígitos para mostrar el valor de Voltaje.
  glcd_text57(35, 8, Magnitud, 1, ON); //Escribe Voltaje en las coordenadas 35, 8.
  glcd_text57(45, 20, Voltaje, 1, ON);  //Escribe el valor de la variable Voltaje.
}
```

El resultado del programa se indica en la figura 3.22.

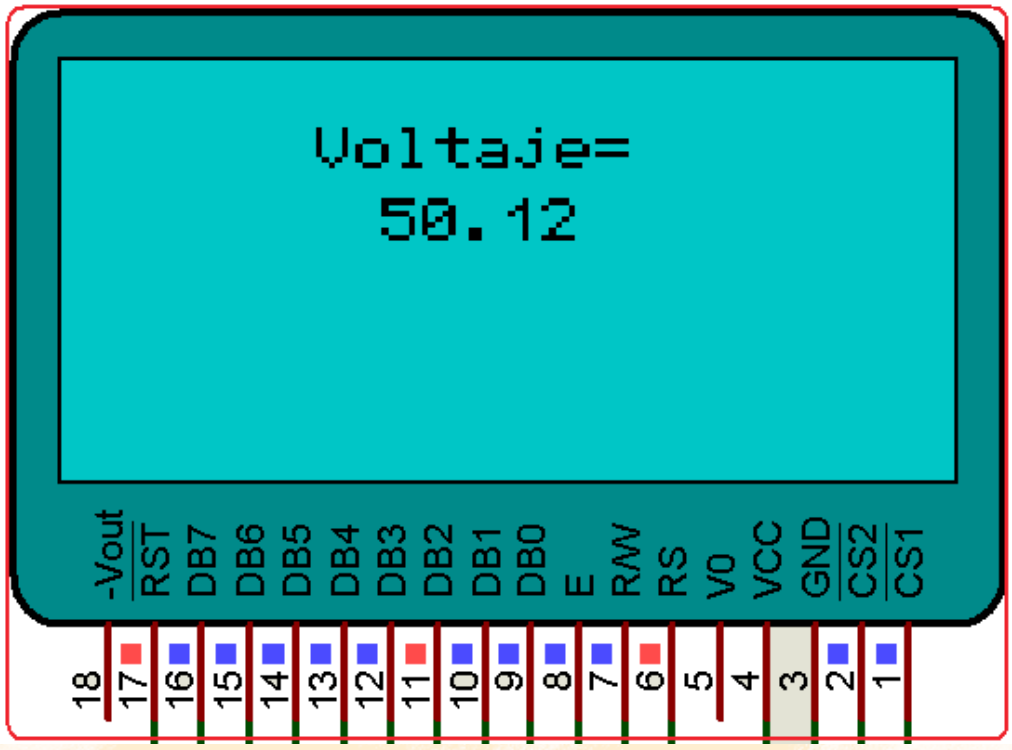


Figura 3.22. Caracteres en el GLCD.
Elaborado por: Los Autores.

Tratamiento de imágenes BMP O JPG

Las nuevas herramientas (software y hardware) de las tecnologías de información exigen el manejo de elementos de Multimedia, razón por la cual, otra de las funciones del microcontrolador es el manejo de imágenes a través de GLCD.

A menudo se requiere personalizar una imagen y presentarlas en la pantalla del GLCD. Para esto es necesario la imagen convertir en una tabla de datos que representan los pixeles de la imagen para luego graficarlos en la pantalla. Existen una serie de programas que realizan estas conversiones entre los que se poden citar los más utilizados: Bitmap2LCD, LCDAssistant, BMP-LCD, LCD-IMAGE-CONVERT, etc. En este caso se ha utilizado el LCD-IMAGE-CONVERT por ser de libre distribución y licencia GNU-GPL. El programa en mención permite cargar una imagen BPM o JPG. Es importante que la imagen deba tener una resolución de 8 bits y ser monocromática.

La figura 3.23, indica la interfaz del programa LCD-IMAGE-CONVERT.

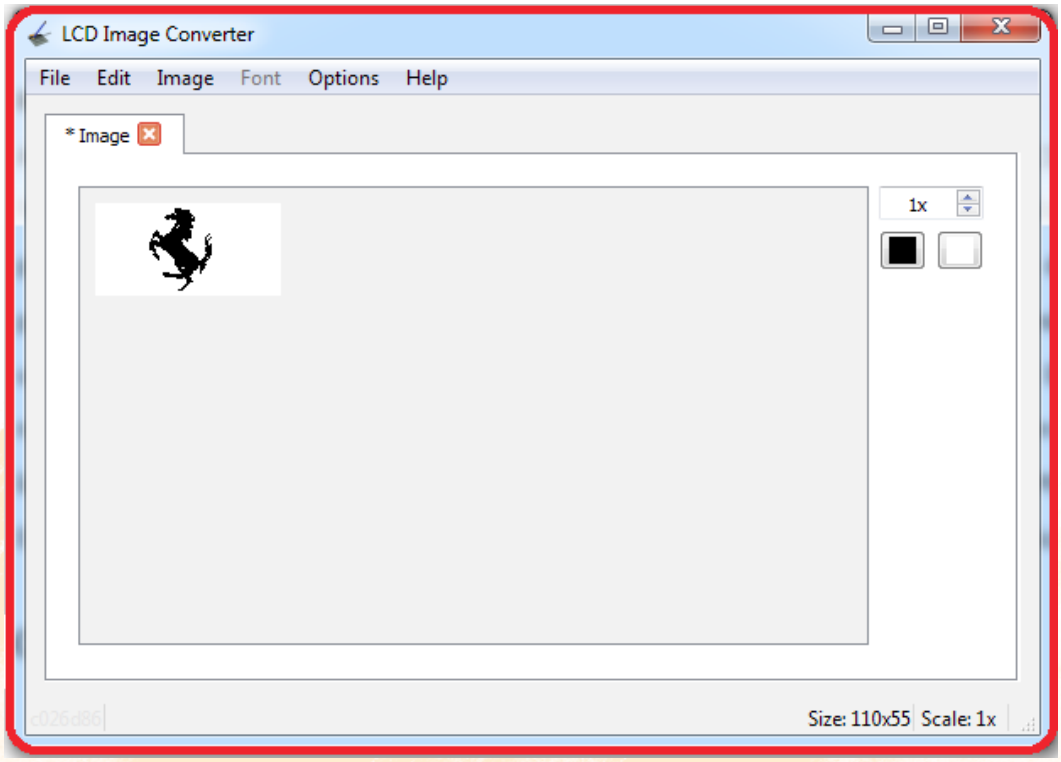


Figura 3.23. Interfaz del programa LCD-IMAGE-CONVERT.
Elaborado por: Los Autores.

Para cargar la imagen se utiliza la pestaña IMAGE/IMPORT y se selecciona la imagen a ser tratada. El proceso de conversión de la imagen es sencillo. En la pestaña OPCION/CONVERSION, se prepara la imagen de salida.

Type: Monochrome (monocromático) y Diffuse Dither (tramado de difusión). También se ha seleccionado Inverse, para invertir el fondo y la imagen del original y así presentar en negro la imagen en el GLCD. Las demás opciones del programa pueden utilizarse según las necesidades. La figura 3.24, detalla las opciones indicadas: Monochrome, Difuse dither e Inverser.

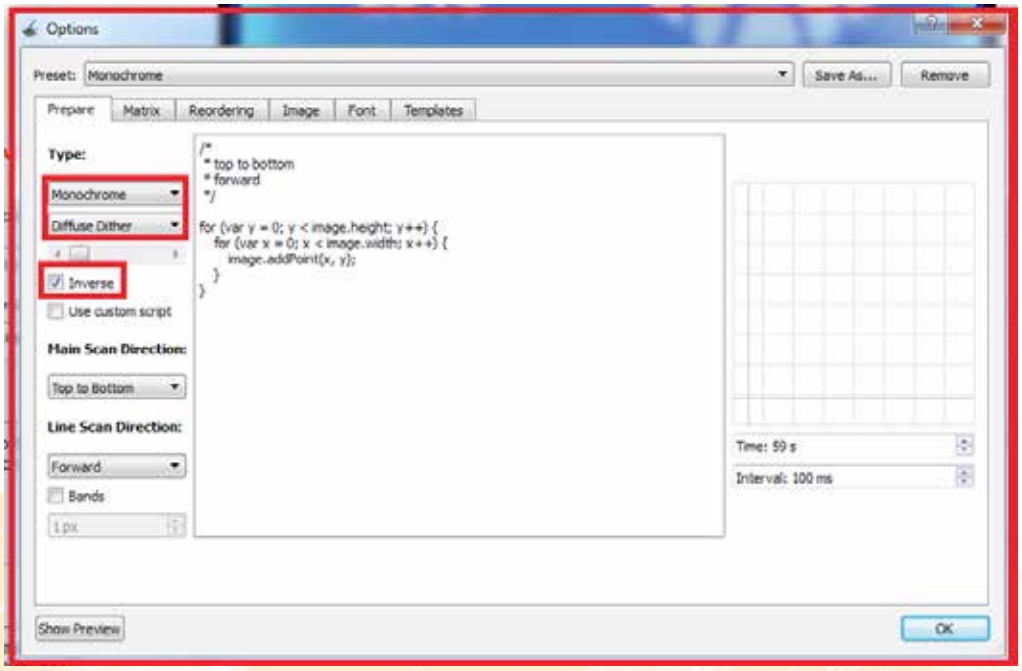


Figura 3.24. Definición del tipo de imagen a ser visualizado.
Elaborado por: Los Autores.

Seguido de esto es importante definir el ancho del bloque (BLOCK SIZE = 8 bits), la separación de las filas (Split to rows), el prefijo utilizado (0x para notación hexadecimal) y el separador de los datos *comas* (,). La figura 3.25, presenta los cambios realizados.

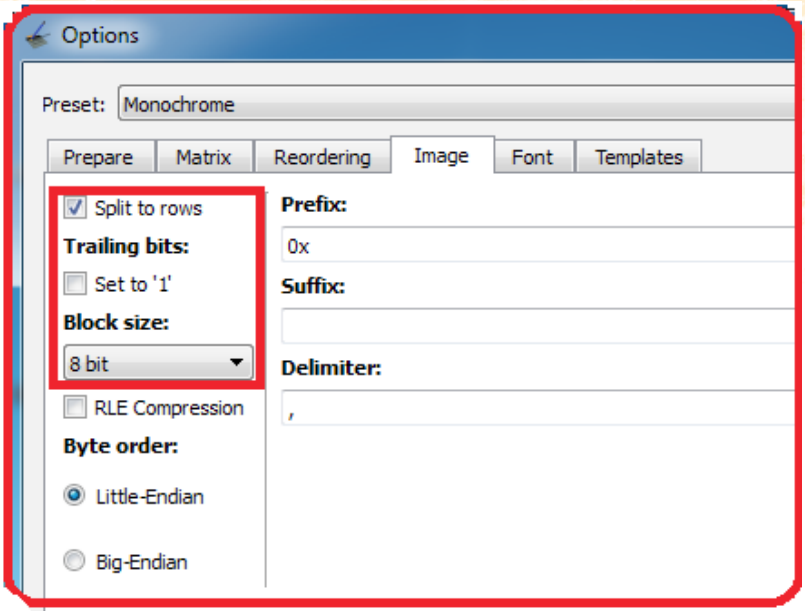


Figura 3.25. Definición del tamaño de la tabla.
Elaborado por: Los Autores.

Se puede hacer una vista previa con el botón SHOW PREVIEW, para observar la tabla generada, la imagen original y la imagen resultante, como se indica en la figura 3.26.

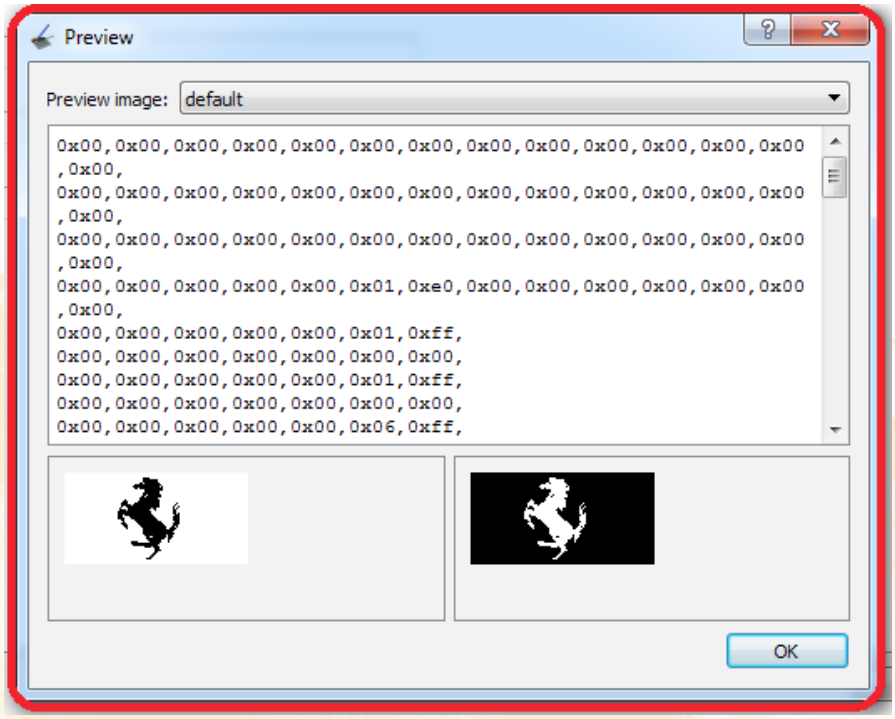


Figura 3.26. Resultado del proceso de conversión de la imagen BMP.
Elaborado por: Los Autores.

Finalmente se procede a convertir la imagen (FILE/CONVERT). Se guarda el archivo con extensión .c. En este archivo se genera el código en c y la tabla de datos respectivos, como se indica en la figura 3.27.

```
#include <stdint.h>
static const uint8_t image_data_Image[770] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x01,0xe0,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x01,0xff,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x01,0xff,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x06,0xff,0x80,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x3f,0xff,0xc0,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x7f,0xff,0xa0,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x37,0xff,0xe0,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xe0,0x00,0x00,0x00,0x00,0x00,
```

Figura 3.27. Tabla parcial de datos de la conversión de la imagen.
Ejercicio 3.18. Mostar una imagen BMP en la pantalla de un GLCD.

Para determinar el tamaño de la tabla, se debe contar el número de filas y columnas de la tabla. En este caso se tiene: ***const int8 imagen[55][14]***

PROGRAMA:

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)     //F_OSC = 12 MHz.
#include <HDM64GS12.c> /        /Librería para el manejo del GLCD.
#include <graphics.c>           //Librería gráfica para funciones del GLCD.
#include <stdlib.h>             //Librería para búsqueda y ordenamiento de datos
#include <math.h>               //Librería para cálculos matemáticos.

char Voltaje[9];                //Array de 10 elementos tipo char
 float x= 50.123457;            //Variable x definida con valor.
 char Magnitud[] = "Voltaje=" //Magnitud eléctrica a desplegarse en la pantalla.

//Pegar la tabla de datos generados en el archivo .c de la conversión de imagen BMP.
//Para el ejemplo por el tamaño de la misma se ha pegado sola una parte de la tabla.
const int8 imagen[55][14] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   ……
   ……
};

//Función para dibujar la imagen.
void glcd_imagen()
{
  char i,j;
  signed char k;
  for( i = 0 ; i < 55 ; i ++ )
  {
    for( j = 0 ; j < 14 ; j ++)
    {
      for(k=7;k>-1;k--)
      {
        if( bit_test(imagen[i][j] ,7-k ))
          glcd_pixel( j*8+k,i, ON );

      }
    }
  }
}
```

```
void main()                    //Función principal.
{
  GLCD_init(1);                //Inicializa el GLCD, encendido.
  GLCD_fillscreen(0);          //Limpia la pantalla.

while(TRUE){

      glcd_imagen();           //Llamamos a la función imagen para dibujar.
      delay_ms(4000);          // en el Display gráfico.
      glcd_fillScreen(0);      //Limpia la pantalla.

    }                          //Ciclo infinito.

     }                                 //Fin del main.
```

La figura 3.28, presenta el resultado de la presentación de la imagen BMP en el LCD.

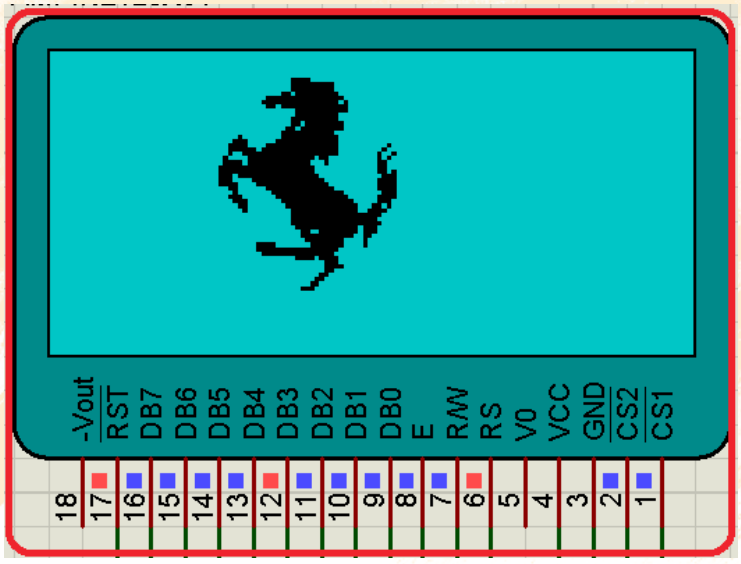


Figura 3.28. Imagen BPM en la pantalla del GLCD.
Elaborado por: Los Autores.

Teclado Matricial

En los PICs, una de las formas más comunes y útiles para ingresar datos es mediante los teclados matriciales. La figura 3.29, presenta el diagrama de conexiones de un teclado matricial (4 x 4).

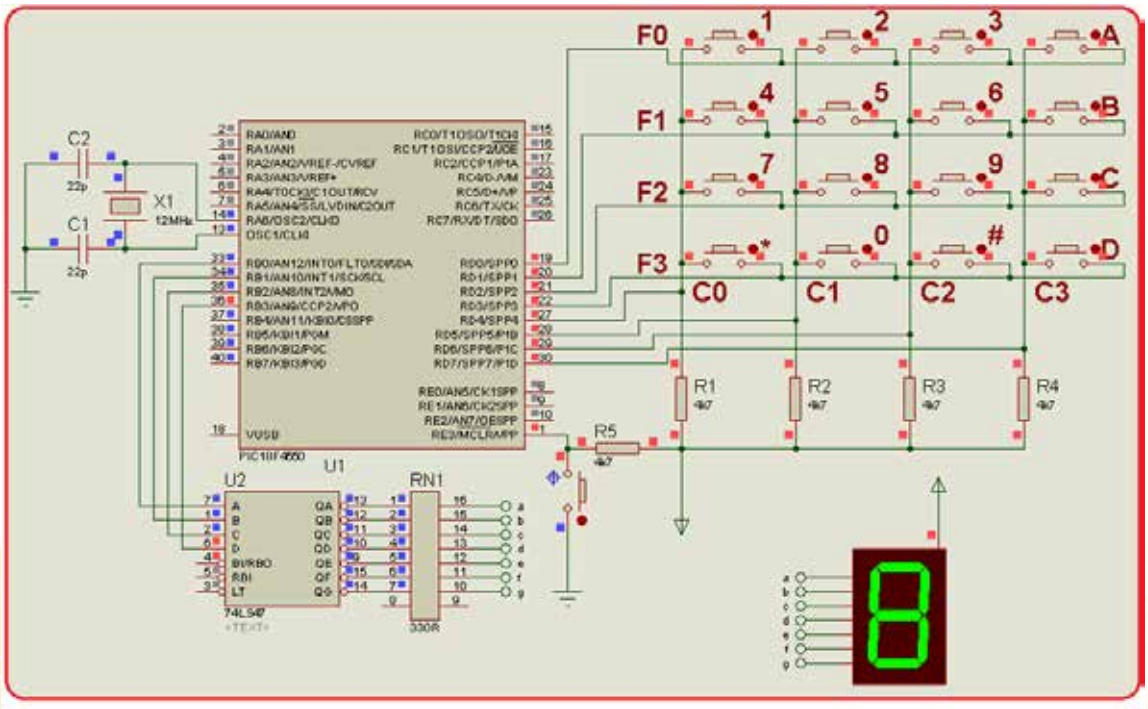


Figura 3.29. Conexión de teclado matricial con PIC.
Elaborado por: Los Autores.

