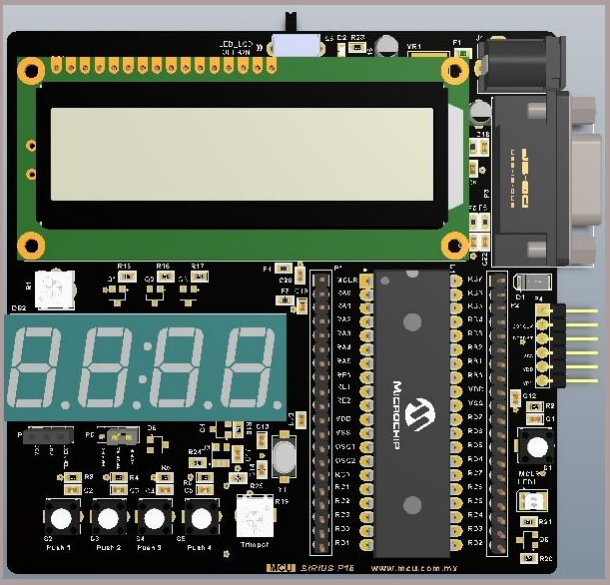
Manual de programación para SIRIUS P16



24 MAYO

MCU
Creado por: Francisco A. Martínez

1 Tabla de contenido

2 Tabla de ilustraciones

3 Introducción

Este manual brinda los conocimientos necesarios para que desarrolles proyectos en el microcontrolador PIC16F877A, además de que pondrás en práctica diferentes ejercicios con la ayuda de nuestra tarjeta de desarrollo **SIRIUS P16**. Esta tarjeta contiene lo necesario para que inicies el camino de la programación a los microcontroladores en lenguaje C.

En nuestro amplio catálogo de tarjetas **SIRIUS** podrás encontrar diferentes familias de microcontroladores, algunas de estas tarjetas serán más complejas y con más aplicaciones.

En la mayoría de nuestras tarjetas **SIRIUS** se cuenta con los siguientes componentes:

I. Pulsadores
II. LED RGB
III. Sensores de temperatura
IV. Display de 7 segmentos
V. Display LCD 16x2
VI. Puerto RS-232
VII. Memoria EEPROM (I2C)

El documento va enfocado para estudiantes, practicantes, aficionados o profesionales en el ámbito de la programación de microcontroladores. Al ser una tarjeta de entrenamiento, este documento comenzará con ejercicios básicos, y conforme se avance los temas serán un poco más complejos.

El objetivo de nuestro trabajo es orientar a todos ustedes a los sistemas embebidos, debido a que el futuro está enfocado en estos sistemas, con estos sistemas se pueden controlar un sinfín de equipos, además de que en la industria y en el hogar se manejan sistemas que manejen el IOT.

En el mercado se pueden hallar tarjetas con sistemas de comunicación por WiFi, Bluetooth, Ethernet, etc. Con estas tecnologías es posible hacer sistemas para IOT, y controlar cualquier aparato desde un teléfono o computadora.

4 Introducción al SIRIUS P16

La tarjeta de desarrolla soporta microcontroladores de la familia PIC16F y PIC18F en encapsulado 40-DIP, para la explicación del manejo de esta tarjeta nos enfocaremos en el PIC16F877A.
Las familias que son compatibles son las siguientes:

- PIC16F887-I/P
- PIC16F18877-I/P
- PIC16F18875-I/P
- PIC16F19175-I/P
- PIC18F4550-I/P
- PIC18F45K40-I/P

Sí se desea algún otro dispositivo que no se encuentre en la lista revisar el Pinout de su dispositivo con el que se usa en este manual.

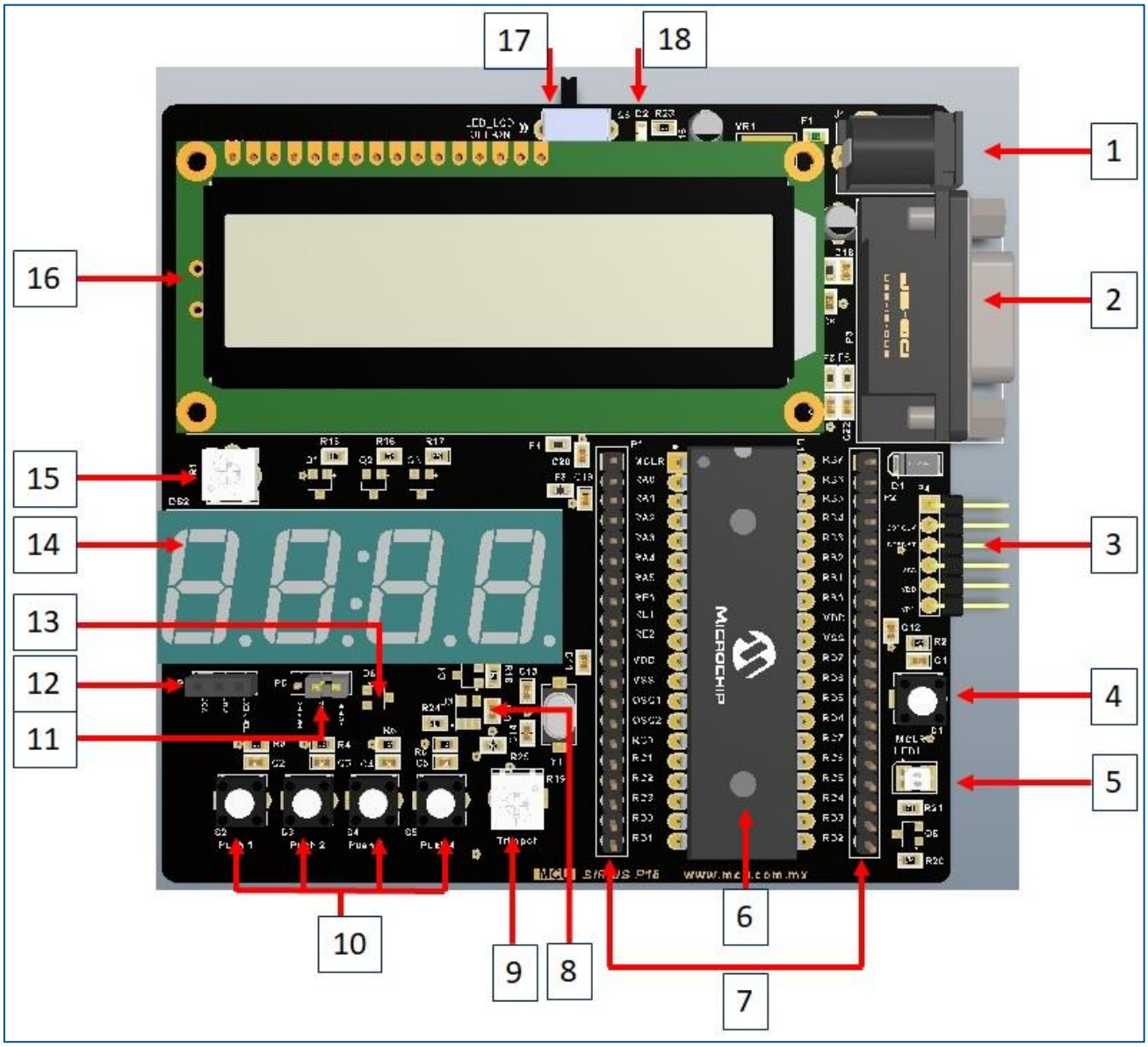
Para conocer el Pinout de este microcontrolador en específico puede dirigirse al siguiente link https://www.microchip.com/wwwproducts/en/PIC16F877A.