Descripción del funcionamiento:

Los puertos RDO, RD1, RD2 y RD3 se establecen como filas F0, F1, F2 y F3 y se definen como salidas. Las columnas C0, C1, C2 y C3 de los puertos RD4, RD5, RD6 y RD7; están fijadas como entradas.

El proceso para censar cual pulsador ha sido activado, se basa en que se todas se filas se colocan en 1 y se empieza a sacar por la primera fila un 0 (F0=0) y se procede a comprobar si se activado algún pulsador desde la columna 0, por ejemplo, suponer que se accione el pulsador 1, el circuito se cierra y la columna C0 es 0 (C0= 0), está condición F0= 0 y C0= 0, se instrucción produce sólo cuando se acciona el pulsador 1, que equivale a la tecla 1 y mediante software si F0= 0 detecta mediante la if (C0 == 0){ x=1; antirebote(); }, dado las condiciones se asigna a la variable *x* el valor de 1. Se llama a la función *antirebote* y se procede a sacar el valor de *x* en el puerto b, para indicar el valor en el *display*. El proceso de *barrido* del *teclado* continúa indefinidamente.

La subrutina *antirebote*, elimina los posibles "rebotes" que ocasiona los contactos mecánicos del pulsador y que darían varios pulsos interfiriendo al funcionamiento normal del teclado.

La instrucción *while(input(C1) == 0) { },* detiene el funcionamiento del Micro (no realiza nada) hasta que se estabilice el pulsador y quede en estado alto el puerto respectivo. Una vez que el pulsador está en reposo el valor de la variable *x* se asigna al puerto B, como estos valores son binarios, mediante el decodificador de 7447 se transforma a 7 segmentos para visualizarlos en el display.

Ejercicio 3.19. Manejo de teclado matricial y visualización en display de 7 segmentos.

PROGRAMA:

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS, WDT, NOPROTECT, NOPUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)   //FOSC=12MHz
#BYTE port_b= 0xF81  //Identificador para el puerto b en  la localidad 0xF81
#BYTE port_d= 0xF83  //Identificador para el puerto d en la localidad 0xF83.
#BIT F0 = 0xF83.0             //Asigna en la localidad 0xF83 bit 0 a la variable FA.
#BIT F1 = 0xF83.1             //Asigna en la localidad 0xF83 bit 1 a la variable FB.
#BIT F2= 0xF83.2              //Asigna en la localidad 0xF83 bit 2 a la variable FC.
#BIT F3 = 0xF83.3             //Asigna en la localidad 0xF83 bit 3 a la variable FD.
#define C0 bit_test(port_d,4)  //Define la variable C0 para el port d4.
#define C1 bit_test(port_d,5)  //Define la variable C1 para el port d5.
#define C2 bit_test(port_d,6)  //Define la variable C2 para el port d6.
#define C3 bit_test(port_d,7)  //Define la variable C3 para el port d7.

void antirebote();             //Función antirebote.
int x;                         //Variable x para almacenar el número del pulsador.

void main()                    //Función principal main.
{
set_tris_b(0x00);              //Puerto b como salida.
set_tris_d(0xf0);         //Puerto D4 primeros bits como salida y 4 bits como entrada.
disable_interrupts(GLOBAL); //Todas las interrupciones desactivadas
port_b = 0;                    //Todo el Puerto B = 0.

//Asigna a todas las variables F0, F1, F2 y F3 el valor de 1.
//Estas son las filas del teclado.
F0=1;
F1=1;
F2=1;
F3=1;
```

```
while(TRUE){            //Bucle de barrido.
 F1=0;                  //La fila F1 = 0.
  if (C0 == 0){         //Censa si la tecla de la columna 0 ha sido accionada.
   x=1;                 //Asigna  x = 1. Valor de la primera tecla.
   antirebote();        //Llamado a la función antirebote.
   }

  if (C1 == 0){         //Censa si la tecla de la columna 1 ha sido accionada.
    x=2;                //Asigna  x = 2. Valor de la segunda tecla.
    antirebote();       //Llamado a la subrutina antirebote.

   }
 if (C2 == 0){          //Censa si la tecla de la columna 2 ha sido accionada.
    x=3;                //Asigna  x = 3. Valor de la tercera tecla.
   antirebote();        //Llamado a la subrutina antirebote.
}
 if (C3 == 0){          //Censa si la tecla de la columna 3  ha sido accionada.
    x=10;               //Asigna  x = 10, considerando el caso un teclado hexadecimal.
// Valor de la cuarta tecla.
   antirebote();//Función antirebote.
 }

 F0=1;          //La fila F0 = 1.

 //Comienza el censado de las teclas conectadas en la fila 1, para los números, 4, 5, 6,
//11 (letra B).
 F1=0;          //La fila F1= 0.
  if (C0 == 0){
   x=4;
   antirebote();
   }
  if (C1 == 0){
   x=5;
   antirebote();
   }
  if (C2 == 0){
   x=6;
   antirebote();
   }
   if (C3 == 0){
   x=11;
   antirebote();
   }
F1=1;                   //La fila F1 = 1.
```

```
 //Comienza el censado de las teclas conectadas en la fila 2, para los números 7, 8, 9,
//12 (letra C).
  F2=0;            //La fila F2= 0.
  if (C0 == 0){
     x=7;1
    antirebote();
     }
  if (C1 == 0){
     x=8;
    antirebote();
     }
  if (C2 == 0){
     x=9;
    antirebote();
     }
  if (C3 == 0){
     x=12;
    antirebote();

     }

  F2=1;            //La fila F2 = 1.
 //Comienza el censado de las teclas conectadas en la fila 3, para los números 13, 0,
//14, 15.

  F3=0;            //La fila F3= 0.
  if (C0 == 0){
     x=13;
    antirebote();
     }
  if (C1 == 0){
     x=0;
    antirebote();
     }
  if (C2 == 0){
     x=14;
    antirebote();
     }
  if (C3 == 0){
     x=15;
    antirebote();
     }
```

```
   F3=1;
 }                                  //Fin del bucle infinito.
}                                   //Fin del main.
 void antirebote(void)              //Función anti rebote.
 {          /                        /No realiza nada hasta que el pulsador esté
inactivo.
   while(C0 == 0) { }
   while(C1 == 0) { }
   while(C2 == 0) { }
   while(C3 == 0) { }
   delay_ms(30);                    //Retardo de 30 ms.
          //Puerto b = x. Se envía al puerto b el dato de x, para visualizar en display.
     port_b = x;
     return;
 }
```

Ejercicio 3.20. Manejo de teclado matricial y visualización en LCD.

Con base en el funcionamiento del circuito anterior se va a optimizar el programa realizando un algoritmo más funcional para el manejo del teclado.

Descripción del algoritmo para el programa

Definición de los pines para conectar el teclado. El teclado se conecta al puerto B, las filas F3:F0, corresponden a los pines RB3:RB0; y las columnas C3:C0 en los pines RB7:RB4. Por ejemplo: #*define F3 PIN_B3,* **d**efine la F3 en el pin RB3.

Realizar el mapa de los caracteres que corresponden a las teclas. Mediante la constante *TECLAS* **tipo** *char***,** se declara la matriz 4 x 4.

```
char const TECLAS[4][4] = { {'1','2','3','A'},
                            {'4','5','6','B'},
                            {'7','8','9','C'},
                            {'*','0','#','D'} };
```

Todas las columnas son entradas y tienen en la posición de reposo de los pulsadores 1L, las filas son salidas que se obtienen los valores de 1110, 1101, 1011, 0111. Para obtener estos valores se desplaza una variable n= 1 se invierte el valor y se asigna al puerto B. A continuación se detalla con un ejemplo.

- Inicialmente el puerto b = 11111111 **(***port_b= 0xFF***).**
- *n= 1***.** Como la variable *n* es un *int8* en binario n= 00000001.
- El valor invertido se asigna al puerto B (*port_b= ~n*), por tanto, el puerto B, vale: 11111110
- Al desplazar (*n=n<<1*) un bit la variable *n* vale: 00000010; y el nuevo valor del puerto es: 11111101.
- Como se necesita que el desplazamiento sea hasta el bit del pin RB3, o sea, la fila F3, la variable *count* va incrementando su valor en cada desplazamiento hasta llegar a 3 para reiniciar un nuevo censo. Al mismo tiempo el dato actual de count corresponde a la fila, por lo que se asigna a la variable F que numera las filas del teclado.

Cuando un pulsador se active, la columna que éste el pulsador conectado se pondrá en 0, mediante la función *columnas* se determina si existe una cambio de estado en las columnas: Si *columnas*; están en alto la función devuelve un 1 o si están en bajo devuelve un 0. La operación AND *(input (C0) & input (C1) & input (C2) & input (C3))* entre todas las variables de las columnas (C0, C1, C2 y C3), permite conocer el estado descrito.

Cuando la función columnas devuelve el valor de 0, inmediatamente se procede a determinar el número de la columna en la cual está conectado el pulsador: (*if(input(C0) == 0){ C=0*).

Una vez establecido la fila y la columna procedemos a leer el valor de la matriz *TECLAS* y asignar a la variable *K,* para finalmente indicar el valor en el LCD. Como son pulsadores mecánicos y tienen el problema de rebotes de los contactos, la subrutina *antirebote* previene estas situaciones.

El circuito de la figura 3.30, muestra las conexiones del teclado, el LCD con el microcontrolador.

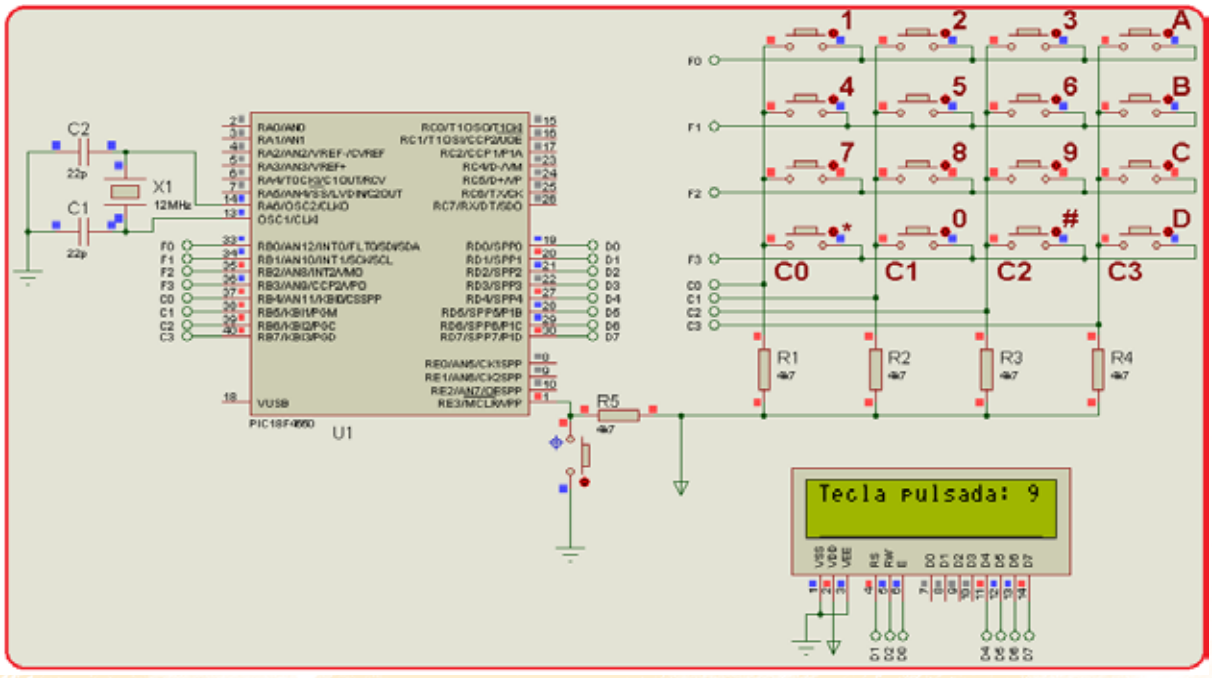


Figura 3.30. Conexión de teclado matricial y LCD con PIC.
Elaborado por: Los Autores.

PROGRAMA:

```
#include <18f4550.h>          //Librería para usar el PIC18F4550.
#fuses HS, WDT, NOPROTECT, NOPUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)   //FOSC=12MHz
#include <lcd.c>              //Incluye librería para manejar el LCD.
#use standard_io(B)           //Usa la librería estándar (io) del puerto B.
#BYTE port_b= 0xF81           //Identificador para el puerto b en  la localidad 0xF81.
#byte tris_b = 0xF93          //Identificador para el registro TRISB  en la localidad 0xF93.
                              //Definición de los pines a utilizar:
#define F0 PIN_B0             //Filas:  F0,F1, F2,F3.
#define F1 PIN_B1
#define F2 PIN_B2
#define F3 PIN_B3
#define C0 PIN_B4             //Columnas: C0,C1,C2,C3.
#define C1 PIN_B5
#define C2 PIN_B6
#define C3 PIN_B7
                              //Declaración de variables.
int count=0;                  //count, cuenta el número de filas.
int8 n =1;                    //Desplaza los ceros (0) en las filas.
char K;                       //Acepta el valor de la tecla pulsada.
int C;                        //Número de Columnas.
int F;                        //Número de Filas.

void antirebote();            //Función antirebote.
```

```
                                        //Distribución del teclado.
char const TECLAS[4][4] = { {'1','2','3','A'},
                {'4','5','6','B'},
                {'7','8','9','C'},
                {'*','0','#','D'} };

                                        //Función columnas: censa el cambio de estado.
short int columnas (void) {
  if(input (C0) & input (C1) & input (C2) & input (C3))
  return (0);
  else
  return (1);
}
void main(){                            //Función principal main.
  lcd_init();                           //Inicializa el LCD.

  printf(lcd_putc, "Pulse una tecla");    //Mensaje en el LCD: "Pulse una tecla".
 set_tris_b(0xF0);                      //Fija el puerto B como entrada(B7:B4) y salida (B3:B0).
 port_b= 0xFF;                          //El puerto B en forma inicial en 1.

  while(true){                          //Bucle infinito.
    if (count<4){                       //Si contador (count) es menor que 4....
      F= count;                         //F recibe 0, 1,2,3 según count se incrementa.
      port_b= ~n;                       //Invierte el valor de n y asigna al puerto B.
      n= n<<1;                          //n desplaza su valor en 1.
      ++count;                          //incrementa count en 1.
      columnas();                       //Detecta si hay cambio en una de las columnas.
     }
     else {                             //Reinicia variables a su valores iniciales.
      count=0;
      port_b= 0xFF;
      n=1;
     }
    if(columnas()) {                    //Si hay un cambio en la columna (columnas= 0)...
     if(input(C0) == 0){                //... y asigna el valor a la columna respectiva.
          C=0;
     }
     else if(input(C1) == 0){
          C=1;
     }
    else  if(input(C2) == 0){

          C=2;
     }
    else  if(input(C3) == 0){
          C=3;
     }
    antirebote();                       //Llama a la subrutina antirebote.
   }
 }                                      //Fin del bucle infinito.
```

```
}                                   //Fin del main.

void antirebote(void)               //Subrutina antirebote.
  {
  while(input(C0) == 0) { }   /    /No realiza nada hasta que el pulsador esté inactivo
  while(input(C1) == 0) { }
  while(input(C2) == 0) { }
  while(input(C3) == 0) { }
  delay_ms(30);
  K =TECLAS[F][C];  //Asigna a K el valor de la tecla ubicada en la fila F, columna C.
  printf(lcd_putc, "\fTecla pulsada: %c", k); //Visualiza el número de la tecla en el LCD.
  return;                           //Retorna al programa principal.

        }
```

Teclado matricial 4x3 controlado con driver kbd.c

El compilador CCS, provee el driver KBD.C, para manejar un teclado matricial 4x3. Incluye las siguientes funciones:

kbd_init(); inicializa el sistema, debe ser la primera función en el programa.

kbd _getc(); devuelve el valor de la tecla pulsada según los datos indicados en la figura 3.31.

```
// Keypad layout:
char const KEYS[4][3] = {{'1','2','3'},
                         {'4','5','6'},
                         {'7','8','9'},
                         {'*','0','#'}};
```

Figura 3.31. Vista parcial de los datos del archivo KBD.C.
Elaborado por: Los Autores.

Modificando esta tabla se puede personalizar de acuerdo al teclado que se disponga.

El archivo KBD.C está diseñado para trabajar con el puerto B o el D. La figura 3.32, indica estas definiciones.

```
#if defined(__PCH__)
#if defined use_portb_kbd
   #byte kbd = 0xF81                    // This puts the entire structure
#else
   #byte kbd = 0xF83                    // This puts the entire structure
#endif
#else
#if defined use_portb_kbd
   #byte kbd = 6                     // on to port B (at address 6)
#else
   #byte kbd = 8                    // on to port D (at address 8)
#endif
#endif

#if defined use_portb_kbd
   #define set_tris_kbd(x) set_tris_b(x)
#else
   #define set_tris_kbd(x) set_tris_d(x)
#endif
```

Figura 3.32. Configuración de puertos para el teclado en el archivo KBD.C.
Elaborado por: Los Autores.

Por defecto el driver KDB.C, está definido para conectar en el puerto D. Activando la línea ***#define use_portb_kbd TRUE***, se puede utilizar el puerto B. La figura 3.33, indica la ubicación del código en el fichero KBD.C.

```
////////////////// The following defines the keypad layout on port D

// Un-comment the following define to use port B
// #define use_portb_kbd TRUE

// Make sure the port used has pull-up resistors (or the LCD) on
// the column pins
```

Figura 3.33. Habilitación del puerto B.
Elaborado por: Los Autores.

Las conexiones del teclado vienen dadas en el mismo fichero como indica la figura 3.34. Estas conexiones pueden ser modificadas según las necesidades.

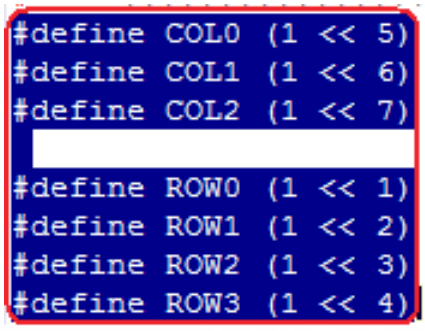


Figura 3.34. Asignación de pines.
Elaborado por: Los Autores.

Con estas definiciones se puede usar en el mismo puerto para manejar el teclado y un LCD.

Ejercicio 3.21. Manejo de teclado 4x3, modificando el driver KBD.C, utilizando PIC16F877A.

Notas:

El archivo KBD.C, ha sido modificado y renombrado para utilizar el puerto B para el teclado y el LCD en el puerto D. Para modificar el driver, abra el fichero KBD.C (se encuentra en el directorio de instalación del PICC, carpeta ***Drivers****) y retire los // que están en la línea de código* ***#define use_portb_kbd TRUE.*** *Renombre por KBDB.C y está listo.*

PROGRAMA:

```
#include <16f877A.h>              //Incluye librería para usar PIC16F877A.
#fuses XT,WDT,NOPROTECT,NOPUT  //Configuración de fusibles.
#use delay (clock=4000000)        //FOSC=4 MHz
#include <kbdb.c>                 //Driver creado  para manejo de teclado por puerto B.
#include <lcd.c>                  //Driver para el LCD por el puerto D.
char k;                           //Variable donde se almacena tecla pulsada.

void main()                       //Función principal main.
{
lcd_init();                       //Inicializa el LCD.
kbd_init();                       //Inicializa el teclado.
port_b_pullups(TRUE);             //Activa las resistencias PULL UP del puerto B.
 printf(lcd_putc,"\fTECLA PULSADA ES:\n");   //Mensaje para la tecla pulsada.
 lcd_gotoxy(8,2);                 //Ubica el cursor en la columna 1, fila 2.
```

```
while(TRUE) {                    //Inicio del bucle...
   k = kbd_getc();               //Detecta la tecla pulsada.
   switch (k)
    {                            //Muestra en el LCD el valor de la tecla pulsada.
     case '0': lcd_putc("0"); break;
     case '1': lcd_putc("1"); break;
     case '2': lcd_putc("2"); break;
     case '3': lcd_putc("3"); break;
     case '4': lcd_putc("4"); break;
     case '5': lcd_putc("5"); break;
     case '6': lcd_putc("6"); break;
     case '7': lcd_putc("7"); break;
     case '8': lcd_putc("8"); break;
     case '9': lcd_putc("9"); break;
     case '#': lcd_putc("#"); break;
     case '*': lcd_putc("*"); break;
     default: break;
    }
   lcd_gotoxy(8,2);              //Ubica el cursor en la 1 columna. fila2
  }                              //Fin del bucle infinito.
}                                //Fin del main.
```

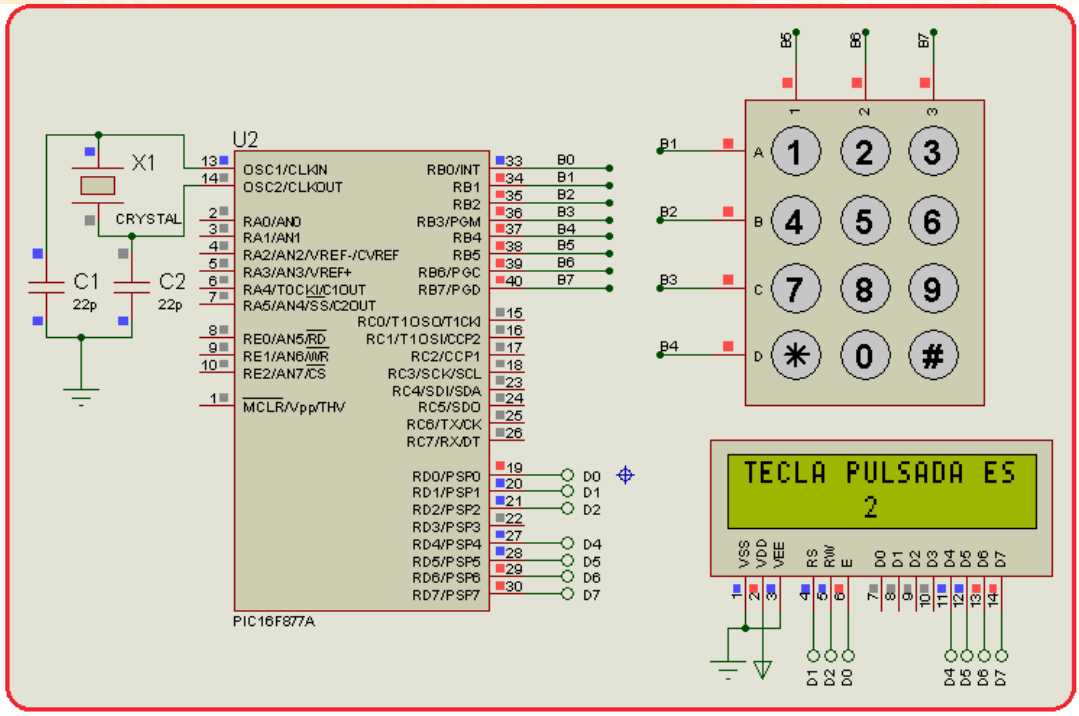


Figura 3.35. Conexiones de teclado 4x3 para PIC16F877A.
Elaborado por: Los Autores.

Ejercicio 3.22. Manejo de teclado 4x3, definiendo el puerto desde el programa utilizando el driver KBD.C y con PIC16F877A.

Sin modificar el archivo KBD.C, se puede directamente en el programa definir el uso del puerto B para el teclado escribiendo la instrucción #*define use_portb_kbd TRUE*. Además se escrito otro código para detectar la tecla, dado que la función *kbd_getc*, devuelve un carácter directamente se puede enviar a imprimir el carácter mediante la función *printf*.

```
#include <16f877A.h>                    //Incluye librería para usar PIC16F877A.
#fuses XT,WDT,NOPROTECT,NOPUT  //Configuración de fusibles.
#use delay (clock=4000000)              //FOSC=4 MHz
#define use_portb_kbd TRUE //Activa el puerto B para manejar el teclado.
#include <kbd.c>                        //Driver para manejo de teclado.
#include <lcd.c>                        //Driver para el LCD por el puerto D.
char k;                                 //Variable donde se almacena tecla pulsada.

void main() {                           //Función principal main.
  lcd_init();                           //Inicializa el LCD.
  kbd_init();                           //Inicializa el teclado.
  port_b_pullups(TRUE);                 //Activa las resistencias PULL UP del puerto B.
  printf(lcd_putc, "Pulse una tecla");     //Solicita que pulse una tecla.
  while(TRUE) {                         //Inicio del bucle...
     k = kbd_getc();          /         /Detecta la tecla pulsada.

    if(k != 0) {                        //Si el carácter es diferente de 0.
      printf(lcd_putc, "\fTecla pulsada: %c", k);//Imprime el valor de k.
    }
  }                                     //Fin del bucle infinito.
}                                       //Fin del main.
```

La forma de modificar el fichero para cambiar el puerto, dependerá de la versión del compilador utilizado, por ello, siempre es conveniente revisar la ayuda de los driver o la descripción de los mismos abriendo los archivos respectivos.

Gestión de la memoria interna EEPROM

La memoria de datos no volátil EEPROM de 256 bytes, se encuentra separada de la memoria RAM y memoria de programa. La memoria EEPROM no es direccionada por los registros comunes de la memoria de programa. Su

direccionamiento se realiza por los SFR (Special Function Special), registros de funciones especiales. La lectura y escritura de la EEROM es realizada en los niveles de voltaje de polarización del circuito integrado (V_{DD}).

Existen cuatro registros especiales que permiten leer/escribir datos en la memoria EEPROM. Estos son:

- EECON1
- EECON2
- EEDATA
- EEADR

El compilador CCS C proporciona una serie de funciones que transparentan al programador el uso de los SFR para el manejo de la EEPROM. El lector interesado en profundizar sus conocimientos en el uso y detalles de los SFR, necesarios para programar en assembler, puede consultar el datasheet el PIC18F4550, disponible en la página oficial de Microchip Technology INC.

Las funciones que CCS proporciona para trabajar con la memoria interna EEPROM son:

value = read_eeprom (address): función básica para leer el valor de la EEPROM interna del PIC. Devuelve un valor entero (int8) de un byte. ***address*** puede ser un entero de 8 o 16 bit, indica la dirección de la localidad de memoria de donde se va a leer el dato y almacenar en la variable ***value.*** Recuerde que el PIC18F4550 dispone de 256 bytes que se direccionan de 0x00 a0xFF.

write_eeprom (address, value): Con esta función se escribe un dato (entero de 8 bits) en la dirección especificada en ***address*** en la memoria interna del PIC. Al igual que ***read_eeprom address*** puede ser un entero de 8 o 16 bits. La ejecución de ésta instrucción puede tomar varios milisegundos.

#rom adress: Esta directiva permite insertar una lista de hasta 84 datos dentro de la memoria de programa (memoria flash) o en la memoria EEPROM. Tiene tres formatos:

#rom address = {list}. La lista de elementos ***list*** son enteros de 16 bits.

#rom int8 address = {list}. La lista de elementos ***list*** son enteros de 8 bits.

#rom char address = {list}. La lista de elementos ***list*** son caracteres de 8 bits.

Hay que considerar que la directiva ***#rom*** crea un segmento para los datos dentro del espacio de la memoria de programa o en la EEPROM. Por tanto, si se requiere escribir los datos en la memoria EEPROM, ***address*** no debe direccionar las localidades del programa, sino a la dirección de la EEPROM del PIC. Para el PIC18F, PCW direcciona la EEPROM en la localidad 0xF00000. Los PIC16F, utilizan la dirección 0x21000.

Ejercicio 3.23. Manejo de la memoria EEPROM.

En este ejercicio se almacena en la memoria EEPROM a partir de la dirección 00 los números 3, 5, 1, 7, 2, utilizando la directiva #rom.

PROGRAMA:

```
#include <18f4550.h>            //Librería para usar el PIC18F4550
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=4000000)      //F_OSC =4MHz
#rom int8 0xF00000= {'5','3','1','7','2'}  //Datos para la EEPROM.
void main()
 {
                //……. Programa
 }
```

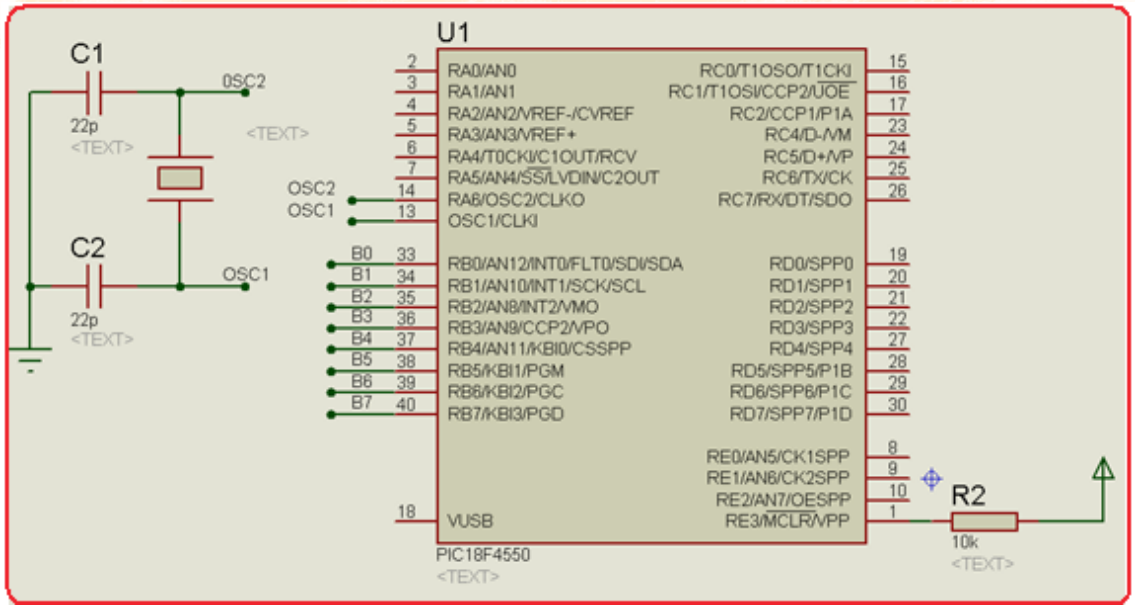


Figura 3.36. Circuito en ISIS para comprobar la escritura de datos en la EEPROM.
Elaborado por: Los Autores.

Para comprobar los resultados en ISIS, en el menú principal y opción DEBUG, poner a "correr" el programa, así: *DEBUG/Start/Restart/Debugging*. Utilizar el menú contextual del PIC (dar un clic con el botón derecho sobre el PIC) y seleccionar *PIC18CPU/Eprom memory-U1*. Se desplegará la tabla de datos como indica la figura 3.37.

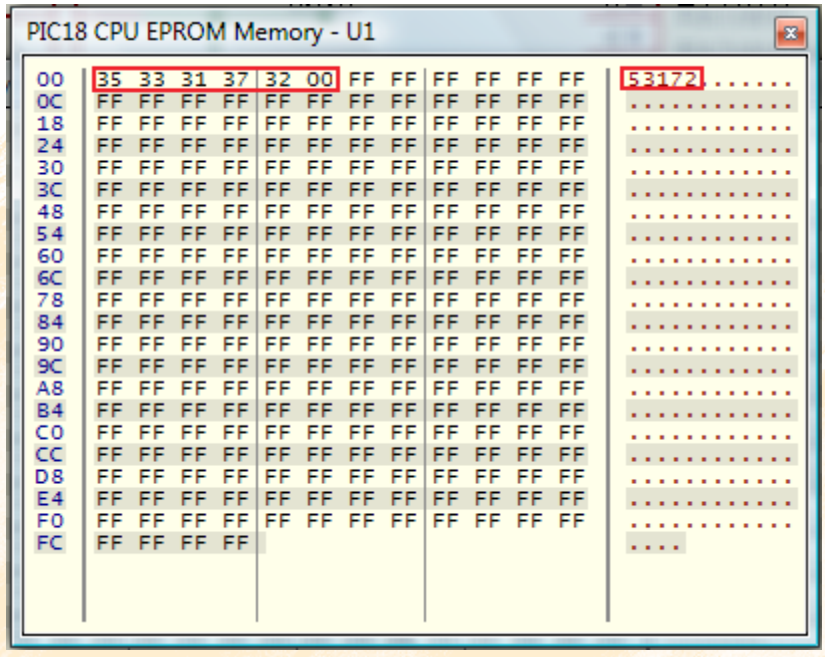


Figura 3.37. Resultados en ISIS de la escritura en la EEPROM para PIC18F4550.
Elaborado por: Los Autores.

Ejercicio 3.24. Abrir una puerta utilizando una clave de seguridad. Condiciones del programa:

- Almacenar la clave en la memoria EEPROM. Clave 3, 7, 5.
- En un LCD se pedirá que ingrese el primero, segundo y tercer digito de la clave.
- Si la clave es correcta en el LCD se indicará el mensaje "Puerta abierta". Un LED conectado en el puerto RA0 se activará por 5 segundos. Esta señal se puede utilizar para acoplar a un circuito para manejar el accionamiento eléctrico de la puerta.
- Si la clave es incorrecta en el LCD se indicará el mensaje "Puerta cerrada". Un speaker conectado en el puerto RA1 se acciona simulando a una alarma.
- Utilizar el PIC16F877A.

Nota: Recuerde modificar el archivo TONES.C, para definir el pin A1, por donde se generará el sonido. El código es:

#define TONE_PIN PIN_A1

Cree otro archivo, TONESA por ejemplo, para no modificar el original.

PROGRAMA:

```
#include <16f877A.h>                        //Incluye librería del PIC16F877A
#fuses XT,NOWDT,NOPROTECT,NOLVP             //Configuración de fusibles.
#use delay(clock= 4000000)      //F_OSC= 4MHz.
#define use_portb_kbd TRUE      //Define el uso del puerto B para el teclado.
#include <TONESA.c>             //Incluye la librería TONESA para generar sonido.
#include <lcd.c>                //Incluye librería LCD.
#include <kbd.c>                //Incluye librería para manejar teclado.
#rom 0x2100={'3','7','5'}       //Graba en la ROM la clave 3 7 5.

 void alarma();                 //Define la función alarma.

void main() {                   //Función principal main.
  char k;                       //Variable K para almacenar dato del teclado.
  int i;                        //Variable para contar el número de datos que ingresa.
  char data[3], clave[3];       //Vectores para almacenar datos del teclado y de la clave.
  lcd_init();                   //Inicializa el LCD.
  kbd_init();                   //Inicializa el teclado.
  port_b_pullups(TRUE);         //Activa las resistencias PULL UP del puerto B.

  while (TRUE) {                //Bucle infinito.
    i=0;                        //Variable i, almacena cuantas teclas se ha pulsado.
    printf(lcd_putc,"\fingrese digito 1\n");        //Pide ingresar el primer digito.
      while(i<=2){              //Mientras i menor o igual a 2.
        k=kbd_getc();           //Asigna a la variable k la tecla pulsada.
        if (k!=0)               //Si k no es igual a 0.
          {data[i]=k;           //almacena en el vector data[i] el dato de la tecla ingresada.


          i++;                  //incrementa i en 1
          printf(lcd_putc,"\fingrese digito %u\n",i+1); //ingresa la tecla según indica i.
        }
    }
```

```
  for (i=0;i<=2;i++) {          //Bucle para leer datos de la eeprom
    clave[i]=read_eeprom(i);}  //Lee y asigna en el vector clave[i] según indica i.
                               //Si clave igual a dato ingresado... abre la
     if ((data[0]==clave[0])&&(data[1]==clave[1])&&(data[2]==clave[2])){
       printf(lcd_putc,"\fPuerta Abierta");  //Puerta abierta
      output_high(PIN_A0);     //Señal para manejar un relé para abrir la puerta.
       delay_ms(5000);         //Retardo 5 s.
       output_low(PIN_A0);}    //Apaga puerta
    else
    {              //Si no son igual dato = clave. La puerta permanece cerrada.
      printf(lcd_putc,"\fPuerta Cerrada");
      for (i=1;i<=5;i++){      //Acciona alarma
        alarma();              //Función alarma.
      }
    }
  }                            //Fin del bucle.
}                              //Fin del main.

void alarma (void)             //Función alarma. Genera sonido que simula una alarma.
{
int fusa=62;
int semicorchea=125;
int corchea=250;

  generate_tone(C_NOTE[0],fusa);
  generate_tone(C_NOTE[1],fusa);
  generate_tone(Eb_NOTE[2],corchea);
  generate_tone(C_NOTE[3],fusa);
  generate_tone(C_NOTE[0],fusa);
  generate_tone(Eb_NOTE[1],corchea);
  generate_tone(C_NOTE[2],fusa);
  generate_tone(Eb_NOTE[3],fusa);
  generate_tone(Ab_NOTE[0],semicorchea);
  generate_tone(G_NOTE[1],semicorchea);
  generate_tone(F_NOTE[2],semicorchea);
  generate_tone(F_NOTE[3],semicorchea);
  generate_tone(Eb_NOTE[0],semicorchea);
  delay_ms(100);
  return;
}
```

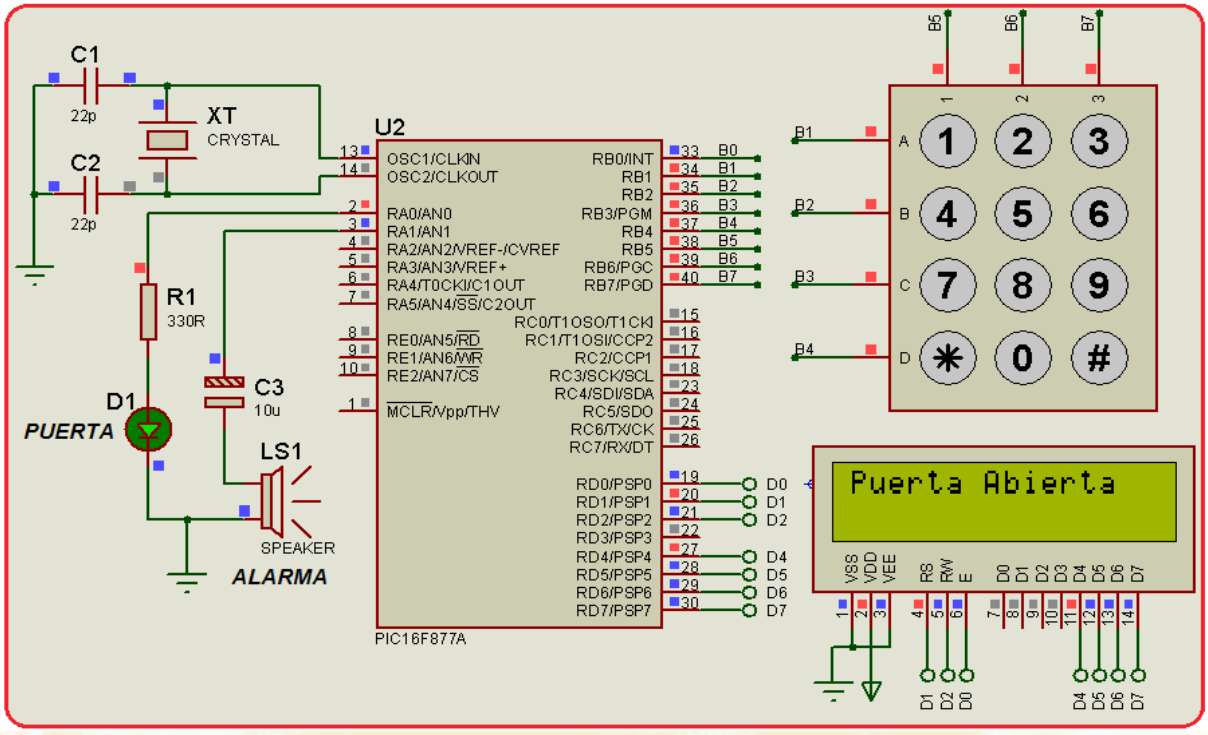


Figura 3.38. Clave de acceso con teclado y alarma.
Elaborado por: Los Autores.

EJERCICIOS ADICIONALES

Ejercicios resueltos

Cerradura digital.

En el siguiente ejercicio se acciona una salida cuando se ingresa una clave previamente almacenado en el programa.

En un array de 4 números se guarda los números de la *clave (int clave [] = {1,2,3,4}). Los datos provenientes del teclado se ingresan en el array datos[], en la función almacenar. Una vez que se tienen los 4 números, se llama a la función procesar para comprobar que esté de acuerdo con los de la clave. Si son correctos se llama a la función abrir* para activar el puerto b6 que acciona la puerta. El puerto b6 permanece en alto 5 segundos. En el caso contrario a la función *sirena activa la alarma. Por el puerto b.7 se generan pulsos que se pueden conectar a un timbre o un zumbador.*

Con fines de aprendizaje se ha incluido el display. Para aplicaciones prácticas se podría omitir el display, ya que no es necesario porque no se puede mostrar los números que ingresan. Por esta razón en el ejercicio,

cada vez que se tecleé un número el display se blanquea momentáneamente indicando que ha ingresado un dato. El display permanece mostrando el 0.

PROGRAMA: CERRADURA-TECLADO PROVISTO DE ANTIRREBOTE