4.1 Contenido del kit de desarrollo SIRIUS P16

La tarjeta de desarrollo contiene una gran cantidad de componentes los cuales pueden ser utilizados al mismo tiempo, esto con ciertas características de programación, pero como vamos a comenzar desde cero, los componentes los usaremos uno a uno, hasta conformar un código el cual sea capaz de responder y controlar todos los componentes sin ningún contra tiempo.

A continuación, se muestra los diferentes componentes que integran a SIRIUS P16.

Imagen 4-1 Tarjeta SIRIUS P16



1. Jack de alimentación (9-12V).
2. Puerto DB-9 para comunicación RS-232.
3. Pin de programación (para grabador MCU_PROG) compatible con pickit2.
4. Botón de reinicio.

5. LED RGB.
6. Microcontrolador DIP-40.
7. Header de conexión al Microcontrolador.
8. Memoria EEPROM.
9. Potenciómetro para señal analógica.
10. Pulsadores
11. Selector de sensor (Interno o externo).
12. Pin de alimentación sensor externo.
13. Sensor de temperatura interna.
14. Display de 7 segmentos 4 dígito.
15. Ajustador de contraste para display LCD.
16. Display LCD 16x2.
17. Interruptor Backlight LCD
18. LED indicador de alimentación.

4.1.1 Descripción de los componentes

4.1.1.1 Jack de alimentación

En este Jack podemos conectar la fuente de alimentación para nuestra tarjeta de desarrollo, los voltajes pueden ir desde los 9V hasta los 12V.
Si se considera usar una fuente mayor a estos voltajes podemos dañar la etapa de regulación de nuestra tarjeta.
Cuando se alimenta por este puerto se encenderá el LED indicador, el cual lo podemos ubicar con el número 18 de la imagen anterior.

4.1.1.2 Puerto DB-9

El conector DB-9 es utilizado para la comunicación serial, utilizando el protocoló RS-232.
Dentro de este documento encontraras la información necesaria para la comunicación entre una computadora y el microcontrolador, o con algún otro dispositivo.
La información correspondiente la podrás encontrar en el capítulo Módulo USART

4.1.1.3 Pin de programación

En estos pines va conectado el grabador, estos pines son totalmente compatibles con nuestros grabadores de la familia PROG_P o VEGA_P, revisar el pin-out de cada una de las tarjetas. De igual forma puede ser conectado cualquier otro grabador para PIC.

4.1.1.4 Botón de reinicio

Este botón es usado para reiniciar el microcontrolador de manera manual, usar solo cuando sea necesario, si es usado durante algún proceso todo volverá al inicio.

4.1.1.5 LED RGB

Esta LED será utilizado para varias actividades, como un indicador de procesos, secuencias, etc.
Al ser un LED RGB es necesario conocer sobre la modulación PWM para obtener todos los colores.

4.1.1.6 Microcontrolador

Este es el cerebro de toda la tarjeta, es la base de todo, y como se menciona en el capítulo Descripción de los componentes se pude montar diferentes microcontroladores.

4.1.1.7 Header de conexión

En este header podemos conectar algún elemento externo. Si se requiere algún sensor externo por este puerto podemos realizar la actividad, al igual que si se desea conectar un osciloscopio para ver alguna señal se puede conectar.

4.1.1.8 Memoria EEPROM

Esta memoria la podemos utilizar para almacenar información utilizando el protocolo I2C, esta comunicación la podrás encontrar en el capítulo Módulos I2C

4.1.1.9 Potenciómetro

El potenciómetro lo utilizaremos para nuestra señal analógica. Este componente se implementará en varias ocasiones.
Tendremos valores de voltaje desde 0 hasta 5V.
Para conocer el funcionamiento de este elemento consulte el capítulo Lectura de Trimpot

4.1.1.10 Pulsadores

Estos cuatro botones serán utilizados para la señal digital, se utilizarán para varias funciones.
Conforme avancemos en los capítulos los implementaremos de diferentes maneras, como son:

a) Encender un LED
b) Configurar la hora de nuestro display.
c) Cambiar de menú
d) Etc.

4.1.1.11 Selector de sensor (externo o interno)

En este header podemos seleccionar que senosr deseamos registrar si con el sensor interno de la tarjeta o con un sensor externo.
La selección la podemos hacer con un jumper.
Para conocer el funcionamiento de este elemento consulte el capítulo Lectura de Temperatura

4.1.1.12 Pin para sensor externo

En este podemos conectar el sensor de temperatura externa o si desea colocar algún otro sensor con una señal analógica lo puede realizar.
Para conocer el funcionamiento de este elemento consulte el capítulo Lectura de Temperatura externa

4.1.1.13 Sensor de temperatura interna

La tarjeta cuenta con un sensor de temperatura interna, al momento de ser utilizado indicara la temperatura ambiente del lugar donde se encuentra la tarjeta.

4.1.1.14 Display de 7 segmentos 4 dígitos

Con este display podemos realizar varias actividades como puede ser:

- Contador de 0 a 999.
- Realizar un reloj formato 24 o 12 horas, esto gracias a los 4 dígitos.
- Mostrar la temperatura con ayuda de los sensores.
- Mostrar voltaje.
- O mostrar un mensaje usando los 7 dígitos, esto sería una muestra previa al uso del display LCD.

Para conocer el funcionamiento de este elemento consulte el capítulo Display de 7 segmentos

4.1.1.15 Ajuste de contraste para LCD

Con este potenciómetro podemos ajustar el contraste de nuestro display LCD.

4.1.1.16 Display LCD 16x2

Este display es uno de los componentes más usados, gracias a que es económico y en el podemos mostrar un sinfín de caracteres y aquí aprenderá a manejar los comandos.
Para conocer el funcionamiento de este elemento consulte el capítulo Control de pantalla LCD

4.1.1.17 Interruptor LED LCD

Este interruptor permite activar o desactivar el LED que ilumina la LCD, solo se debe mover el selector a On/OFF

4.1.1.18 LED indicador de alimentación

Este LED nos será de gran in indicará cuando nuestra tarjeta sea conecta a nuestro Jack de alimentación.

5 Inicializar un proyecto nuevo

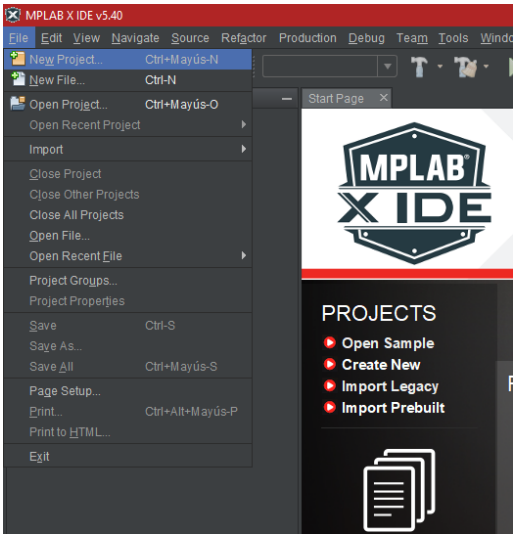
Antes de comenzar con la programación de nuestra tarjeta es necesario conocer los principios básicos para crear un nuevo proyecto, en el programa MPLAB.