```
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)     //F_OSC =12MHz
#BYTE port_b= 0xF81             //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83             //Identificador para el puerto d en la localidad 0xF83.
#BIT LED = 0xF81.5              //LED indicador.
#BIT puerta = 0xF81.6
 //Puerta
#BIT alarma = 0xF81.7           //Alarma

#BIT FA = 0xF83.0               //Asigna en la localidad 0xF83 bit 0 a la variable FA.
#BIT FB = 0xF83.1               //Asigna en la localidad 0xF83 bit 1 a la variable FB.
#BIT FC = 0xF83.2               //Asigna en la localidad 0xF83 bit 2 a la variable FC.
#BIT FD = 0xF83.3               //Asigna en la localidad 0xF83 bit 3 a la variable FD.
#define C1 bit_test(port_d,4)  //Define la variable C1 para el port d4.
#define C2 bit_test(port_d,5)  //Define la variable C1 para el port d5.
#define C3 bit_test(port_d,6)  //Define la variable C1 para el port d6.
#define C4 bit_test(port_d,7)  //Define la variable C1 para el port d7.
#define num 4                   //Define la variable num tipo array para 4 elementos.
void antirebote();              //Función anti rebote.
void almacenar();               //Función almacenar.
void procesar();                //Función procesar.
void sirena();                  //Función sirena.
void abrir();                   //Función abrir.
int i;                          //Para contar el número de veces que repite la alarma.
int x;                          //Toma el valor de la tecla.
const int clave[]={1,2,3,4};    //Almacena clave.
int n;                          // Para contar el número de datos que ingresan.
int datos[num];                 //Almacena el número de la tecla.
void main()                     //Función principal main.
{
set_tris_b(0x00);               //Puerto b como salida.
set_tris_d(0xf0);               //Puerto d, los 4 primeros bits como salida y lo 4 bits como entrada.
disable_interrupts(GLOBAL);     //Todas las interrupciones desactivadas.
port_b = 0;                     //Puerto b =0.
//Asigna a todas las variables FA, FB, FC y FD el valor de 1.
// Estas son las filas del teclado.
FA=1;
FB=1;
FC=1;
FD=1;
n= 1;                           //Asigna n = 1. Esta variable controla los elementos del array num

while(TRUE){                    //Bucle de barrido.
  FA=0;                         //La fila FA = 0.
```

```
 if (C1 == 0){                  //Censa si la tecla de la columna 1 ha sido accionada.
  x=1;                          //Asigna  x = 1. Valor de la primera tecla.
  antirebote();                 //Subrutina antirebote.
 }
 if (C2 == 0){                  //Censa si la tecla de la columna 2 ha sido accionada.
   x=2;                         //Asigna  x = 2. Valor de la segunda tecla.
  antirebote();                 //Subrutina antirebote.
}
 if (C3 == 0){                  //Censa si la tecla de la columna 3 ha sido accionada.
   x=3;                         //Asigna  x = 3. Valor de la tercera tecla.
  antirebote();                 //Subrutina antirebote.
}
 if (C4 == 0){                  //Censa si la tecla de la columna 4 ha sido accionada.
   x=10;                        //Asigna  x = 10, considerando el caso un teclado hexadecimal.
                                //Valor de la cuarta tecla.
   antirebote();                //Función anti rebote.
}
 FA=1;                          //La fila FA = 1.
//Comienza el censado de las teclas conectadas en la fila B, para los números, 4, 5, 6, 11(letra B).
 FB=0;                          //La fila FB= 0.
 if (C1 == 0){
   x=4;
  antirebote();
 }
 if (C2 == 0){
   x=5;
  antirebote();
 }

 if (C3 == 0){
   x=6;
  antirebote();
 }
  if (C4 == 0){
   x=11;
   antirebote();
 }
FB=1;                           //La fila FB = 1.
//Comienza el Censado de las teclas conectadas en la fila B, para los números 7, 8, 9, 12 (letra C).
 FC=0;                          //La fila FC= 0.
 if (C1 == 0){
   x=7;
   antirebote();
 }
 if (C2 == 0){
   x=8;
  antirebote();
 }

 if (C3 == 0){
```

```
    x=9;
    antirebote();
   }
 if (C4 == 0){
    x=12;
    antirebote();
   }
 FC=1;                          //La fila FC = 1.
 //Comienza el Censado de las teclas conectadas en la fila B, para los números 13, 0, 14, 15.
 FD=0;                          //La fila FD= 0.
  if (C1 == 0){
    x=13;
   antirebote();
   }
   if (C2 == 0){
    x=0;
   antirebote();
   }
  if (C3 == 0){
    x=14;
   antirebote();
  }
  if (C4 == 0){
    x=15;
    antirebote();
   }
 FD=1;

}                               //Fin del bucle infinito.

//FUNCIÓN ANTIRREBOTE
 void antirebote()
 {
  while(C1 == 0) { }      /          /No realiza nada hasta que el pulsador esté inactivo
  while(C2 == 0) { }
  while(C3 == 0) { }
  while(C4 == 0) { }
  delay_ms(30);                 //Retardo de 30 ms.
  port_b = 15;                  //Blanquea el display por un….
  delay_ms(200);                //Retardo de 200 ms.
  portb_b=0;                    //Puerto B = 0.
 almacenar();                   //Llamado a la subrutina almacenar.
 return;
  }
 //ALMACENAR DATOS DEL TECLADO
void almacenar()
 {
 if (n <= 4){
  datos[n]= x;
```

```
 n= n + 1;
 }
if(n==5){
 procesar();                    //Función procesar.
}
return;
}

//COMPARA LOS DATOS INGRESADOS CON LOS GUARDADOS
 void procesar()
 {
  n = 1;
  if (datos[1]== clave[0] && datos[2]== clave[1] && datos[3]== clave[2] &&
      datos[4]== clave[3])
     abrir();
  else
     sirena();
 return;
 }

 //ABRE LA PUERTA
 void abrir()
 {
  puerta=1;
  delay_ms(5000);
  puerta=0;
  return;
 }

 //ACTIVA LA ALARMA
 void sirena()
{
 for (i=1;i<=10; ++i)
 {
  alarma=1;
  delay_ms(500);
  alarma=0;
  delay_ms(500);
  }
 return;
}
```

Contador MOD – 100, con dos displays manejados por puertos B y D.

En un array se almacena los datos codificados para el display de 7 segmentos tipo ánodo común.

const int x[]= {64,121,36,48,25,18,2,120,0,16};

Mediante un bucle anidado se apunta al dato del array para indicarlo en cada display “i” para las unidades y “j” para las decenas.

```
for (j=0; j<=9;++j){
 for (i=0; i<=9;++i)
```

PROGRAMA:

```
#include <18f4550.h>              //Librería para usar el PIC18F4550.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)       //FOSC =12MHz
#BYTE port_d= 0xF83               //Identificador del Puerto D.
#BYTE port_b= 0xF81               //Identificador del Puerto B.
int i;                            //Variable i para identificar decenas.
int j;                            //Variable j para identificar decenas.
int n;                            //Controla bucle de tiempo.
const int x[]= {64,121,36,48,25,18,2,120,0,16};

void display(void)                //Función para desplegar resultados.
{
                                  //Bucle para mostrar datos en displays y cambiar datos cada 1 s.
  for (n=1; n<=25;++n) {
     port_b= x[i];                //Muestra datos en el display unidades por……
     delay_ms(20);                //…… 20 ms.
     port_d= x[j];                //Muestra datos en display decenas por….
     delay_ms(20);                //….20 ms
    }
  return;
}

void main(void)                   //Función principal.
{
set_tris_b(0x00);                 //Puerto B como salida.
set_tris_d(0x00);                 //Puerto D como salida.
disable_interrupts(GLOBAL);       //Todas las interrupciones desactivadas.
 port_d = 64;                     //Puerto D= 64, equivalente a 0.

 while(TRUE){                     //Bucle infinito.
  for (j=0; j<=9;++j){
     for (i=0; i<=9;++i)
     {
      display();                  //Llamado a la función display.
     }
  }
 }                                //Fin de bucle infinito
}                                 //Fin del main.
```

En la función “display” se muestra los datos. Además en la misma mediante un lazo se genera un tiempo de 1s, para cambiar los datos.

40ms*25= 1s.

```
for (n=1; n<=25;++n) {
    port_b= x[i];
    delay_ms(20);
    port_d= x[j];
    delay_ms(20);
   }
```

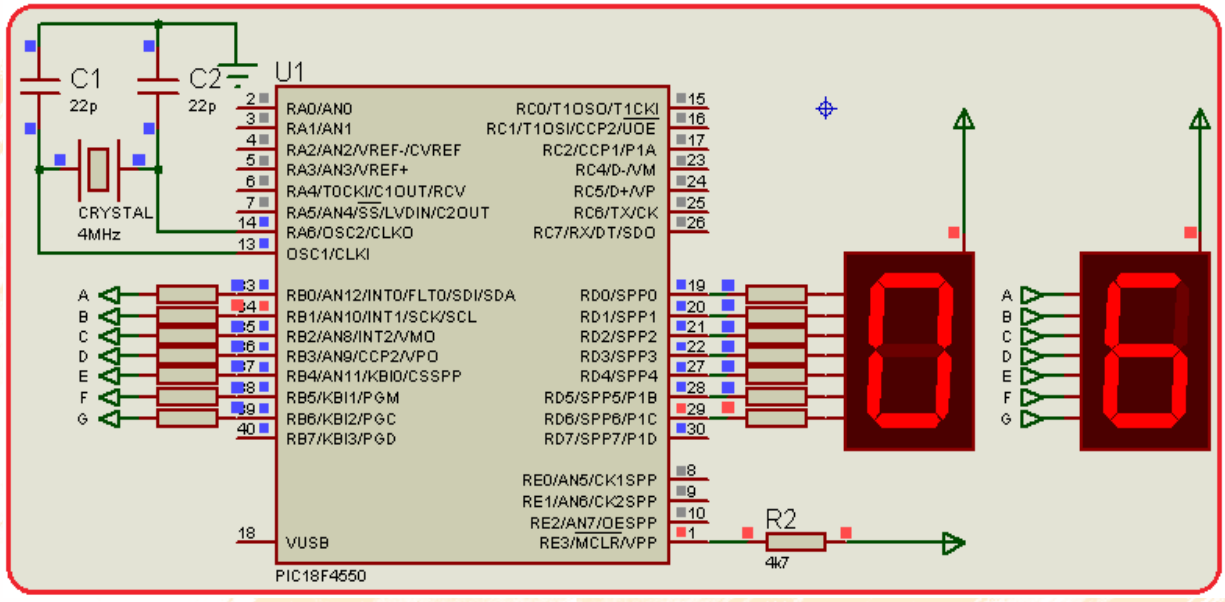


Figura 3.39. Contador MOD-100.
Elaborado por: Los Autores.

Visualizar en un GLCD el estado de las entradas del puerto A.

En el programa siguiente se va a visualizar el estado del puerto A mediante un GLCD. Cuando un pin del puerto A está activo (1L) un cuadro se muestra en blanco, por el contrario cuando el pin está desactivado (0L) el cuadro se pinta. La figura 3.40, muestra las conexiones y el detalle de lo explicado. Se observa que los puertos A0, A2, A3 y A5, están en 0 y los puertos A1, A4 y A6.

Para utilizar todo el puerto A, se configura el uso oscilador interno mediante el fuse *INTRC_IO*. Con esto, además se habilita el pin RA6 como entrada.

PROGRAMA:

```
 #include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses INTRC_IO,NOWDT,NOPROTECT,NOPUT,NOLVP //Configuración de fusibles.
#use delay (clock=4000000)       //F_OSC =4MHz.
#include <HDM64GS12.c>           //Librería para manejo del GLCD.
#include <graphics.c>            //Librería gráfica con funciones del GLCD.
#use standard_io(a)              //Función estándar del Puerto A.
#use standard_io(d)              //Función estándar del Puerto D.
void main() {
   CHAR A6[ ]="A6";              //Variable char “A6”.
   CHAR A5[ ]="A5";              //Variable char “A5”.
   CHAR A4[ ]="A4";              //Variable char “A4”.
   CHAR A3[ ]="A3";              //Variable char “A3”.
   CHAR A2[ ]="A2";              //Variable char “A2”.
   CHAR A1[ ]="A1";              //Variable char “A1”.
   CHAR A0[ ]="A0";              //Variable char “A0”.
   CHAR MEN [ ]= "PUERTO A";     //Mensaje PUERTO A".
   glcd_init(ON);                //Inicializa el GLCD.
                                 //Escribe los caracteres en la posiciones indicadas.
   glcd_text57(12, 30,A6, 1, 1);
   glcd_text57(28, 30,A5, 1, 1);
   glcd_text57(44, 30,A4, 1, 1);
   glcd_text57(60, 30,A3, 1, 1);
   glcd_text57(76, 30,A2, 1, 1);
   glcd_text57(92, 30,A1, 1, 1);
   glcd_text57(108, 30,A0, 1, 1);
   glcd_text57(10,10,MEN, 2, 1);  //Mensaje “PUERTO A”
while(TRUE){                     //Bucle infinito.
        //Dibuja rectángulo en la posición indicada y de acuerdos a las medidas dadas.
        //Si el puerto es 0 rellena caso contrario en blanco.
   if (input_state(PIN_A6)==0) glcd_rect(11,40,25,60,1,1);
   else glcd_rect(11,40,25,60,1,0);glcd_rect(11,40,25,60,0,1);
   if (input_state(PIN_A5)==0) glcd_rect(27,40,41,60,1,1);
   else glcd_rect(27,40,41,60,1,0);glcd_rect(27,40,41,60,0,1);
   if (input_state(PIN_A4)==0)glcd_rect(43,40,57,60,1,1);
   else glcd_rect(48,40,57,60,1,0);glcd_rect(43,40,57,60,0,1);
   if (input_state(PIN_A3)==0) glcd_rect(59,40,73,60,1,1);
   else glcd_rect(59,40,73,60,1,0);glcd_rect(59,40,73,60,0,1);
   if (input_state(PIN_A2)==0) glcd_rect(75,40,89,60,1,1);
   else glcd_rect(75,40,89,60,1,0); glcd_rect(75,40,89,60,0,1);
   if (input_state(PIN_A1)==0) glcd_rect(91,40,105,60,1,1);
   else glcd_rect(95,40,105,60,1,0);glcd_rect(91,40,105,60,0,1);
   if (input_state(PIN_A0)==0)  glcd_rect(107,40,121,60,1,1);
   else glcd_rect(107,40,121,60,1,0);glcd_rect(107,40,121,60,0,1);
  delay_ms(200);
   }                             //Fin del bucle infinito.
}                                //Fin del main.
```

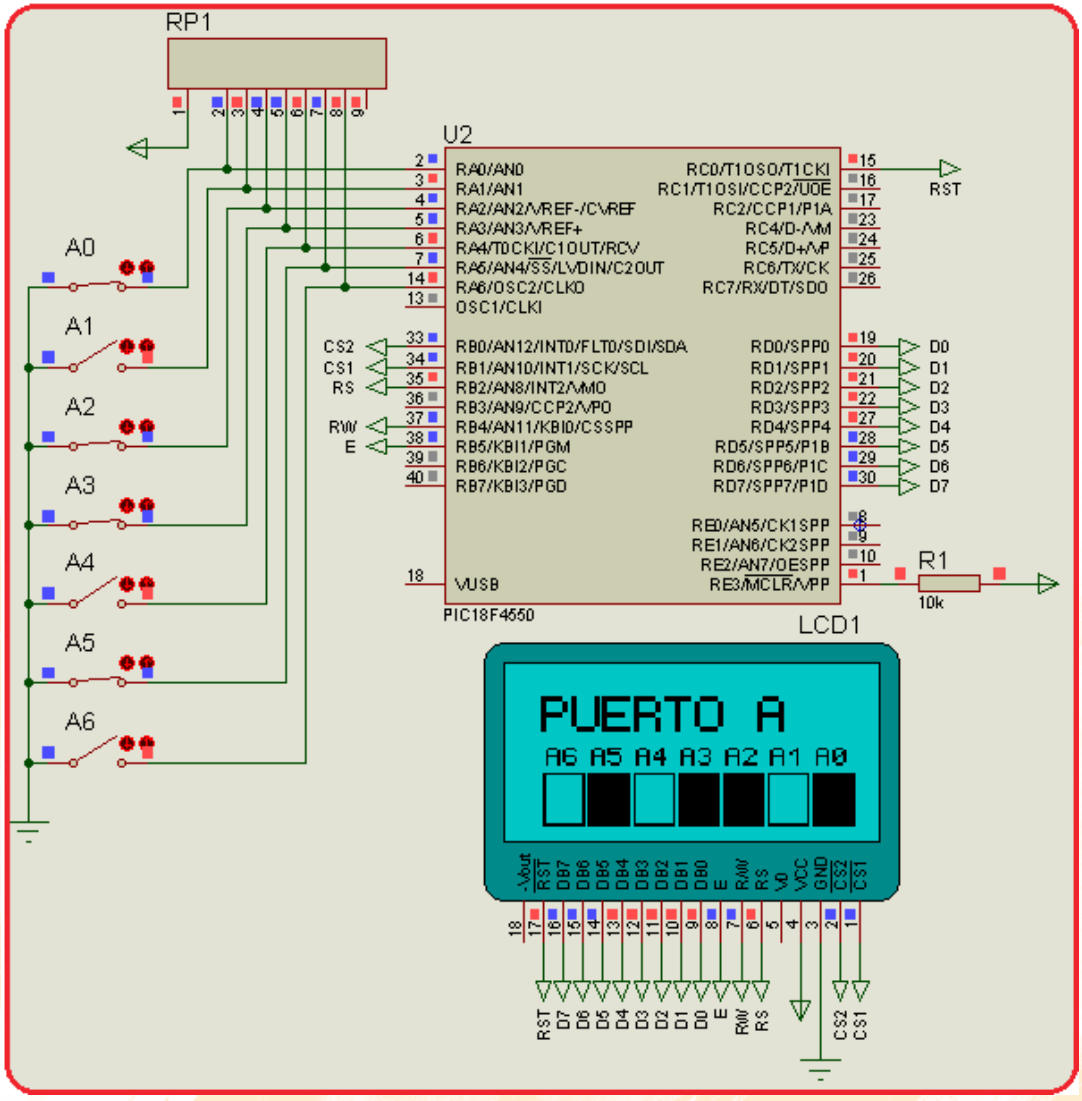


Figura 3.40. Visualización del estado del puerto A en el GLCD.
Elaborado por: Los Autores.

Manejo de LCD con PIC16F628A.

Uno de los microcontroladores más populares entre los estudiantes es el PIC16F628A, por lo que incluiremos un ejercicio para manejar un LCD, mediante este Micro.

Algunas consideraciones para utilizar el microcontrolador 16F628A:

Utilizar el oscilador interno de 4 MHZ: #fuses INTRC y #use delay (clock=4000000).

Se utiliza el puerto B para conectar el LCD. Por tanto se debe definir que se utilizará el puerto B en el programa: #define use_portb_lcd TRUE

Realizar un contador de 0 – 9.

PROGRAMA:

```
#include <16f628.h>                    //Librería para usar el PIC16f628.
#fuses INTRC, NOMCLR, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=4000000)             //F_OSC =4MHz.
#include <lcd.c>                       //Librería lcd.c.
#define use_portb_lcd TRUE             //Define uso del puerto B para manejar el LCD.
int i;          //Variable i.

void main(void)                        //Función principal main
{
lcd_init();                            //Inicializa el LCD.
while (TRUE){                          //Inicio del bucle.
  lcd_gotoxy(5,1);                     //Sitúa el cursor en 5 columna, 1 fila.
  lcd_putc("CONTADOR");                //Escribe CONTADOR.
 for (i=0; i<=9;++i)                   //La variable empieza desde 0 hasta 9.
    {
     lcd_gotoxy(8,2);                  //Sitúa el cursor en 7 columna, 2  fila.

       printf(lcd_putc," %d ",i);      //Escribe en el LCD el valor actual de i.
       delay_ms(500);                  //Retardo de 500 ms.
    }
 }                                     //Fin del bucle infinito.
 }                                     //Fin del main.
```

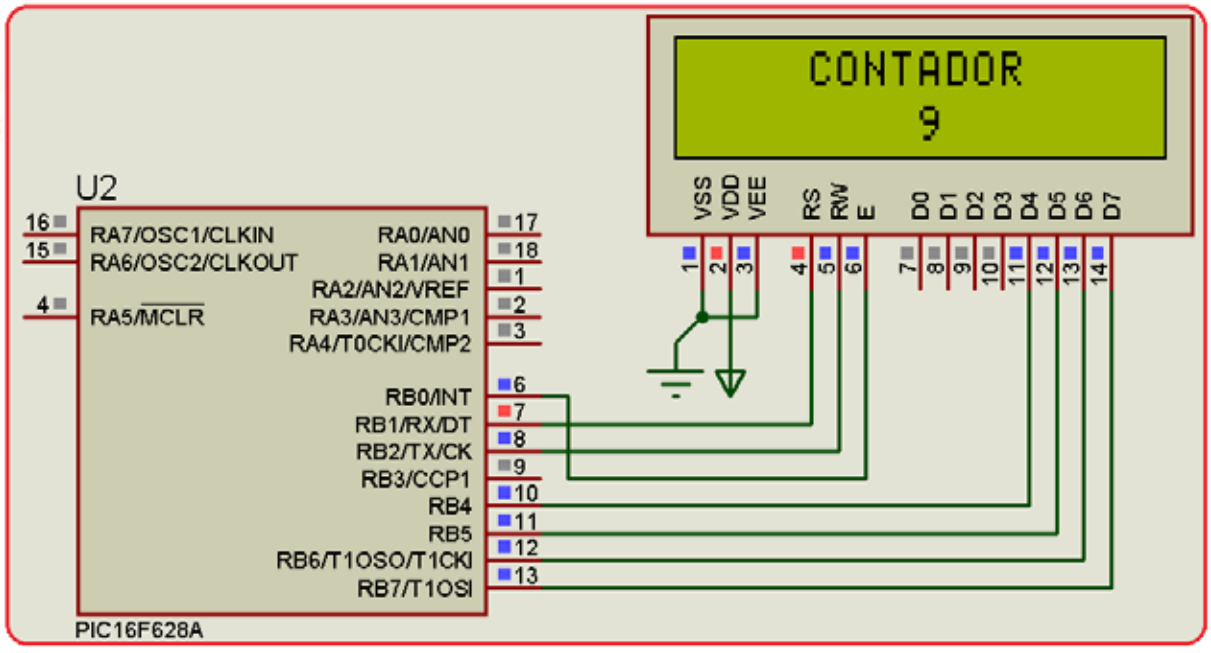


Figura 3.41. Manejo de LCD con Microcontrolador 16F628A.
Elaborado por: Los Autores.

Realice un programa para controlar dos display directamente conectados al puerto D, como se indica en la figura 3.42. Programar para un contador MOD-100.

Realice un contador módulo 24, utilizando el esquema de la figura 3.42.

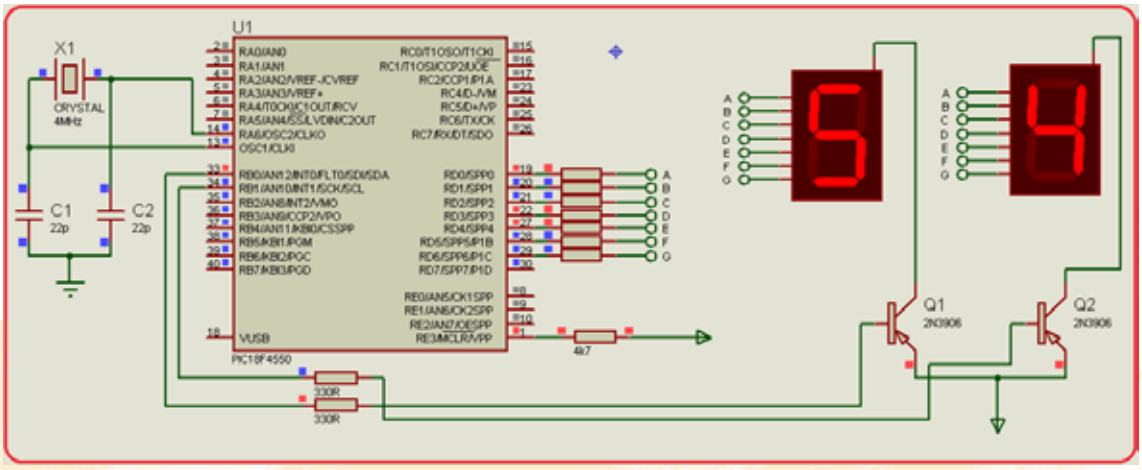


Figura 3.42. Diagrama para los ejercicios 3.5 y 3.6.
Elaborado por: Los Autores.

Elabore el programa para un contador MOD-100. El display de unidades está conectado con un decodificador 7447 a las 4 primeras líneas del puerto B y el display de decenas en las cuatro siguientes líneas como indica en la figura 3.43.

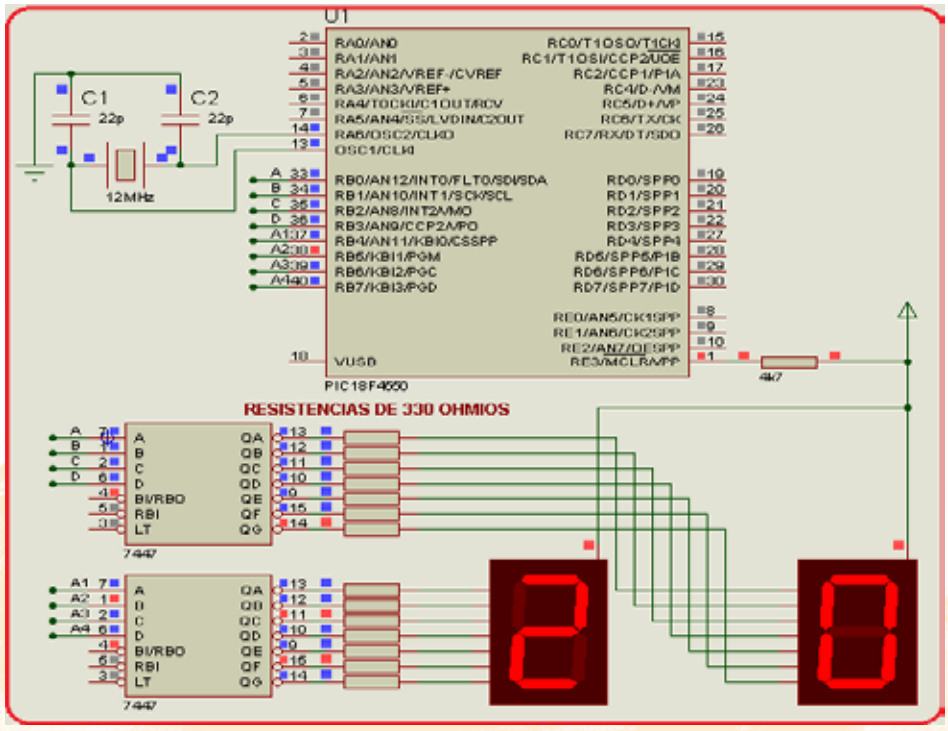


Figura 3.43. Ejercicio 3.7.
Elaborado por: Los Autores.

Utilizando un módulo LCD conectado en el puerto B, diseñe el circuito y el programa para desplazar un texto de derecha a izquierda con el mensaje "Programación PIC-Microcontroladores".

Utilizando un módulo LCD conectado en el puerto B, diseñe el circuito y el programa para escribir en el LCD la palabra "Programación PIC-CCS". Las letras irán apareciendo una por una como si estuviéramos escribiendo. En la figura 3.44, se muestra cómo va apareciendo las letras.

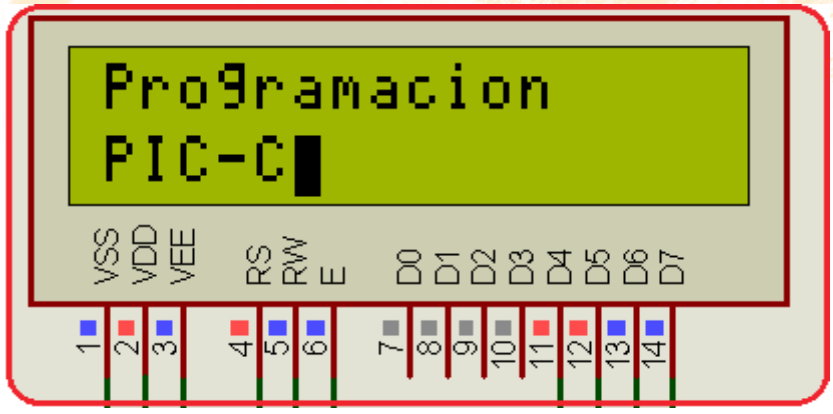
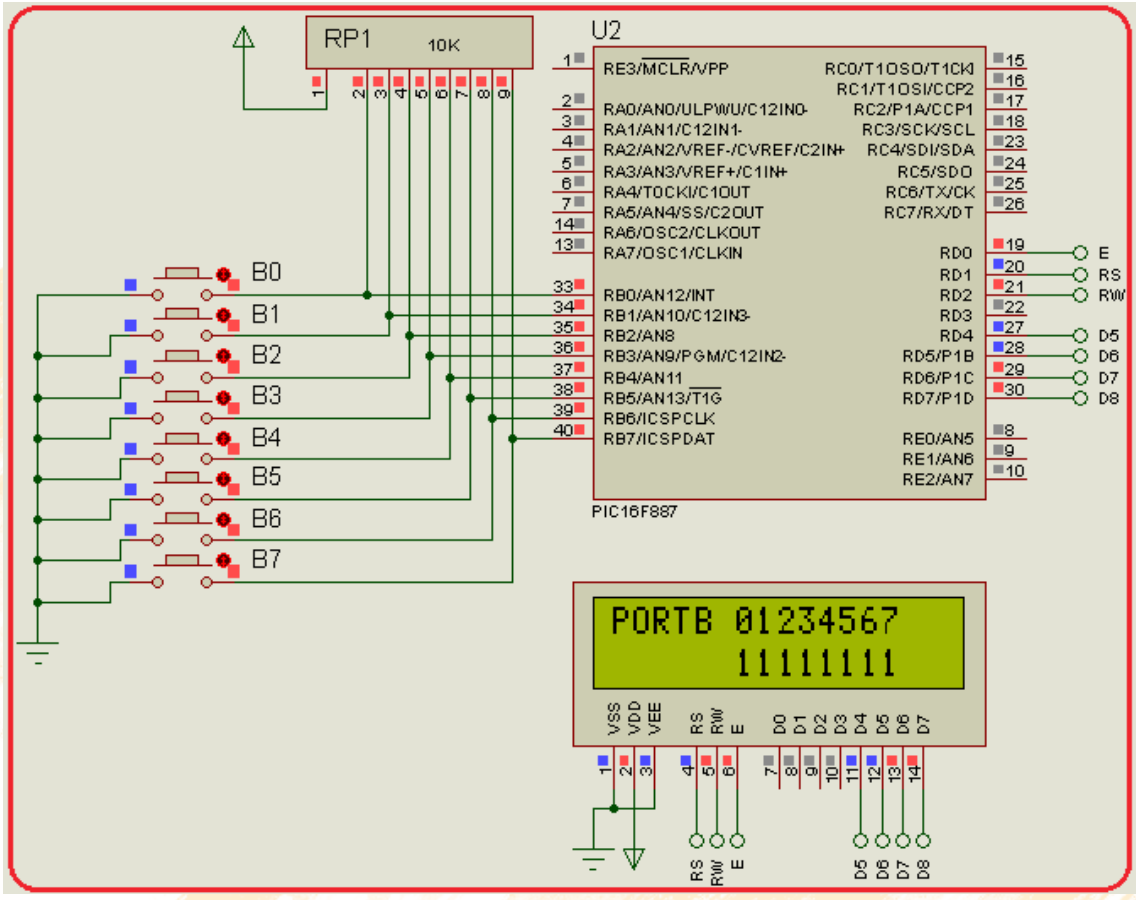
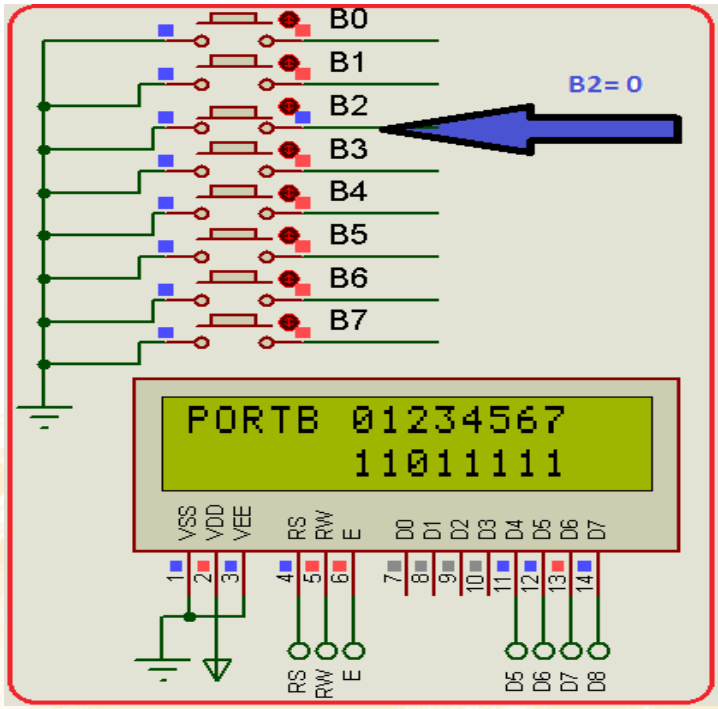


Figura 3.44. Escritura de letras de izquierda a derecha.
Elaborado por: Los Autores.

De acuerdo al gráfico de la figura 3.45, todo el puerto B está conectado a alto, como se observa en el LCD. Al cerrar cualquier pulsador el puerto recibe un 0, por lo tanto en el LCD debe cambiar a 0. Realice el programa respectivo para que cumpla con lo indicado. La figura 3.45, a) detalla las conexiones a utilizarse para el ejercicio y b) el resultado esperado.



a). Conexiones.



b). Resultado esperado.

Figura 3.45. Ejercicio 3.10.
Elaborado por: Los Autores.

Almacenar una clave de 5 dígitos en la memoria EEPROM interna. Mediante un teclado 4x3 y el driver KBD.c, ingresar los datos. En un LCD se debe indicar un * por cada número indicado. Si la clave es correcta accionar una puerta. En el LCD indicar puerta abierta. Si la clave es incorrecta indicar que tiene dos oportunidades más. Al finalizar la tercera oportunidad, y la clave es incorrecta, activar una alarma que solo podrá ser desactivada mediante un pulsador escondido. La alarma debe generar una señal sonora similar a una sirena policial.

Realice un programa para mostrar en un GLCD, 8 círculos concéntricos todo la pantalla GLCD, como se ve la figura 3.46.

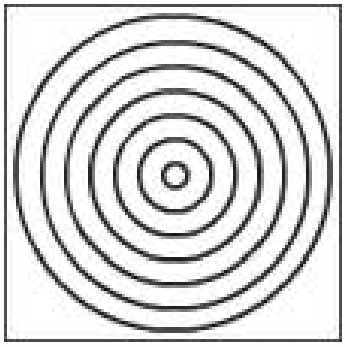
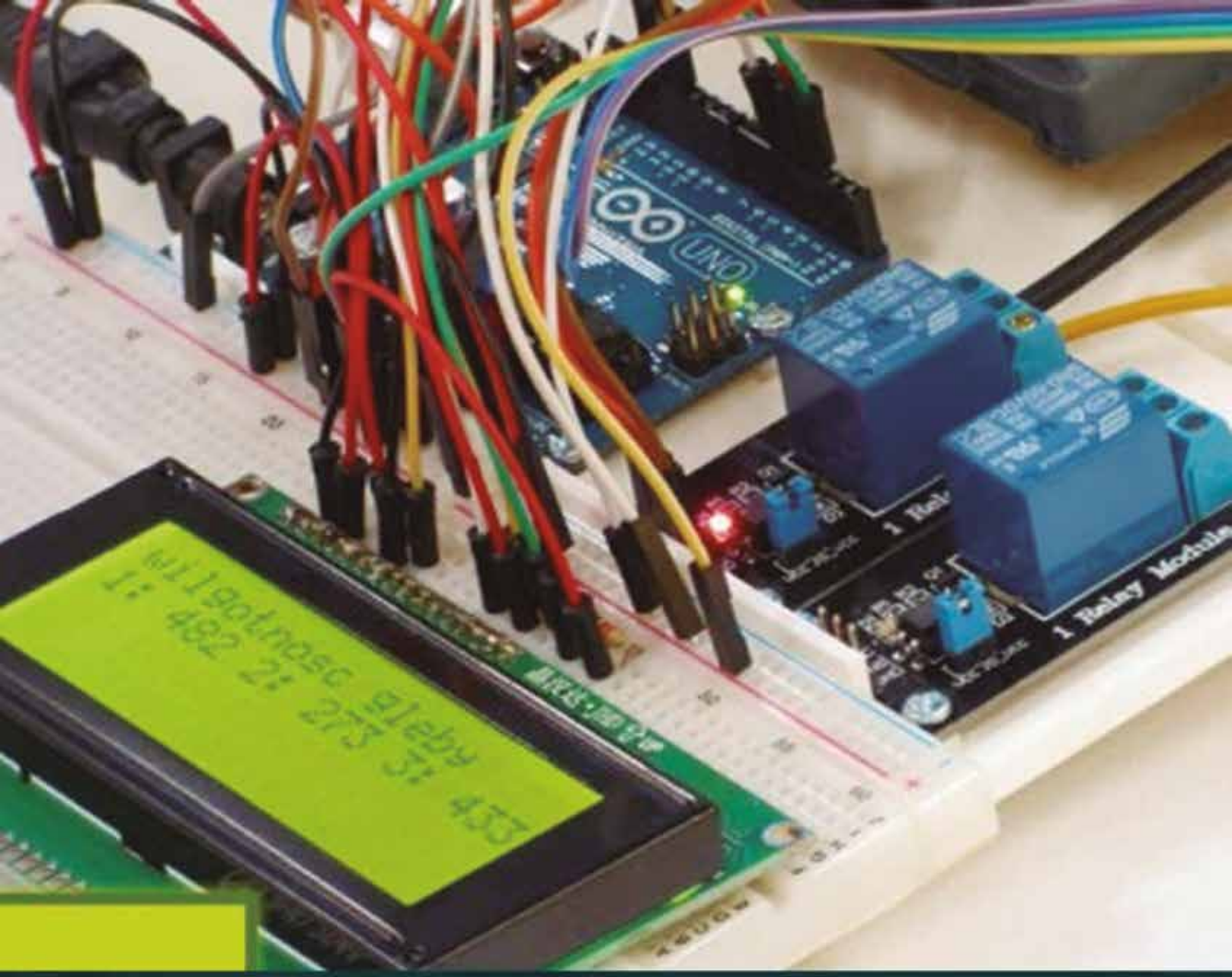


Figura 3.46. Círculos concéntricos generados en un GLCD.
Elaborado por: Los Autores.

Muestre en una pantalla GLCD, el logo de la Universidad y en la parte inferior el nombre.

Visualice en un GLCD el estado del puerto B. Los interruptores se conectan en el puerto B (active las resistencias PULL-UP) y el GLCD en el puerto D. Envié los mensajes necesarios al GLCD. El estado activo se muestra con un círculo en blanco y el estado apagado con un círculo pintado.



CAPÍTULO 4

INTERRUPCIONES EXTERNAS

Introducción

Una interrupción se define como un pedido de alta prioridad que un dispositivo exterior o un evento de programación solicita a la CPU para ejecutar otro programa.

El microcontrolador acepta dos tipos de interrupciones: las interrupciones por periféricos y las externas.

El uC 18F4550 posee 21 fuentes de interrupciones. Se distinguen dos grupos de interrupciones:

Grupo general de interrupciones externas:

- Interrupción del Temporizador 0.
- Interrupción por cambio de estado en PORTB: Puertos B.4, B.5, B.6, B.7. En este caso cualquier cambio produce la misma interrupción.
- Interrupción externa 0: Puerto B.0
- Interrupción externa 1: Puerto B.1
- Interrupción externa 2: Puerto B.2

Grupo de interrupciones de periféricos:

- Interrupción del SPP
- Interrupción del A/D
- Interrupción de recepción de la EUSART
- Interrupción de transmisión de la EUSART
- Interrupción del MSSP
- Interrupción del CCP1
- Interrupción del Temporizador 2
- Interrupción del Temporizador 1
- Interrupción de fallo del oscilador
- Interrupción del comparador
- Interrupción del USB
- Interrupción de escritura en FLASH/EEPROM
- Interrupción de colisión del Bus (MSSP)
- Interrupción de detección de anomalías en VDD
- Interrupción del Temporizador 3
- Interrupción del CCP2

El grupo de registros que controlan las interrupciones son:

- INCONT

- INCONT2
- INCONT3
- RCON
- RCON
- PIR1, PIR2
- PIE1, PIE2
- IPR1, IPR2

Directivas de control

El compilador CCS posee una serie de directivas que tienen diferentes funciones para manejar las interrupciones de forma transparente. Las directivas utilizadas para las interrupciones del 18F4550 se muestran en la tabla 4.1.

DIRECTIVA	DESCRIPCIÓN
INT_EXT	Interrupción externa por RB0
INT_EXT1	Interrupción externa por RB1
INT_EXT2	Interrupción externa por RB2
INT_RB	Interrupción externa por el cambio de estado de los bit RB4:RB7 del puerto B
INT_RTCC	Interrupción por desbordamiento del TIMER0
INT_PSP	Interrupción del canal de transmisión de datos paralelo
INT_AD	Interrupción por fin de conversión del módulo A/D
INT_RDA	Interrupción por recepción EUSART
INT_TBE	Interrupción por trasmisión EUSART
INT_SSP	Interrupción por actividad en el módulo SPI o I2C
INT_CCP1	Interrupción por módulo 1 de captura, comparación y PWM
INT_CCP2	Interrupción por módulo 2 de captura, comparación y PWM
INT_TIMER1	Interrupción por el temporizador 1
INT_TIMER2	Interrupción por el temporizador 2
INT_TIMER3	Interrupción por el temporizador 3
INT_OSCF	Interrupción por fallo en el oscilador
INT_COMP	Interrupción por el comparador
INT_USB	Interrupción por actividad en el puerto USB
INT_EEPROM	Interrupción por fin de escritura en la EEPROM
INT_BUSCOL	Interrupción por colisión en el bus SPI
INT_LVD	Interrupción por bajo voltaje

Tabla 4.1. Directivas usadas para el manejo de interrupciones.
Fuente: Rangel, 2012. Internet.

Estas mismas directivas son utilizadas en las funciones para deshabilitar y habilitar las interrupciones, según el formato:

ENABLE_INTERRUPTS(Directiva);
DISABLE_INTERRUPTS(Directiva);

Ejemplo:
ENABLE_INTERRUPTS(INT_RB); Habilita la interrupción RB (RB4:RB7).
DISABLE_INTERRUPTS(INT_EXT); Deshabilita la interrupción Externa 0.

Prioridad de las interrupciones

En los microcontroladores de gama alta el sistema de interrupciones posee dos niveles de prioridad, alta y baja. La diferencia entre una prioridad y otra, es que las interrupciones de alta prioridad pueden interrumpir una de baja prioridad.

Cuando una interrupción de alta prioridad es ejecutada se ejecuta a partir de la dirección 0008H en cambio el nivel bajo en la dirección 0018H.

Todas las interrupciones pueden ser programadas en cualquiera de las dos prioridades, excepto la interrupción externa por RB0, está siempre tendrá alta prioridad. Se puede forjar a modo de compatibilidad en alta prioridad mediante el bit IPEN= 0 del registro RCON.

Para trabajar de modo transparente el compilador CCS permite incluir la directiva

#DEVICE HIGH_INTS=TRUE

Con lo que permite incluir las palabras HIGH y FAST lo cual altera la prioridad de la interrupción. Así en los PIC18 se pueden modificar las directivas y tener las siguientes interrupciones:

- ***#INT_nnn:*** Prioridad normal de interrupción, la más baja. El compilador salva y restaura los registros claves y no detiene una interrupción en progreso.
- ***#INT_nnn FAST:*** Prioridad alta de interrupción. El compilador no salva ni restaura los registros claves. Esta interrupción detiene a una interrupción en progreso. Solo el programa permite una interrupción de prioridad FAST.
- ***#INT_nnn HIGH:*** Interrupción de alta prioridad. El compilador salva y restaura los registros claves. Esta interrupción detiene a una interrupción en progreso.

Ejemplo:

```
#include <18f4550.h>
#DEVICE HIGH_INTS=TRUE
#int_rda HIGH
#int_ int_ssp FAST
#int_ad
```

La directiva ***#DEVICE HIGH_INTS=TRUE,*** permite modificar el orden de prioridad de las interrupciones. Se debe colocar a nivel de la librería del dispositivo que se está utilizando. La interrupción **int_ad,** tiene la prioridad más baja, seguido de la interrupción **int_ssp** y la más alta prioridad tiene la interrupción **int_rda.**

También se puede definir la prioridad de las interrupciones utilizando **#priority *ints***

Donde ***ints*** es una lista de uno o más interrupciones separadas por comas.

Ejemplo:

#priority int_rda, int_ int_ssp, int_ad

Las prioridades son como en el ejemplo anterior.

Cuando existen dos o más interrupciones definidas en software con el mismo nivel de prioridad, se ejecutará aquella que a nivel de hardware del microcontrolador tenga mayor prioridad.

Tipos de interrupciones externas

El PIC 18F4550 provee de 3 interrupciones externas INT0, INT1 y INT2. Es muy común que la interrupción externa en los microcontroladores de gama media se refiera al puerto RB0 y por compatibilidad Microchip mantiene para sus micros de gama alta, adicionando la interrupción externa 1 y la interrupción externa 2.

Las directivas y funciones de CCS que permiten controlar las interrupciones externas son:

 # INT_XXX; esta directiva indica cual interrupción se va utilizar. Donde, *XXX* indica el nombre de la interrupción, así:

- ***INT_EXT***; interrupción externa 0.

- ***INT_EXT1***; interrupción externa 1.
- ***INT_EXT2***; interrupción externa 2.

ext_int_edge (source, edge); controla la interrupción y el flanco de la interrupción externa. Dónde:

source, especifica el número de la interrupción 0 (RB0), 1 (RB1), 2 (RB2). Se puede omitir en este caso por default toma la interrupción 0.

edge, indica el flanco en el cual se dispara la interrupción.

- ***(L_TO_H);*** *flanco de subida.*
- ***(H_TO_L);*** *flanco de bajada.*

Ejemplos:

- ***ext_int_edge(H_TO_L);*** la interrupción externa se detectará en pin RB0 en flanco de bajada.
- ***ext_int_edge(1, L_TO_H);*** la interrupción externa se detectará en pin RB1 en flanco de bajada.
- ***ext_int_edge(2, H_TO_L);*** la interrupción externa se detectará en pin RB2 en flanco de bajada.

enable_interrupts(GLOBAL); habilitación global de todas las interrupciones.

enable_interrupts(INT_XXX); habilita de la interrupción externa referida en *XXX*.

Nota: El compilador CCS acepta las mayúsculas o minúsculas en el argumento de las funciones.

Las instrucciones de habilitación, deben estar activadas en la función principal para que pueda detectar las interrupciones externas.

Interrupción externa 0

Esta interrupción utiliza las directivas y funciones indicadas para las interrupciones externas. Es la interrupción de más alta prioridad y no puede ser modificada su prioridad. La interrupción es detectada por el cambio de estado en el pin RB0, dependiendo del flanco que se haya fijado. La figura 4.1, muestra el diagrama de conexiones para utilizar la interrupción por RB0.

Ejercicio 4.1. Interrupción externa RB0.

En el siguiente ejercicio, se utiliza la interrupción externa INT_EXT. El programa inicia prendiendo y apagando un LED1 que está conectado al puerto RD0. Cuando existe un cambio de estado en el puerto RB0, es decir, cuando se accione el pulsador, se genera una interrupción, en la cual, un LED2 que está conectado en el puerto RD0 se prende y se apaga cinco veces.

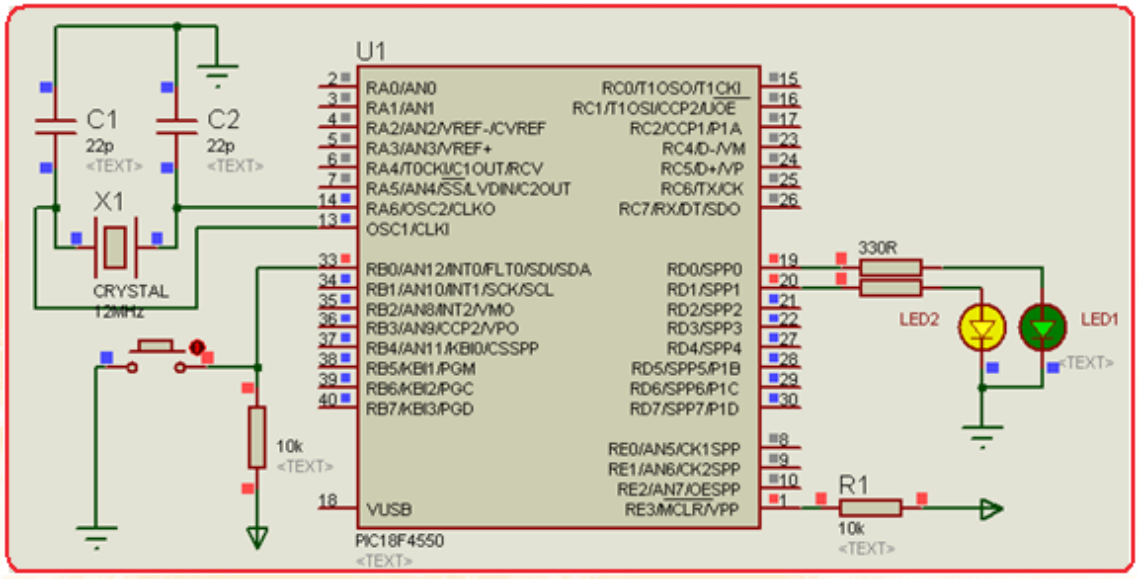


Figura 4.1. Circuito para comprobar la interrupción externa RB0.
Elaborado por: Los Autores.

PROGRAMA:

```
//USO DE LA INTERRUPCIÓN EXTERNA RBO
#include <18f4550.h>          //Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOWRT, NOPUT, NOWDT //Configuración de fusibles.
#use delay (clock=12000000) /  /F_OSC = 12MHz.
#BYTE port_b= 0xF81           //Identificador para el puerto b en la localidad 0xF81.
#BYTE port_d= 0xF83           //Identificador para el puerto d en la localidad 0xF83.
int n =0;                     //Variable para el contador.

//Función para la interrupción.
#int_ext                      //Activa la  interrupción externa RB0.
PortB_Interrupt()             //Nombre de la interrupción.
{
  ext_int_edge(H_TO_L);       //Detecta el cambio de estado de RB0 en el flanco negativo.
```

```
 for(n=1; n<=5; n++){          //El LED conectado en el puerto D1 titila 5 veces.
 output_high(PIN_D1);
 delay_ms(300);
 output_low(PIN_D1);
  delay_ms(300);
 }
 }

void main(void)                //Función principal main.
{
  set_tris_b(0b00000001);      //Fija el puerto RB1 como entrada y el resto como salida.
  set_tris_d(0b00000000);      //Fija el puerto D como salida.
  enable_interrupts(GLOBAL); //Habilitación de todas(GLOBAL) las interrupciones.
  enable_interrupts(INT_EXT); //Habilita la interrupción externa RB0.

 while(TRUE){                  //El LED del PIN D0 titila indefinidamente.
  output_high(PIN_D0);
  delay_ms(50
0);
  output_low(PIN_D0);
  delay_ms(500);
 }                             //Fin del bucle infinito.
}                              //Fin del main.
```

Interrupción externa 1: INT_EXT1

Esta interrupción utiliza las mismas directivas y funciones de la interrupción externa INT_EXT. Se debe incluir en la directiva o habilitación de la función el número 1 para referirse a la interrupción externa 1. La interrupción es detectada por el cambio de estado en el pin RB1, dependiendo del flanco que se haya fijado. La figura 4.2, muestra el diagrama de conexiones para utilizar la interrupción por RB1.

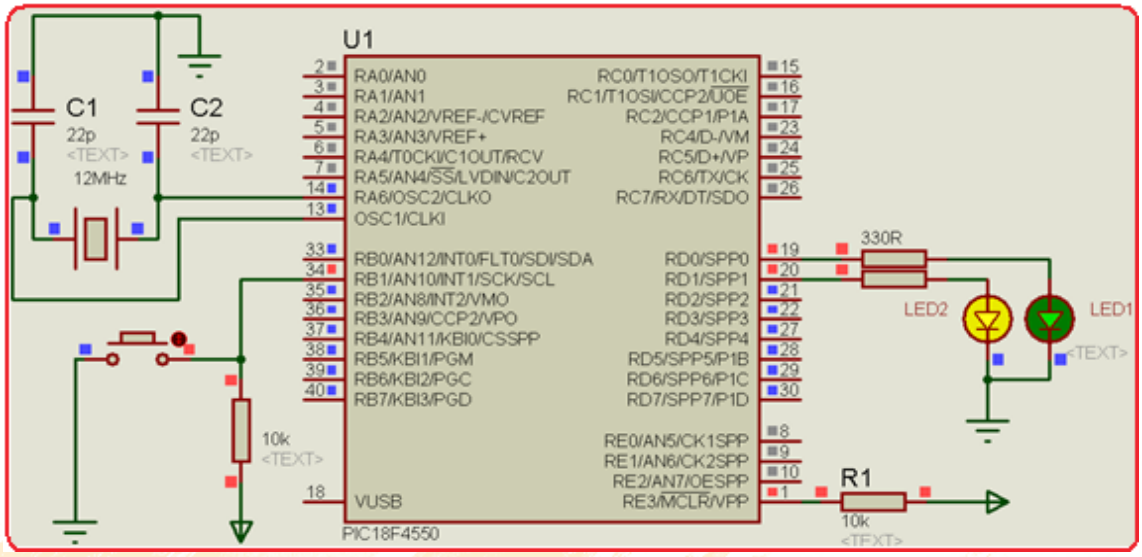


Figura 4.2. Circuito para comprobar la interrupción externa 1.
Elaborado por: Los Autores.

Ejercicio 4.2. Interrupción externa 1.

En el programa principal el LED1 conectado en RD0 titila indefinidamente. Al ocurrir el cambio de estado (accionar el pulsador) en flanco de bajada un LED2 conectado en RD1 titila 5 veces.

PROGRAMA:

```
//USO DE LA INTERRUPCIÓN EXTERNA RB1
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOWRT, NOPUT, NOWDT //Configuración de fusibles.
#use delay (clock=12000000) /  /F_OSC = 12MHz.
#BYTE port_b= 0xF81             //Identificador para el puerto b en la localidad 0xF81
#BYTE port_d= 0xF83             //Identificador para el puerto d en la localidad 0xF83.
int n =0;                       //Variable para el contador.

//Función para la interrupción.
#int_ext 1                      //Activa la  interrupción externa RB1.
PortB_Interrupt()               //Nombre de la interrupción.
{
  ext_int_edge(1,H_TO_L);       //Detecta el cambio de estado de RB1 en el flanco
// negativo.
  for(n=1; n<=5; n++){          //El LED conectado en el puerto D1 titila 5 veces.
  output_high(PIN_D1);
  delay_ms(300);
  output_low(PIN_D1);
   delay_ms(300);
  }
 }
```

```
void main(void)                    //Función principal main.
{
  set_tris_b(0b00000010);          //Fija el puerto RB1 como entrada y el resto como
// salida.
  set_tris_d(0b00000000);          //Fija el puerto D como salida.
  enable_interrupts(GLOBAL); //Habilitación de todas (GLOBAL) las interrupciones.
  enable_interrupts(INT_EXT1);     //Habilita la interrupción externa RB1.

 while(TRUE){                      //El LED del PIN D0 titila indefinidamente.
  output_high(PIN_D0);
  delay_ms(500);
  output_low(PIN_D0);
  delay_ms(500);
  }                                 //Fin del bucle infinito.
}                                   //Fin del main.
```

Interrupción externa 2: INT_EXT2

Esta interrupción utiliza las mismas directivas y funciones de la interrupción externa INT_EXT. Se debe incluir en la directiva o habilitación de la función el número 2 para referirse a la interrupción externa 2. La interrupción es detectada por el cambio de estado en el pin RB2, dependiendo del flanco que se haya fijado. La figura 4.3, muestra el diagrama de conexiones para utilizar la interrupción por RB2.

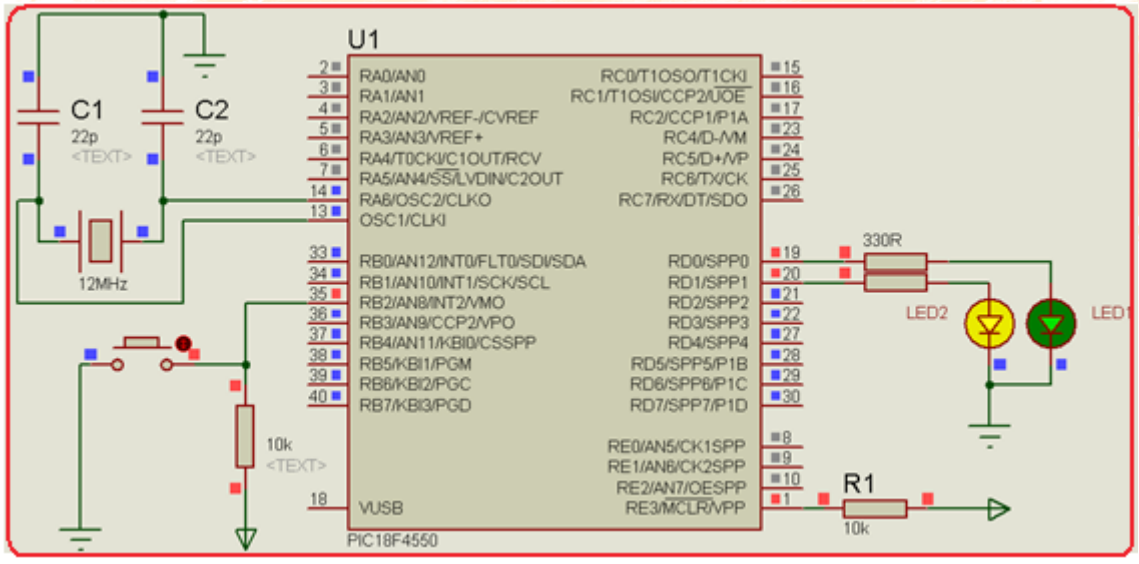


Figura 4.3 Circuito para comprobar la interrupción externa 2.
Elaborado por: Los Autores.

Ejercicio 4.3. Interrupción externa 2.

En el programa principal el LED1 conectado en RD0 parpadea indefinidamente. Al ocurrir el cambio de estado (accionar el pulsador) en flanco de bajada un LED2 conectado en RD1 parpadea 5 veces.

PROGRAMA:

```
//USO DE LA INTERRUPCIÓN EXTERNA RB2
#include <18f4550.h>            //Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOWRT, NOPUT, NOWDT //Configuración de fusibles.
#use delay (clock=12000000) /  /F_OSC = 12MHz.
#BYTE port_b= 0xF81             //Identificador para el puerto b en la localidad 0xF81
#BYTE port_d= 0xF83             //Identificador para el puerto d en la localidad 0xF83.
int n =0;                       //Variable para el contador.