Para comenzar debemos de contar con el programa MPLAB X IDE instalado en una computadora, también debemos de contar con el compilador XC8.
https://www.microchip.com/mplab/mplab-x-ide
https://www.microchip.com/en-us/development-tools-tools-and-software/mplab-xc-compilers

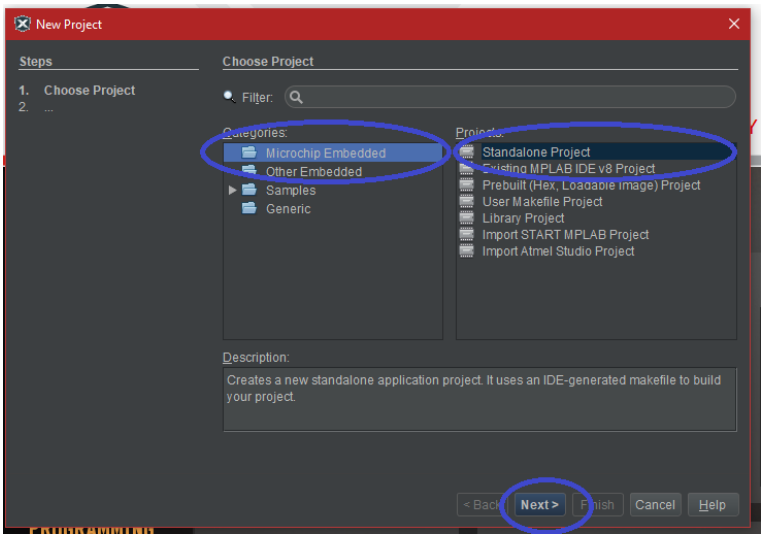
5.1 Pasos para iniciar un nuevo proyecto

I. Abrir el programa MPLEB X IDE

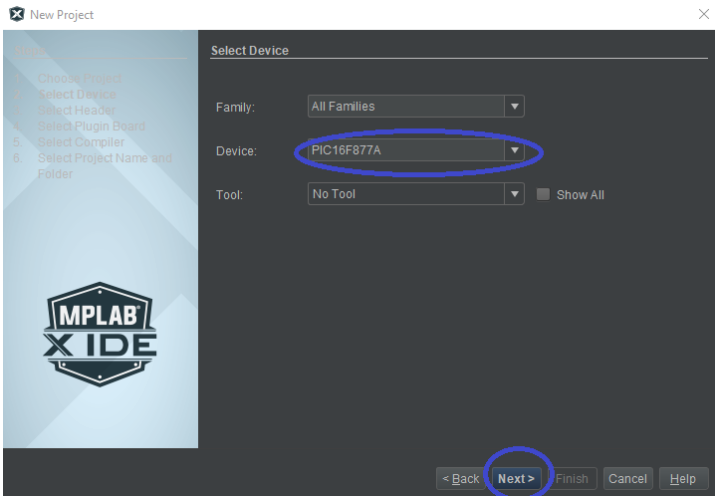
II. Dirigirnos a File->New Project



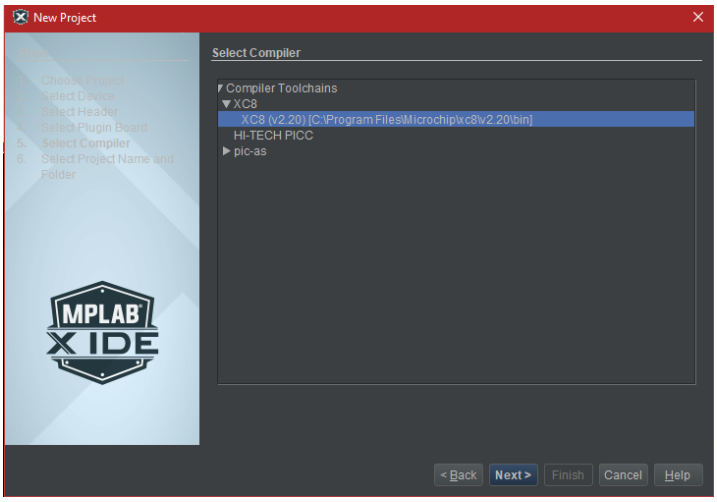
III. En la ventana emergente seleccionamos la categoría de Microchip Embedded, y en proyectos Standalone Project, y a continuación Next.



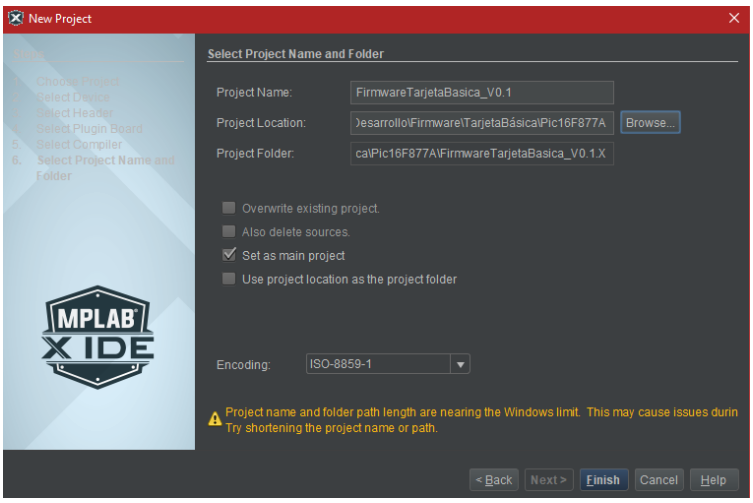
IV. Seleccionamos el tipo de dispositivo, en este caso vamos a utilizar el PIC16F877A, y después hacemos clic en Next.



V. Seleccionamos el compilador para este caso es para XC8, y hacemos clic en Next.



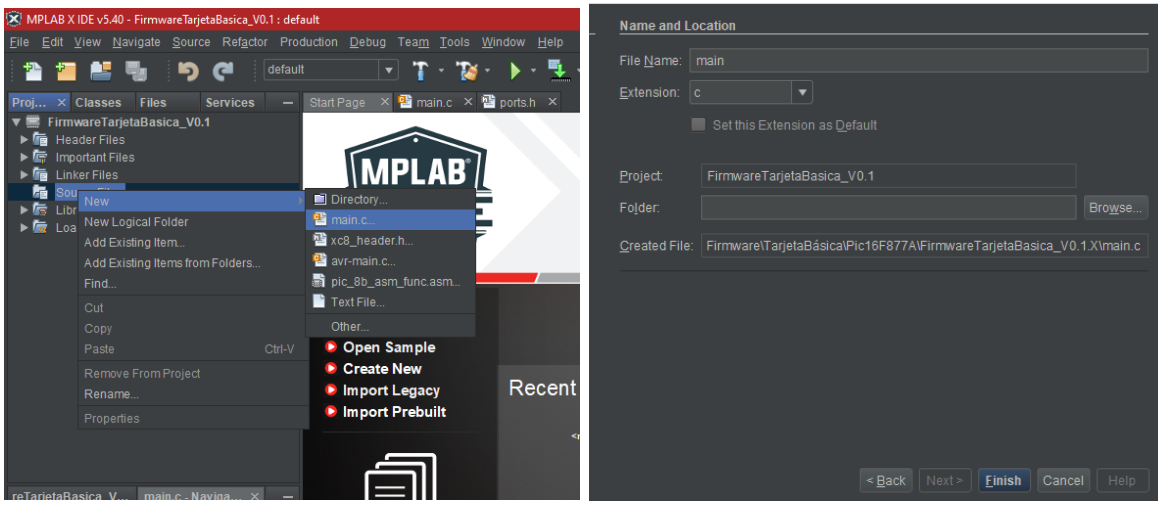
VI. Le asignamos un nombre al proyecto, y elegimos la ubicación de dicho proyecto, hacemos clic en Finish.



VII. Del lado izquierdo estar nuestro proyecto, para agregar archivos .c o .h a nuestro proyecto hacemos clic derecho sobre la carpeta que elijamos para que contenga los archivos.

Por ejemplo, si queremos agregar el main.c nos dirigimos sobre la carpeta Source Files hacemos clic derecho y elegimos New y seleccionamos main.c, le podemos cambiar el nombre.

Ya con esto podemos trabajar en nuestro proyecto.



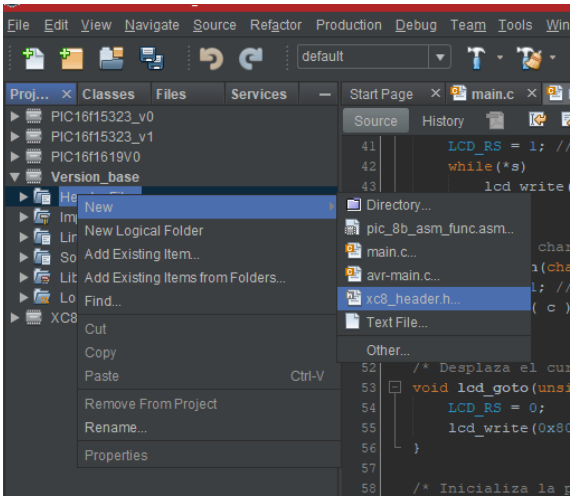
5.1.1 Cabeceras H

VIII. Si queremos agregar cabeceras a nuestro proyecto, e s decir que tengamos un archivo donde se encuentren librerías, variables, o declaremos funciones, es necesario crear un archivo .h, en este archivo podremos colocar todo eso y más. De igual manera podemos agregar archivos .c los cuales realizaran las tareas en específico, esto con el objetivo de que nuestro main principal quede lo más limpio posible.

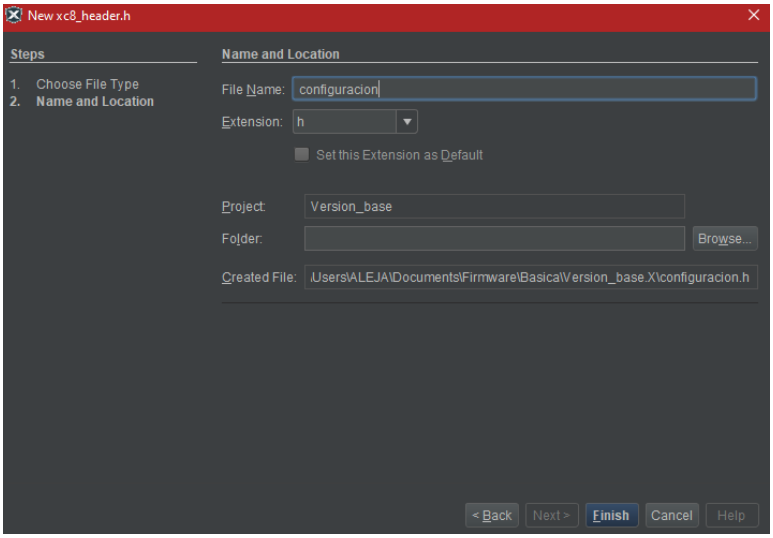
Es recomendable crear estos archivos cuando se tienen varias tareas, o el proyecto realiza varios procesos, de esta forma tendremos organizado nuestro proyecto.

Para ello hacemos lo siguiente

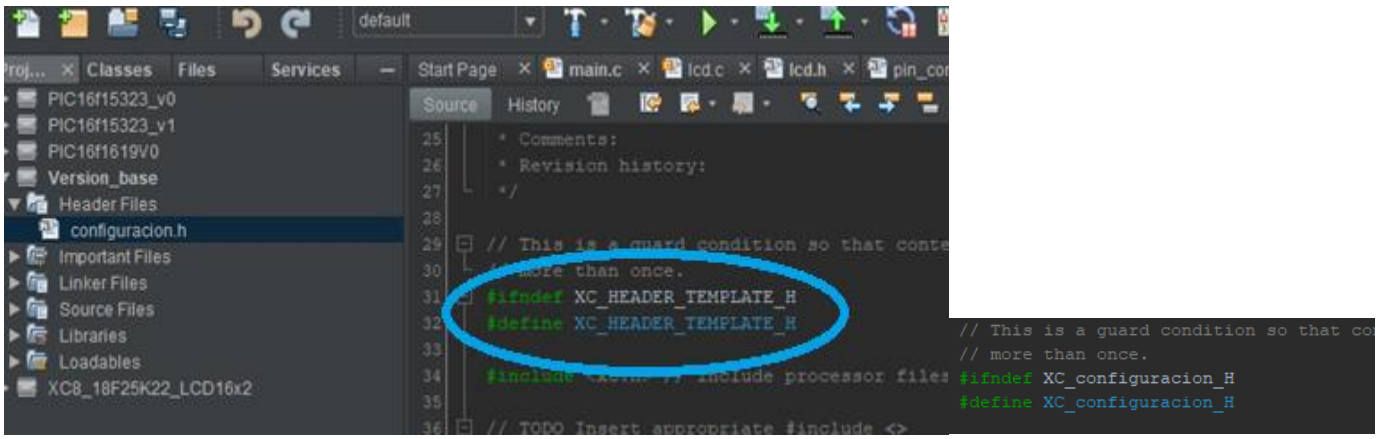
a. Clic derecho sobre la carpeta “Header Files”, new-> xc8_header.h



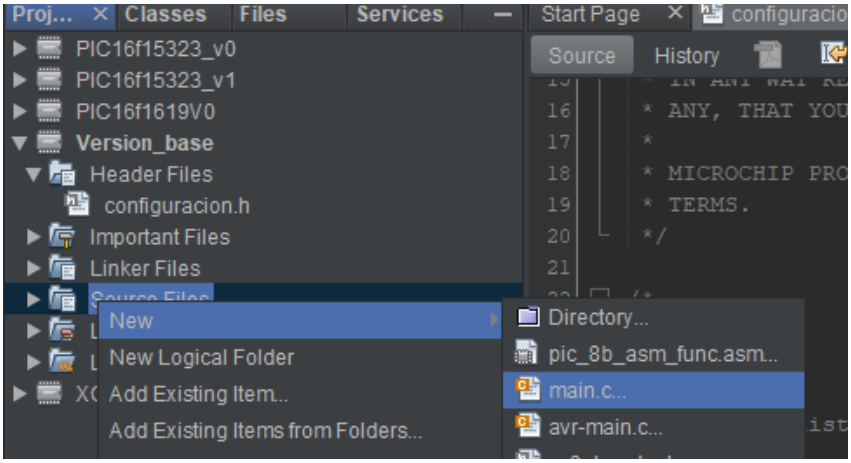
b. Se abrirá una ventana, en la cual escribimos el nombre de nuestro archivo, verificar que la extensión sea “h”, y oprimimos en finish.



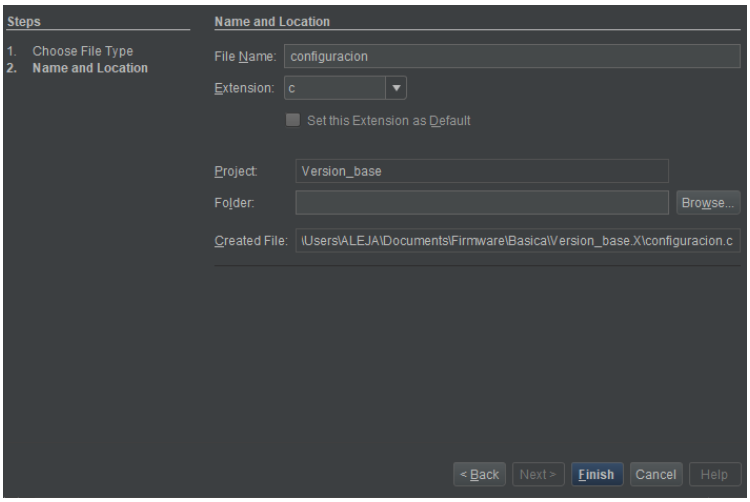
c. Esperamos un momento y quedara listo, antes de continuar debemos de modificar el nombre, de ser necesario debemos de cambiar los siguientes datos, es recomendable escribir el mismo nombre del archivo.



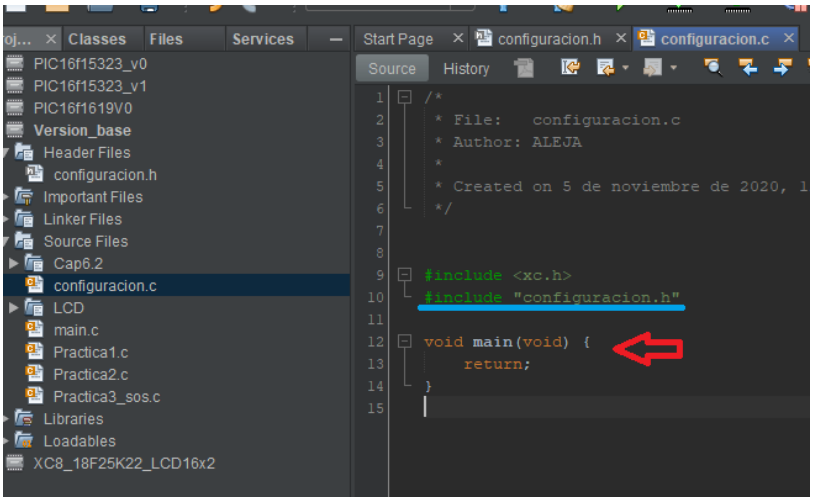
d. Para agregar los archivos .c, hacemos clic derecho sobre “Source Files”, new->main.c



e. Agregamos el nombre del archivo, es recomendable que el nombre sea igual al archivo .h, verificamos que la extensión sea c, y hacemo clic en finish.



f. Agregamos la librería .h que creamos anteriormente, borramos el void main, y podremos agregar las funciones necesarias. En el main se debe de agregar tambien la librería .h



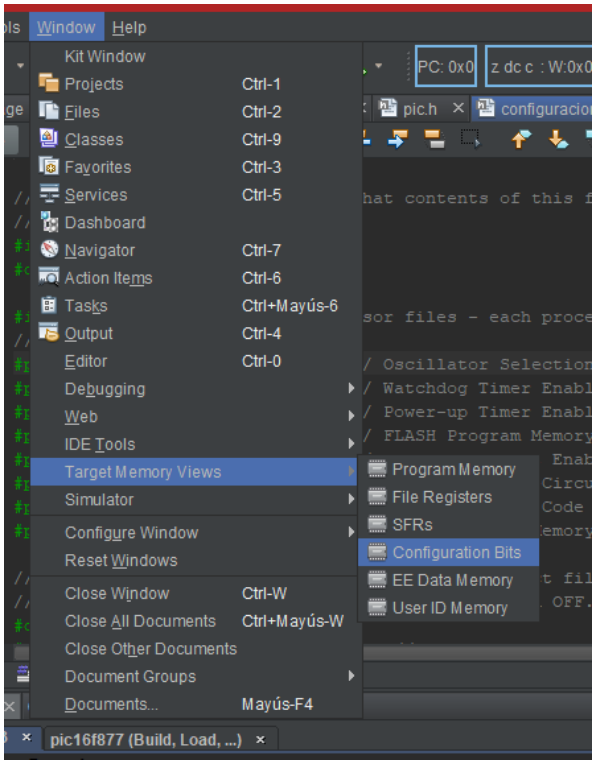
5.2 Configuración de Bits

Debemos de configurar los bits del microcontrolador, estos bits contienen los datos con los que va a iniciar el microcontrolador, tal y como son.

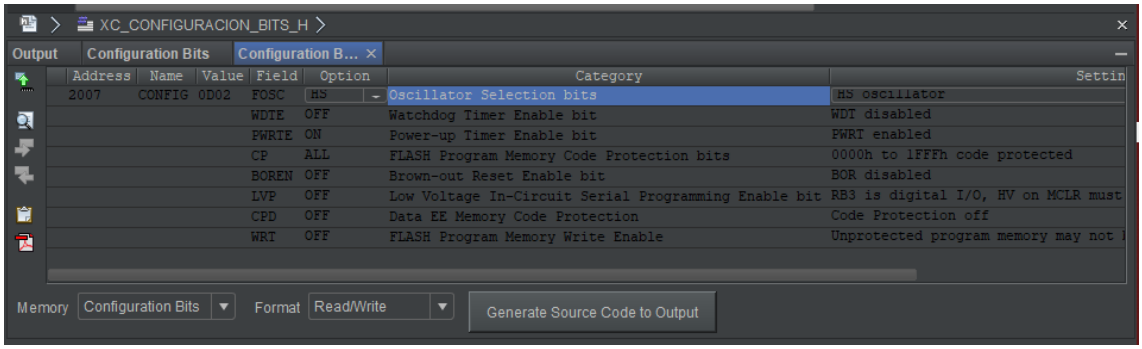
- Tipo de oscilador, interno o externo, de alto o bajo
- Watchdog
- Protección de memoria
- Etc.

Esta información la podemos encontrar en todas las hojas de especificaciones de todos los microcontroladores por ejemplo en la hoja de datos del PIC16F877 en la pagina 120 se encuentra dicha información, ahora bien, para configurar estos bits realizamos lo siguiente.

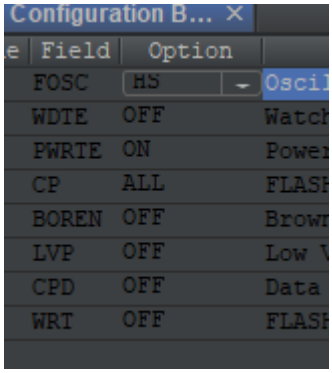
I. Nos dirigimos a Window→Target Memory Views → Configuration Bits



II. Se abrirá una ventana en la parte inferior del MPLAB

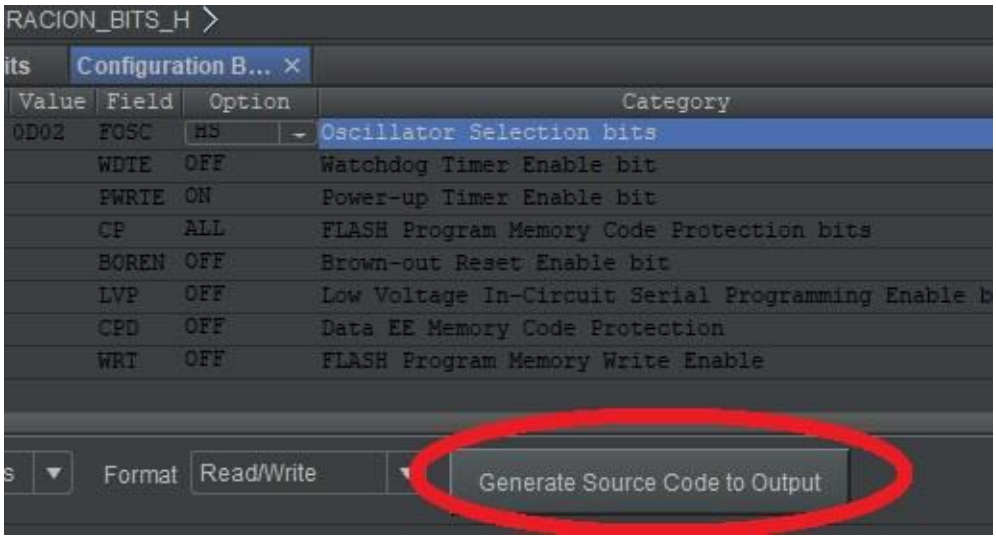


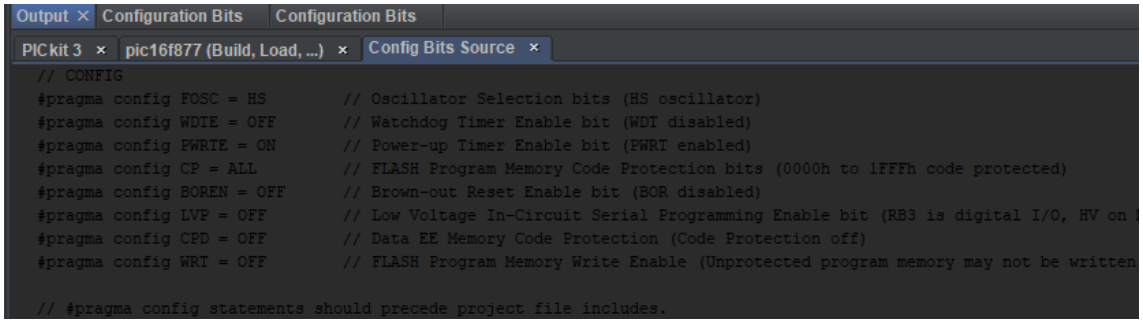
III. En esta ventana se realizará la configuración de los bits, para esta tarjeta y este microcontrolador usaremos la siguiente configuración. Al colocar el cursor en alguno de las opciones mostrara una ventana en la cual se debe de seleccionar la opción correcta.



IV. Al terminar daremos clic en el cuadro de **Generate Source Code To Output**, y nos generara el código, el cual lo demos de copiar en la cabecera del proyecto

ya sea en el main o generar una cabecera de configuración y ahí lo colocaremos, esta cabecera la llamaremos en el main.





La cabecera de configuración la llamaremos “configuracion_bits.h”

```
#ifndef XC_CONFIGURACION_BITS_H
#define   XC_CONFIGURACION_BITS_H

#include <xc.h> // include processor files – each processor file is guarded.
// CONFIG
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = ON       // Power-up Timer Enable bit (PWRT enabled)
#pragma config CP = ALL         // FLASH Program Memory Code Protection bits (0000h to 1FFFh code protected)
#pragma config BOREN = OFF      // Brown-out Reset Enable bit (BOR disabled)
#pragma config LVP = OFF        // Low Voltage In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must
be used for programming)
#pragma config CPD = OFF        // Data EE Memory Code Protection (Code Protection off)
#pragma config WRT = OFF        // FLASH Program Memory Write Enable (Unprotected program memory may not be written to
by EECON control)

// #pragma config statements should precede 17roject file includes.
// Use 17roject enums instead of #define for ON and OFF.
#define __PICCPRO__
#define _XTAL_FREQ 16000000                 //Oscilador Interno de 16MHZ


#endif     /* XC_HEADER_TEMPLATE_H */
```

6 Puertos de entrada y salida

Todos los microcontroladores cuentan con puertos de entrada y salida, existen varios microcontroladores con cantidades diferentes de puertos, depende mucho las aplicaciones si se requiere de pocas entradas o no, para este caso los puertos son desde el PORTA hasta el PORTE. Los puertos de entrada y salida digital son los periféricos más sencillos que dispone un microcontrolador, y mediante estos periféricos podemos controlar otros dispositivos o simplemente supervisar el estado de otros.

Cuando los bits de registro TRIS se escriben con “1” indican que las terminales del PORT son entradas (input), mientras que las que se escriben con “0” indican que son terminales de entrada (output). Esto es fácil de recordar ya que el “1” es similar a la “I” de “Input” y el “0” a la “O” de “Output”.

6.1 Pulsadores y LED

Los elementos básicos son el manejo de pulsadores o switch, con estos componentes se puede ver su uso principalmente en teclados, o para seleccionar dentro de un menú alguna opción, para controles, etc.
Y un segundo elemento básico es el LED, el LED hoy en día es muy frecuente verlo en diferentes equipos, por ejemplo, en lámparas, en los semáforos, en faros para vehículos, etc., hasta es posible encontrarlos en letreros como de gasolineras, o en letreros de alguna tienda mostrando ofertas.
En este capítulo mostraremos el manejo de estos dos elementos.

Demostraremos el funcionamiento de la lectura de los puertos de entradas digitales y la configuración que se requiere para establecer si el puerto es entrada o salida. El objetivo es que usted sea capaz de configurar los pulsadores como lo desea, nosotros mostraremos algunos ejemplos de la aplicación de algunos de estos pulsadores.

6.1.1 Secuencia de encendido

Realizar un programa el cual lea los puertos de los pulsadores (PUSH) y encienda cada uno de los LED del LED RGB, es decir:

- Si pulsamos el PUSH 1 encenderemos el LED rojo y apagamos los demás
- Si pulsamos el PUSH 2 encendemos el LED verde y apagamos los demás
- Si pulsamos el PUSH 3 encendemos el LED azul y apagamos los demás
- Y si pulsamos el PUSH 4 apagaremos todos los LED.

En la tarjeta podemos localizar 4 pulsadores y un led RGB **Imagen 4-1 Tarjeta** , o de una forma general en **Imagen 0-1** se muestra el diagrama de conexión de toda la tarjeta. En la siguiente imagen se muestra las conexiones de los pulsadores y del LED RGB.

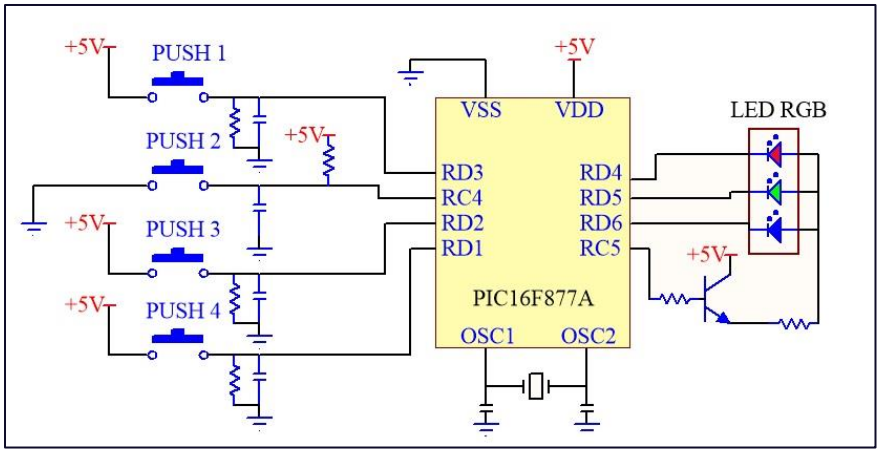


Imagen 6-1 Conexión pulsadores y LED RGB

En la siguiente imagen se muestra la ubicación de estos pulsadores sobre la tarjeta, cada pulsador cuenta con su serigrafia.

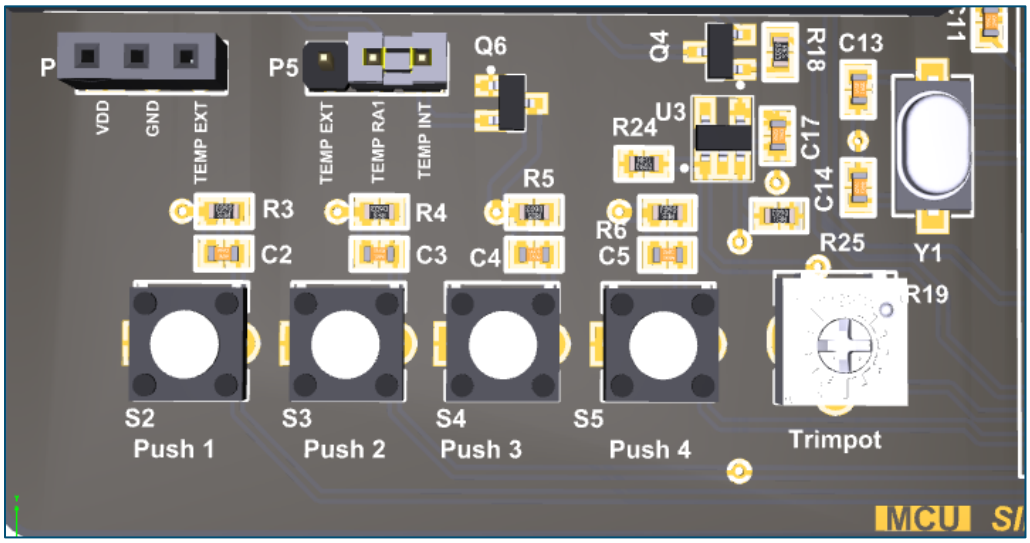


Imagen 6-2 Posición de los pulsadores

Las entradas de los pulsadores son las siguientes

Pulsador	Puerto
S2 (PUSH 1)	RD3
S3 (PUSH 2)	RC4
S4 (PUSH 3)	RD2
S5 (PUSH 4)	RD1

Mientras que las salidas para el LED son las siguientes:

LED color	Puerto
Rojo	RD4
Verde	RD5
Azul	RD6
Ánodo	RC5

Conociendo estos elementos podemos definir cuales puertos configuraremos como salida y cuales como entrada. Como sabemos para configurar los puertos como entrada se debe de asignar el valor 1, mientras que para las entradas se configuran con el valor 0.

Iniciemos con el puerto D, en base a los resultados de las tablas anteriores podemos sacar el valor del puerto, los pulsadores ocupan el RD1 al RD3 por lo tanto se le asigna el 1, mientras que las salidas ocupan del RD4 al RD6 a estas se le asigna el valor 0.

Puerto D							
7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0

Solo nos falta configurar el pulsador S3 el cual esta conecto al RC4 y configurar la salida para el ánodo del LED el cual esta conecto al RC5. RC4 recibe el valor 1 mientras que RC5 se le asigna el valor 0. Mientras que los demás los podemos dejar como salida o entradas, en este caso los dejaremos como salidas.

Puerto C							
7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0

Como ya sabemos que puertos son salidas ahora realizamos una tabla donde haremos el muestreo del encendido de los LED del RGB.
Debemos de considerar que el encendido de cada LED se debe de hacer con un pulso bajo, por lo tanto, los demás deben de quedar en alto. Los bits más bajos al ser entradas no se ocupan y se le asigna el cero.

Acción del LED	Puerto D								Valor Hexadecimal
	7	6	5	4	3	2	1	0	
Rojo	0	1	1	0	0				0x60
Verde	0	1	0	1					0x50
Azul	0	0	1	1					0x30
Apagado	0	1	1	1					0x70

Como queremos que el ánodo del LED se quede encendido únicamente deberemos de asignarle el valor de 1.

Puerto C								Valor Hexadecimal
7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	0x20

Para configurar cada uno de los puertos del microcontrolador podemos implementar la instrucción **TRISXbits.TRISX# = 1/0;**
Donde:

X. - Puerto que se desea configurar.
#. - Bit del puerto a configurar.
1/0.- Si se requiere como entrada o salida, se le asigna 1 o 0.

Por ejemplo: Se desea configurar el puerto D0 como entrada, y el puerto A3 como salida
TRISDbits.TRISD0 = 1;

TRISAbits.TRISA3 = 0;

Mientras que para leer un puerto en específico se sigue la misma lógica de la instrucción anterior, pero ahora se implementa PORT en lugar de TRIS. Podemos utilizar tambien el PORT para modificar la salida de nuestros puertos.

PORTXbits.PORTX# = 1/0;

Donde:

X. - Puerto que se desea configurar.

#. - Bit del puerto a configurar.

1/0.- Si se requiere un nivel lógico alto o bajo debemos de asignarle el valor 1 o 0.

Por ejemplo:

- Si se requiere leer el puerto D0 realizamos lo siguiente

```
if (PORTDbits.PORTD0 == 1){
        Si el puerto D0 es igual a un nivel lógico alto (1)
         agregamos las instrucciones necesarias
}
```

  Usted puede colocar las condiciones de lectura si es alto o bajo.

- Si quiere cambiar el estado del puerto de salida realizamos lo siguiente
  - Para una salida con nivel lógico en alto

```
PORTAbits.PORTA3 = 1;
```

  - Para una salida con nivel lógico bajo

```
PORTAbits.PORTA3 = 0;
```

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h" // Llamar cabecera de bits de configuración

void main(void) {

    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como entradas, 2-5 como salidas
    TRISE = 0X00;          //Todo como salidas
    /*Configuración PORT D*/
    TRISDbits.TRISD0 = 0;   // Salida ResDigit 4
    TRISDbits.TRISD1 = 1;   // Entrada Push 4
    TRISDbits.TRISD2 = 1;   // Entrada Push 3
    TRISDbits.TRISD3 = 1;   // Entrada Push 1
    TRISDbits.TRISD4 = 0;   // Salida LED Rojo
    TRISDbits.TRISD5 = 0;   // Salida LED Verde
    TRISDbits.TRISD6 = 0;   // Salida LED Azul
    TRISDbits.TRISD7 = 0;   // Salida Control LCD DB4
    /*Configuración PORT C*/
    TRISCbits.TRISC0 = 0;   // Salida ResSegmento E
    TRISCbits.TRISC1 = 0;   // Salida ResSegmento F
    TRISCbits.TRISC2 = 0;   // Salida ResSegmento G
    TRISCbits.TRISC3 = 0;   // Salida ResSegmento Punto
    TRISCbits.TRISC4 = 1;   // Entrada Push 2
    TRISCbits.TRISC5 = 0;   // Salida Ánodo RGB
    TRISCbits.TRISC6 = 0;   // RX colocar en bajo
    TRISCbits.TRISC7 = 0;   // TC colocar en bajo
```

```
    // Apagamos todos los puertos
    PORTA = 0x00;
    PORTB = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTE = 0x00;

    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo

    while (1){
        if(PORTDbits.RD3 == 1){     //Si el Push1 es activado encendemos LED Rojo
            PORTDbits.RD4 = 0;  // LED rojo ON
            PORTDbits.RD5 = 1;  // LED verde OFF
            PORTDbits.RD6 = 1;  // LED azul OFF
            PORTCbits.RC5 = 1;  // Ánodo ON
        }
        if(PORTCbits.RC4 == 0){     //Si el Push2 es activado encendemos LED Verde
            PORTDbits.RD4 = 1;
            PORTDbits.RD5 = 0;
            PORTDbits.RD6 = 1;
            PORTCbits.RC5 = 1;
        }
        if(PORTDbits.RD2 == 1){     //Si el Push3 es activado encendemos LED Azul
            PORTDbits.RD4 = 1;
            PORTDbits.RD5 = 1;
            PORTDbits.RD6 = 0;
            PORTCbits.RC5 = 1;
        }
        if(PORTDbits.RD1 == 1){     //Si el Push4 es activado Apagamos el sistema
            PORTDbits.RD4 = 1;
            PORTDbits.RD5 = 1;
            PORTDbits.RD6 = 1;
            PORTCbits.RC5 = 0;
        }

    }
}
```

Notas:

6.1.2 Encendido con retardo (delay)

El microcontrolador puede generar retardos de tiempo esto mediante código o enviar al microcontrolador en un estado de reposo, lo ideal es que no detengamos el proceso del microcontrolador y que el tiempo sea programado por medio de un Timer, pero este proceso se explicara más adelante.
Por el momento comenzaremos con los retardos internos del microcontrolador para ello usamos la función (delay) la cual puede ser de milisegundos o segundos, según sea la aplicación.

En este caso realizaremos dos ejemplos, uno donde el LED este encendido por 500ms y se apague por 500 ms, esto al oprimir un pulsador y se desactivará después de oprimir otro pulsador.
El otro ejemplo es generar una señal de emergencia SOS.

En el primer ejemplo se desea leer el PUSH 1, cuando se active el pulsador comenzara con las instrucciones para encender el LED rojo por un tiempo de 500 ms, seguido de estar apagado por otros 500 ms, esto se repetirá continuamente y se detendrá en el momento en que sea presionado el PUSH 4.

Código fuente

```
#include <xc.h>
#include "configuracion_bits.h" // Llamar cabecera de bits de configuración

void main(void) {
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como entradas, 2-5 como salidas
    TRISE = 0X00;          //Todo como salidas
    /*Configuración PORT D*/
    TRISDbits.TRISD0 = 0;   // Salida ResDigit 4
    TRISDbits.TRISD1 = 1;   // Entrada Push 4
    TRISDbits.TRISD2 = 1;   // Entrada Push 3
    TRISDbits.TRISD3 = 1;   // Entrada Push 1
    TRISDbits.TRISD4 = 0;   // Salida LED Rojo
    TRISDbits.TRISD5 = 0;   // Salida LED Verde
    TRISDbits.TRISD6 = 0;   // Salida LED Azul
    TRISDbits.TRISD7 = 0;   // Salida Control LCD DB4
    /*Configuración PORT C*/
    TRISCbits.TRISC0 = 0;   // Salida ResSegmento E
    TRISCbits.TRISC1 = 0;   // Salida ResSegmento F
    TRISCbits.TRISC2 = 0;   // Salida ResSegmento G
    TRISCbits.TRISC3 = 0;   // Salida ResSegmento Punto
    TRISCbits.TRISC4 = 1;   // Entrada Push 2
    TRISCbits.TRISC5 = 0;   // Salida Ánodo RGB
    TRISCbits.TRISC6 = 0;   // RX colocar en bajo
    TRISCbits.TRISC7 = 0;   // TC colocar en bajo
    // Apagamos todos los puertos
    PORTA = 0x00;
    PORTB = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTE = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo

    unsigned char Estado = 0;
    unsigned char ON = 0;
    while(1){
        if(PORTDbits.RD3 == 1 || Estado == 1){     //Si el Push1 es activado encendemos LED Rojo o si el Estado esta activado
```

```
            Estado = 1;                     //Estado activado
        }
        if(PORTDbits.RD1 == 1){     //Si el Push4 es activado apagamos el sistema
            PORTDbits.RD4 = 1;
            PORTDbits.RD5 = 1;
            PORTDbits.RD6 = 1;
            PORTCbits.RC5 = 0;
            Estado = 0;                     //Estado desactivado
        }
        if (Estado == 1 && ON == 0){
            PORTDbits.RD4 = ~PORTDbits.RD4; //Cambiamos de estado el LED Rojo
            PORTDbits.RD5 = 1;
            PORTDbits.RD6 = 1;
            PORTCbits.RC5 = 1;
            ON = 1;
            __delay_ms(500);
        }
        if(PORTDbits.RD1 == 1){     //Si el Push4 es activado apagamos el sistema
            PORTDbits.RD4 = 1;
            PORTDbits.RD5 = 1;
            PORTDbits.RD6 = 1;
            PORTCbits.RC5 = 0;
            Estado = 