//Función para la interrupción.
#int_ext2                       //Activa la  interrupción externa RB2.
PortB_Interrupt()               //Nombre de la interrupción.
{
 ext_int_edge(1,H_TO_L);              //Detecta el cambio de estado de RB1 en el flanco negativo.
 for(n=1; n<=5; n++){                 //El LED conectado en el puerto D1 titila 5 veces.
 output_high(PIN_D1);
 delay_ms(300);
 output_low(PIN_D1);
  delay_ms(300);
 }
 }

void main(void)                      //Función principal main.
{
  set_tris_b(0b00000100);            //Fija el puerto RB2 como entrada y el resto como
// salida.
  set_tris_d(0b00000000);            //Fija el puerto D como salida.
  enable_interrupts(GLOBAL);         //Habilitación de todas(GLOBAL) las interrupciones.
  enable_interrupts(INT_EXT2);       //Habilita la interrupción externa RB2.

 while(TRUE){                        //El LED del PIN D0 titila indefinidamente.
  output_high(PIN_D0);
  delay_ms(500);
  output_low(PIN_D0);
  delay_ms(500);
 }                                   //Fin del bucle infinito.
}                                    //Fin del main.
```

Interrupción RB

La interrupción RB se produce por el cambio de estado en los puertos RB4:RB7. No hay control del flanco. Las directivas y funciones para gestionar esta interrupción son:

#INT_RB; esta directiva solicita la ejecución de la interrupción externa RB4:RB7.

enable_interrupts(GLOBAL); habilitación global de todas las interrupciones.

enable_interrupts(INT_RB); habilitación de la interrupción externa RB4:RB7.

El compilador CCS controla los bits del registro ***INTCON*** en forma transparente para el programador y en el caso de la interrupción externa ***int_ex,*** no hay ningún problema en el funcionamiento, lo que no ocurre con la interrupción ***int_rb.***

Veamos en detalle el funcionamiento del registro INTCON para comprender mejor. El registro INTCON está formado por los bits mostrados en la figura 4.4.

INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Figura 4.4. Bits del registro INTCON.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Antes de la interrupción:

- GIE= 1 (Habilitación de las interrupciones globales activada).
- RBIE=1 (Permiso de interrupción por cambio de estado en RB7:RB4)
- RBIF=0 (Señalización por cambio de estado de las patillas RB7:RB4 a nivel bajo).

Al momento de la interrupción:

- GIE= 0 (Habilitación de las interrupciones globales desactivada).
- RBIE=1 (Permiso de interrupción por cambio de estado en RB7:RB4).
- RBIF=1 (Indica que se ha producido la interrupción por el cambio de estado de las patillas RB7:RB4 a nivel alto).

Al salir de la interrupción:

- GIE= 1 (Habilitación de las interrupciones globales activada).
- RBIE=1 (Permiso de interrupción por cambio de estado en RB7:RB4)
- RBIF=0 (Señalización por cambio de estado de las patillas RB7:RB4 a nivel bajo).

El último paso no se cumple en el PIC18F4550 (existe el mismo problema con los PIC16F8XX) y el programa no sale de la función de interrupción, por lo cual se debe considerar la nota que Microchip indica en relación al bit RBIF:

"*A mismatch condition will continue to set this bit. Reading PORTB, and then waiting one additional instruction cycle, will end the mismatch condition and allow the bit to be cleared.*"

Una condición de conflicto fija el bit RBIF (mantiene RBIF= 1), el mismo bit puede ser aclarado (RBIF= 0) mediante una lectura del Puerto B. Si no se realiza la lectura del puerto, otra forma de solucionar el problema es fijando los valores del registro *INTCON* mediante software. Para el caso *INTCON*= 0b10001000. Recuerde GIE= 1, RBEI= 1, RBIF= 0 y los demás bits en 0.

Otra consideración importante en el programa es la identificación de la dirección de memoria que ocupa el registro *INTCON*. De acuerdo al datasheet del PIC18F4550, el registro INTCON ocupa la dirección en la localidad FF2.

Ejercicio 4.4. Manejo de la interrupción RB.

El programa utilizado con la interrupción externa se implementa para la interrupción RB (RB4:RB7).

PROGRAMA:

```
//USO DE LA INTERRUPCIÓN EXTERNA RB4---RB7
#include <18f4550.h>  //Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOWRT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000) /  / F_OSC = 12MHz.
#BYTE port_b= 0xF81  //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83  //Identificador para el puerto d en la localidad 0xF83.
#byte INTCON = 0xff2          //Identificador del registro INTCON en la localidad 0xFF2.
int n =0;

//Función interrupción
#int_rb                       //Habilita la interrupción RB4:RB7.
PortB_Interrupt()             //Nombre de la función.
{
INTCON= 0b10001000;           //Inicializa el registro INTCON.
 for(n=1; n<=5; n++){         //El LED conectado en el pin D1 titila 5 veces.
 output_high(PIN_D1);
 delay_ms(300);
 output_low(PIN_D1);
  delay_ms(300);
 }
 }

void main(void) {             //Función principal main.
  set_tris_b(0b11110000);     //Fija al Puerto B4:B7 como entrada y el resto como salida.
  set_tris_d(0b00000000);     //Fija al Puerto D como salida.

  enable_interrupts(GLOBAL); //Habilitación de todas(GLOBAL) las interrupciones.
  enable_interrupts(INT_RB);  //Habilita la interrupción externa RB4:RB7.

  while(TRUE){                //El LED conectado al pin D0 titila indefinidamente.
  output_high(PIN_D0);
  delay_ms(500);
  output_low(PIN_D0);
  delay_ms(500);
 }                            //Fin del bucle infinito.
}                             //Fin del main.
```

La figura 4.5, muestra las conexiones para el uso de la interrupción externa RB.

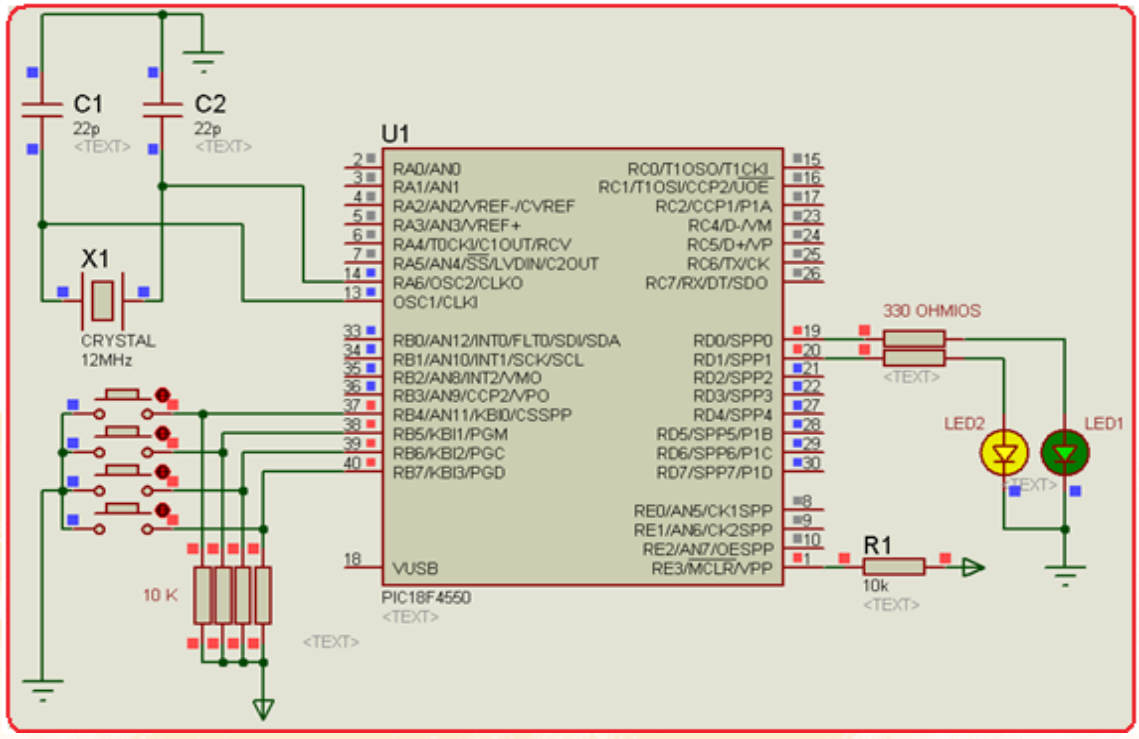


Figura 4.5. Circuito para comprobar la interrupción RB.
Elaborado por: Los Autores.

EJERCICIOS ADICIONALES

Ejercicios resueltos

Contador ascendente – descendente. Mediante una interrupción por el puerto RB0, cambiar el sentido de cuenta de un contador MOD-10. El contador siempre empezará la cuenta en 0 (ascendente) o en 9 (descendente). Inicialmente el contador debe ser ascendente.

Se a escribir dos funciones para el contador ascendente void ascendente(void) y void descendente(void) para el contador descendente. Mediante un bucle for se genera un tiempo de 1 s para que avance o retroceda la cuenta del contador. Es fácil escribir delay_ms(1000); y obtener el segundo requerido, pero en este caso el microcontrolador responde a la interrupción una vez que termine este tiempo de espera del delay, lo que es inaceptable en circuitos donde se requiere que la respuesta sea inmediata.

```
for (n=1; n<=25; ++n){
port_d= x[j];
delay_ms(20);
En un array se almacena los datos codificados para el display de 7 segmentos.
const int x[]= {64,121,36,48,25,18,2,120,0,16};
```

Mediante la instrucción port_d= x[j]; se obtiene los datos en el puerto D. La variable j actúa de índice para apuntar al dato del array. La variable i determina si el contador es ascendente o descendente. Para, i = 0 descendente, i = 1 ascendente. Para que el contador empiece en 0 o 9, la variable j tomará estos valores límites.

PROGRAMA:

```
#include <18f4550.h>    /                 /Librería para usar el PIC18F4550.
#fuses HS, NOPROTECT, NOWRT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)               //FOSC = 12MHz.
#include <lcd.c>                          //Incluye librería de manejo del LCD.
#include <stdlib.h>                       //Incluye librería stdlib.h.
#BYTE port_b= 0xF81                       //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83                       //Identificador para el puerto d en la localidad 0xF83.
int1 i = 1;                               //Contador ascendente i = 1, descendente I = 0.
signed int j;                             //Variable índice para apuntar a los elementos del array.
int n;                                    //Variable para repetir 25 veces y generar 1 s.
const int x[]= {64,121,36,48,25,18,2,120,0,16}; //Datos decodificados del 0 al 9.

#int_ext                                  //Interrupción externa.
void PortB_Interrupt(void)                //Función de la interrupción externa.
{
  i=~i;                                   //Invierte la variable i.
  if (i==0) {                             //Si i = 0 (contador descendente).
port_d = 16;                              //El Puerto D = 16 (corresponde al 9 en 7 segmentos).
j=9;                                      //La variable j empieza en 9.
  }
  if (i==1) {                             //Si i = 1 (contador ascendente).
port_d = 64;                              //El Puerto D = 64 (corresponde al 0 en 7 segmentos).
j=0;                                      //La variable j empieza en 0.
  }
}

void ascendente(void){                    //Función contador ascendente.
  for (j=0; j<=9;++j){                    //La variable j cambia de 0 a 9 incrementa en 1.
    for (n=1; n<=25;++n){                 //Genera 1 s.
      port_d= x[j];         /             /Asigna al Puerto D, el valor indicado por j en el vector x.
      delay_ms(20);                       //Retardo de 20 ms.
```

```
    }
  }
  return;                          //Retorno de la función.
}

void descendente(void){            //Función contador descendente.
 j=9;                              //Inicializa j en 9.
 while (j>=0){                     //Mientras j es mayor o igual a 0....
  for (n=1; n<=25; ++n){           //...repita 25 veces para generar 1s y en ..
    port_d= x[j];          /       /... el Puerto D asigne el valor indicado por j en el vector x.
    delay_ms(20);                  //Retardo de 20 ms.

  }
  --j;                             //Decrementa j en 1.
 }
 return;                           //Retorno de la función
}

void main(void){                   //Función principal main.
  set_tris_b(0xFF);                //Fija el puerto B como entradas.
  set_tris_d(0x00);                //Puerto D  como salidas.
   enable_interrupts(INT_EXT);     //Habilita interrupción externa.
   enable_interrupts(GLOBAL);      //Habilita interrupciones globales.
   setup_adc(adc_off);             //Desactiva ADC.
   ext_int_edge(H_TO_L);           //Habilita la interrupción en flanco de bajada.

 while(TRUE){                      //Bucle infinito.
   if (i ==1){                     //Si i = 1 contador ascendente.
      ascendente();                //Función contador ascendente.
    }
    if (i ==0){                    //Función contador descendente
      descendente();               //Función contador descendente.
    }
 }                                 //Fin del bucle infinito
}                                  //Fin del main.
```

Para reflexionar y comprobar.

*¿Por qué **j** es una variable con signo? ¿Qué pasaría si fuera solo un tipo int.?*

Para reflexionar y comprobar.

¿Por qué j es una variable con signo? ¿Qué pasaría si fuera solo un tipo int.?

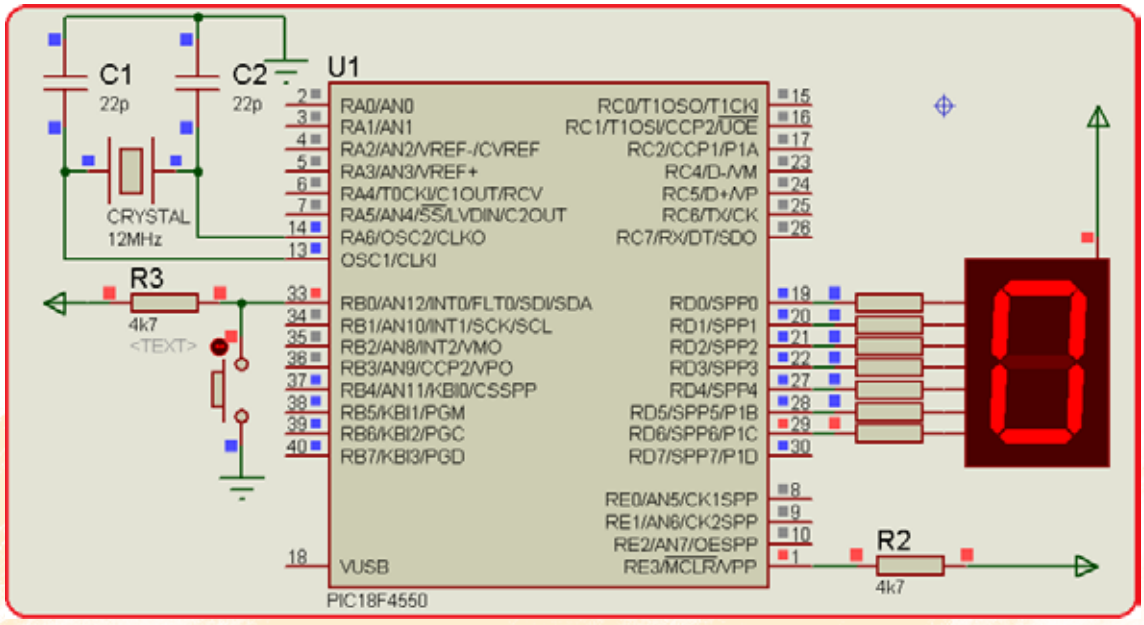


Figura 4.6. Contador ascendente – descendente.
Elaborado por: Los Autores.

Detector de cruce por cero, para disparar un SRC.

Microchip Technology INC., sugiere que se puede construir un sencillo circuito detector de cruce por cero conectando directamente la fuente de alimentación de 120 V_{AC} al puerto RB0 a través de una resistencia de 5MΩ, como muestra la figura 4.7.

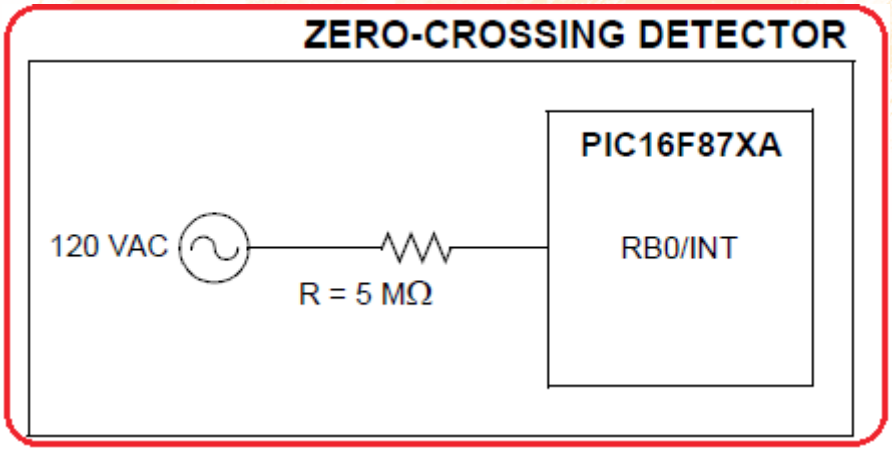


Figura 4.7. Detector de cruce por cero.
Fuente: Home Automation Using the PIC16F877A. Microchip Tecnhology INC.

Para el desarrollo del programa se debe tomar en cuenta que cada vez que exista un cruce por cero, el puerto RB0 detecta el cruce y debe generar un pulso. Dado que el SCR conduce en el semiciclo positivo el flanco que debe detectar será de bajo a alto, es decir cuando la señal de CA, cambie del semiciclo negativo al positivo.

PROGRAMA:

```
#include <18f4550.h>    /                  /Librería para usar el PIC18F4550.
#fuses HS, NOWRT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)               //FOSC = 12MHz.
#include <lcd.c>                          //Incluye librería de manejo del LCD.
#include <stdlib.h>                       //Incluye librería stdlib.h.
#BYTE port_b= 0xF81                       //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83                       //Identificador para el puerto d en la localidad 0xF83.

#int_ext                                  //Interrupción externa.
void PortB_Interrupt(void)                //Función interrupción por RB0.
{
 ext_int_edge(L_TO_H);                    //Habilita la interrupción en flanco de bajada.
 output_high(PIN_D0);                     //Puerto D0 en alto, por ...
 delay_ms(1);                             //1 ms.
 output_low(PIN_D0);                      //Puerto D0 en bajo.
 }

void main(void) {          /              /Función principal.
  set_tris_b(0xFF);                       //Fija el Puerto B como entrada.
  set_tris_d(0x00);                       //Fija el Puerto B como salida.
  Portd_d= 0;                             //Todo el Puerto D en 0.
  enable_interrupts(INT_EXT);             //Habilita interrupción externa.
  enable_interrupts(GLOBAL);              //Habilita interrupciones globales.
  setup_adc(adc_off);                     //Desactiva ADC.
  ext_int_edge(L_TO_H);                   //Habilita la interrupción en flanco de bajada.
  while(TRUE){                            //Bucle infinito.
                                          //Resto del programa.
 }                                        //Fin del bucle infinito.
}                                         //Fin del main.
```

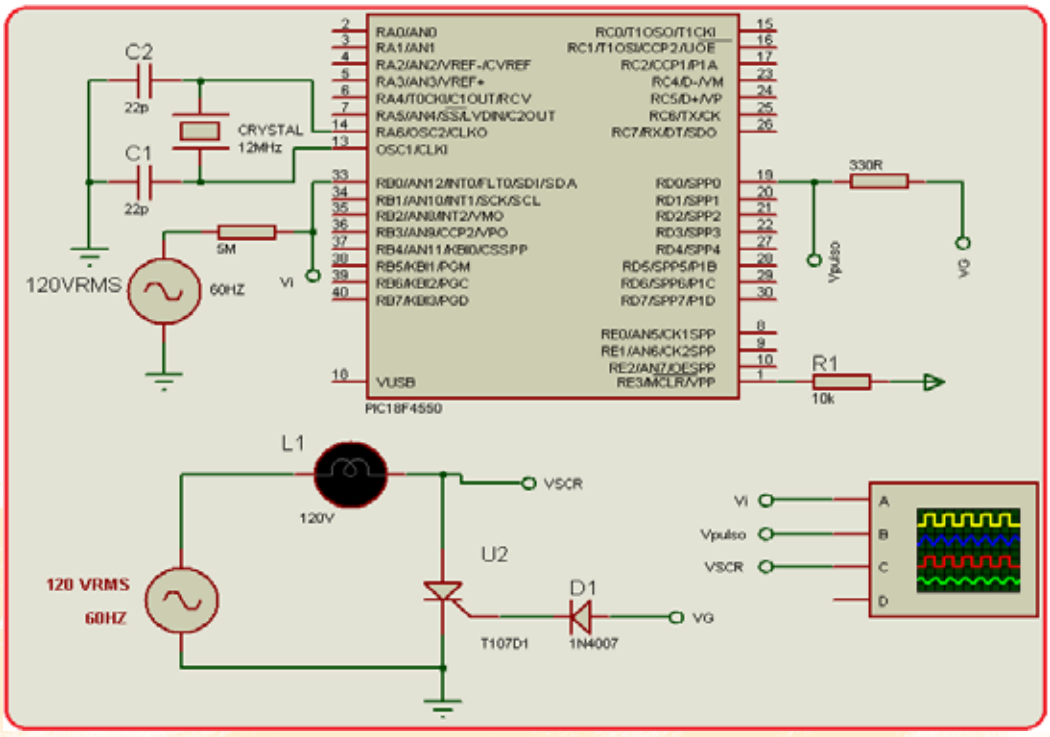
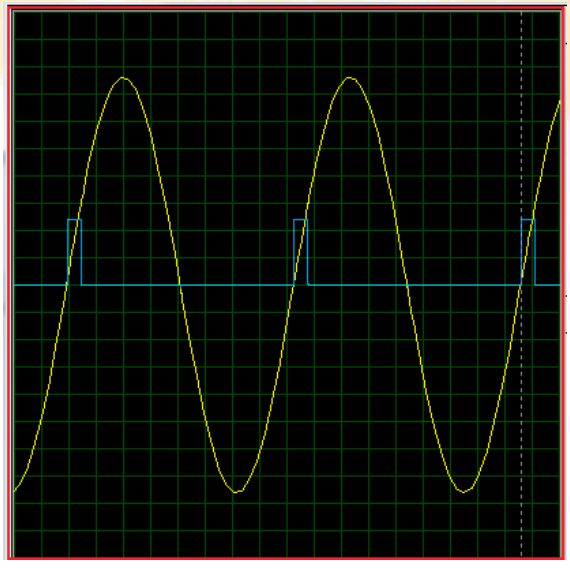
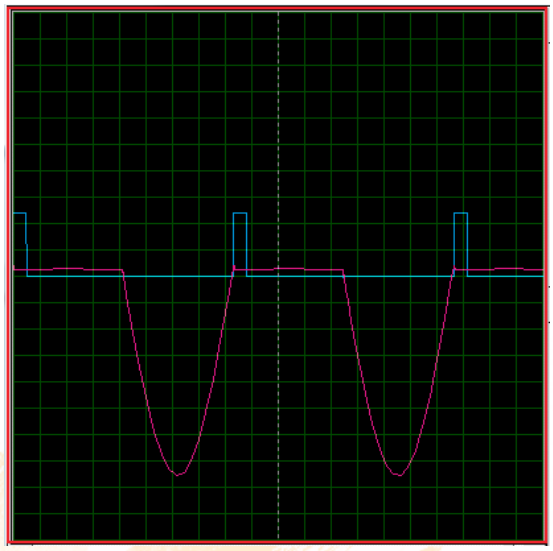


Figura 4.8. Detector de cruce por cero, para disparar un SRC.
Elaborado por: Los Autores.

En la figura 4.9, se presentan las formas de onda visualizadas en el osciloscopio virtual, en las mismas se nota la señal de entrada al puerto RB0 (Vi), el voltaje generado en RD0, que sirve para disparar al SCR y la señal en los terminales de ánodo y cátodo del SCR.



a) Onda de entrada y pulso.



b) Onda del SCR y pulso.

Figura 4.9. Formas de onda del puerto RB0, pulso y VAK.
Elaborado por: Los Autores.

Manejo de teclado matricial usando interrupción RB (RB4:RB7).

Utilizando la interrupción RB4:RB7, se controlará un teclado matricial 4x4. Como se observa en la figura 4.9, las filas FA, FB, FC, y FD se conectan a los puertos RB0, RB1, RB2 y RB3, definidos como salidas. Las columnas C1, C2, C3 y C4 se conectan a los puertos RB4, RB5, RB6 y RB7, definidos como entradas. Además, se activan las resistencias PULL-UP para conectar internamente las columnas a 5. El principio de funcionamiento es el que sigue:

Por cada fila se sacará 0 y 1 en orden de fila en fila, así:

FD=1, FC=1, FB=1, FA=0

FD=1, FC=1, FB=0, FA=1

FD=1, FC=0, FB=1, FA=1

FD=0, FC=1, FB=1, FA=1

Al accionar el pulsador la columna que está normalmente en alto cambia de estado, pasa a bajo lo que provoca la interrupción. La tabla 4.2, muestra los valores que adquiere el puerto B para cada caso en el cual se haya accionado una tecla.

Se lee el valor del puerto B y se asigna el valor correspondiente para ser mostrado en el LCD.

Equivalente en decimal	C4	C3	C2	C1	FD	FC	FB	FA	TECLA
238	1	1	1	0	1	1	1	0	1
237	1	1	1	0	1	1	0	1	4
235	1	1	1	0	1	0	1	1	7
231	1	1	1	0	0	1	1	1	*
222	1	1	0	1	1	1	1	0	2
221	1	1	0	1	1	1	0	1	5
219	1	1	0	1	1	0	1	1	8
215	1	1	0	1	0	1	1	1	0
190	1	0	1	1	1	1	1	0	3
189	1	0	1	1	1	1	0	1	6
187	1	0	1	1	1	0	1	1	9
183	1	0	1	1	0	1	1	1	#
126	0	1	1	1	1	1	1	0	A
125	0	1	1	1	1	1	0	1	B
123	0	1	1	1	1	0	1	1	C
119	0	1	1	1	0	1	1	1	D

Tabla 4.2. Estado del puerto B cuando se accionan las teclas.
Elaborado por: Los Autores.

PROGRAMA:

```
#include <18f4550.h>    /                  /Librería para usar el PIC18F4550.
#fuses HS, NOWRT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=12000000)               //F_OSC = 12MHz.
#include <lcd.c>                          //Incluye librería de manejo del LCD.
#include <stdlib.h>                       //Incluye librería stdlib.h.
#BYTE port_b= 0xF81                       //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83                       //Identificador para el puerto d en la localidad 0xF83.
#byte INTCON = 0xff2                       //Identificador del registro INTCON en la localidad 0xFF2.
#int_rb                                   //Interrupción RB4:RB7.
VOID PortB_Interrupt(VOID)                //Función para interrupción.
{
  INTCON= 0b10001000;                     //Inicializa el registro INTCON.

  IF(port_b== 238){                       //Si el puerto b es igual a 238, se ha pulsado la tecla 1…
   lcd_putc("1\b");                       //…..muestre en el LCD el 1.
  }
  IF(PORT_B== 237){                       //……..
    lcd_putc("4\b");
  }
  IF(PORT_B== 235){
    lcd_putc("7\b");
  }
```

```
 IF(PORT_B== 231){
   lcd_putc("*\b");
 }
 IF(PORT_B== 222){
   lcd_putc("2\b");
 }
 IF(PORT_B== 221){
  lcd_putc("5\b");
 }
 IF(PORT_B== 219){
   lcd_putc("8\b");
 }
 IF(PORT_B== 215){
   lcd_putc("0\b");
 }
 IF(PORT_B== 190){
   lcd_putc("3\b");
 }
 IF(PORT_B== 189){
   lcd_putc("6\b");
 }
 IF(PORT_B== 187){
   lcd_putc("9\b");
 }
  IF(PORT_B== 183){
   lcd_putc("#\b");
  }
 IF(PORT_B== 126){
   lcd_putc("A\b");
 }
 IF(PORT_B== 125){
   lcd_putc("B\b");
 }
 IF(PORT_B== 123){
  lcd_putc("C\b");
 }
  IF(PORT_B== 119){
   lcd_putc("D\b");
  }
}
void main(void)                          //Función principal main.
{
  LCD_INIT();                            //Inicializa el LCD.
  set_tris_b(0XF0);                      //Fija el Puerto B como entrada(RB4:RB7)/salida (RB0:RB4).
  set_tris_d(0X00);                      //Puerto D como salida.
  port_b_pullups(TRUE);                  //Activa las resistencias PULL UP del puerto B.
  enable_interrupts(int_rb);             //Habilita interrupción RB4:RB7.
  enable_interrupts(GLOBAL);             //Habilita interrupciones globales.
  lcd_putc("\f");                        //Limpia el LCD.
  lcd_putc("\fTECLA PULSADA ES\N");          //LA TECLA PULSADA ES …….
```

```
while(TRUE){                    //Bucle infinito.
  port_b=14 ;                   //FA= 0 , el resto en 1.
 port_b=13 ;                    //FB= 0 , el resto en 1.
 port_b=11 ;                    //FC= 0 , el resto en 1.
 port_b=7 ;                     //FD= 0 , el resto en 1.
 lcd_gotoxy(7,2);               //Posiciona el cursor columna 7 y fila 2.
  //RESTO DEL PROGRAMA
 }                              //Fin de bucle infinito.
}                               //Fin del main.
```

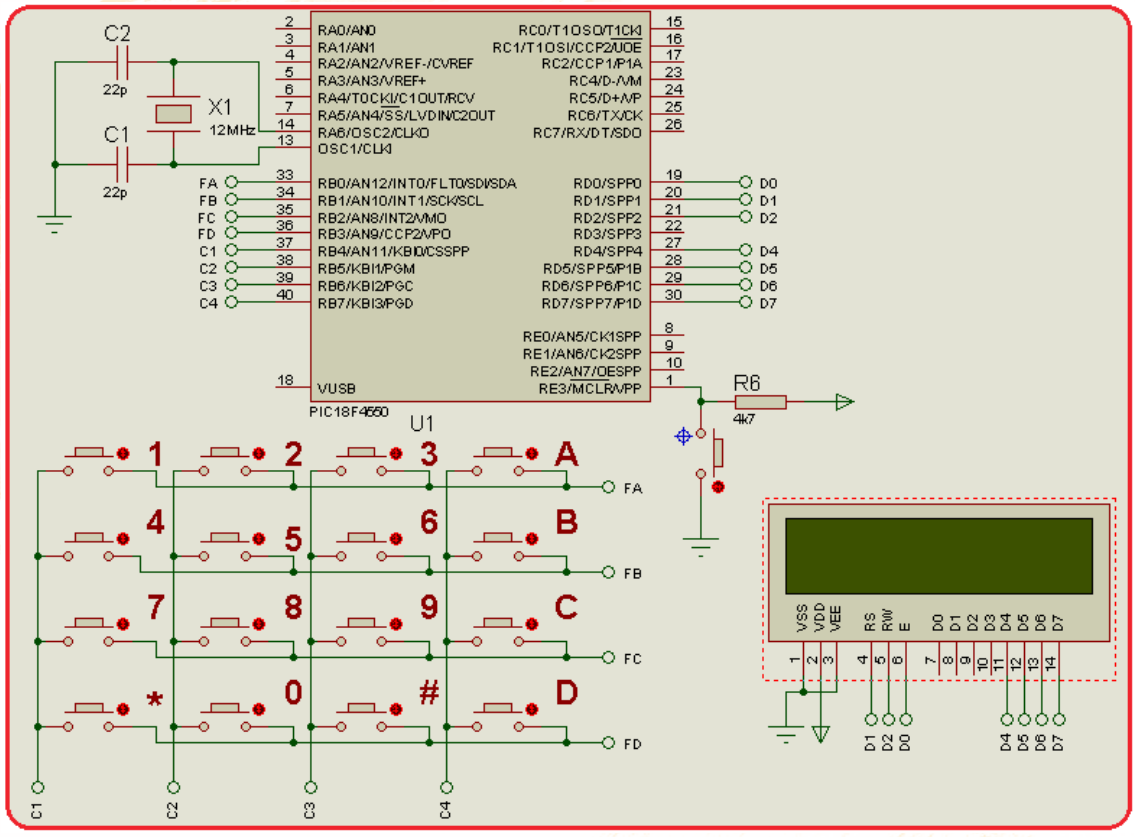


Figura 4.10. Manejo de teclado 4X4 mediante interrupción RB4:RB7.
Elaborado por: Los Autores.

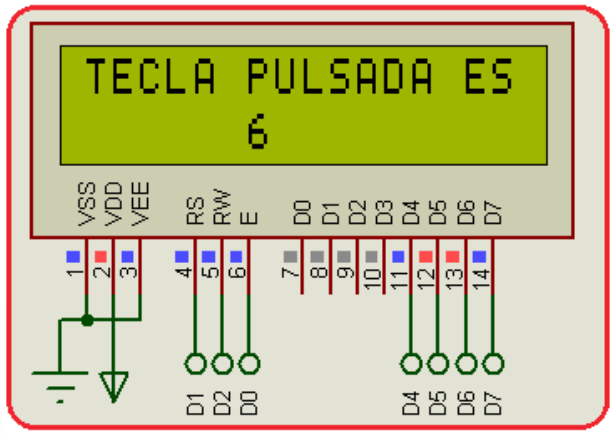


Figura 4.11. Imagen del resultado en el LCD al pulsar la tecla "6".
Elaborado por: Los Autores.

Ejercicios propuestos

Diseñe un contador ascendente – descendente. El contador debe cambiar de ascendente a descendente o viceversa, inmediatamente de accionar un pulsador y empezará su cuenta en el mismo valor que estaba antes del accionamiento del pulsador. Utilice la interrupción externa para el cambio pedido.

Diseñe un dado electrónico en un display de 7 segmentos que genere del 1 al 6, cada vez que se presione un pulsador de conectado al puerto RB1. Mediante una interrupción se puede borrar el dado, para un nuevo juego, para este caso se dispone de otro pulsador. Usar la interrupción externa RB0.

Realice un circuito para generar pulsos de disparo para un TRIAC, utilizando la interrupción externa.

Incrementar un contador cada vez que se actúe sobre un pulsador conectado en RB1. Activar un LED cuando la cuenta llegue a 5 y apagar cuando llegue a nueve. Usar la interrupción RB1.

Activar un LED, al accionar cualquier pulsador conectado en los puertos RB4:RB7. Usar la interrupción RB.

Se desea automatizar el control para el arranque y parada de dos motores (M1 y M2), cumpliendo las siguientes especificaciones:

MOTOR 1: El arranque se producirá por un pulso momentáneo sobre dos pulsadores simultáneamente, produciéndose el enclavamiento.

MOTOR 2: El arranque será similar al anterior, siendo la condición de enclavamiento que el MOTOR 1 esté activado 5 segundos (utilizar los mismos pulsadores).

Ambos motores dispondrán del correspondiente pulsador de paro y relé térmico de protección, cuya activación será señalizada. La señal de los relés térmicos debe ser procesada por interrupciones externas. No utilizar *delay* superiores 250 ms.

Con un pulsador de "INICIO", se activa la bomba B y permanece encendida hasta alcanzar el nivel máximo y se apagará. Se abre la válvula de vaciado V. La bomba B se activa de forma automática al llegar al nivel mínimo cerrando la válvula de vaciado V. Nuevamente en forma automática se activa la bomba B y se repite el ciclo. Además se dispone de dos sensores de REVOSE e INSUFICIENTE, que actuarán cuando ocurra una falla en los sensores NIVEL MÁXIMO y MÍNIMO. En este caso bloquearán inmediatamente la BOMBA y la VÁLVULA hasta reiniciar el sistema con el botón INICIO. Debe existir un botón de paro de emergencia para apagar el motor o cerrar la electroválvula V. Tres LEDS indicarán la situación respectiva, máximo, mínimo y funcionamiento normal. Usar interrupciones externas y RB, según las necesidades. El circuito de la figura 4.11 describe la situación del sistema de control para el tanque.

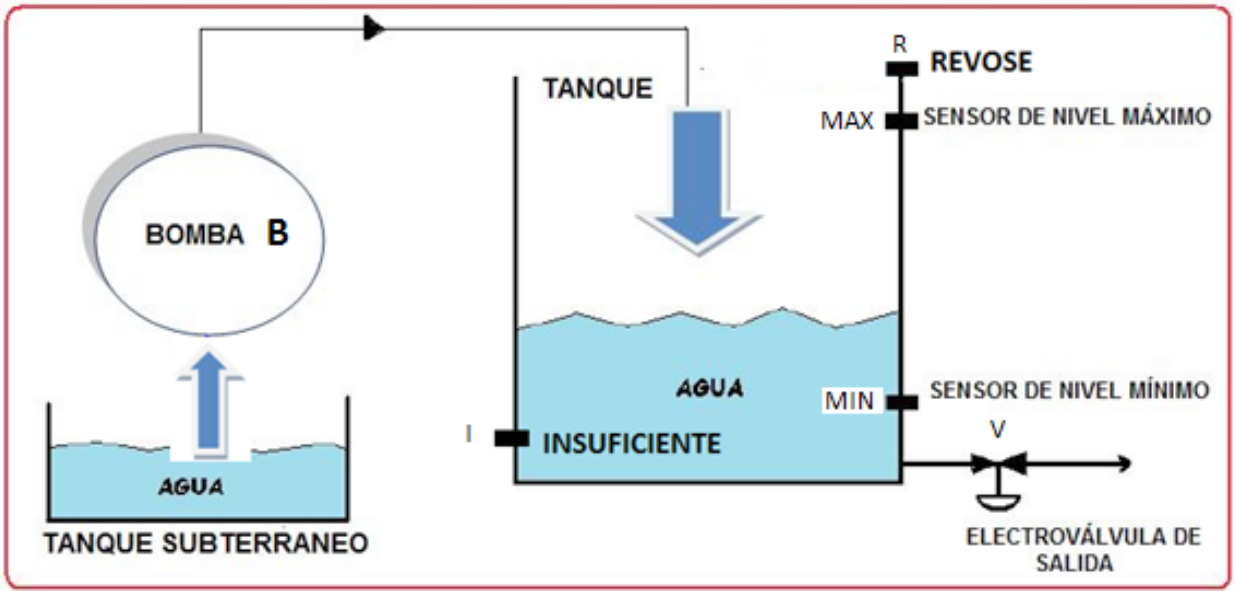


Figura 4.12. Sistema de control de nivel de un tanque.
Elaborado por los autores.

BIBLIOGRAFÍA

Eduardo García Breijo. Compilador C CSS y Simulador PROTEUS para Microcontroladores PIC. MARCOMBO, S.A. Barcelona - España. 2008

PIC18F2455/2550/4455/4550 Data Sheet. Microchip Tecnhology INC.: Disponible en Internet: www.microchip.com.

Home Automation Using the PIC16F877A. *Microchip Tecnhology INC.* Disponible en Internet: www.microchip.com.

CCS C Compiler Manual PCD. Disponible en Internet: https://www.ccsinfo.com/downloads/ccs_c_manual.pdf

Software CCS. Disponible en Internet: http://www.ccsinfo.com/ccsfreedemo.php.

Manual de usuario del Compilador PCW de CCS. Disponible en Internet: http://www.cursos.ucv.cl/eie48700/referencias/CCS_C_Manual.pdf

Electrónica, Microcontroladores y Psicología. Melodías con PIC. Disponible en Internet: https://picmind.es.tl/Melod%EDas-con-PIC.htm

Profesor Rangel. Microcontroladores PIC Gama Alta. Disponible en Internet: http://profesor-rangel.blogspot.com/p/manejo-de-interrupciones.html

Publicaciones Científicas

ISBN: 978-9942-765-36-9

ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA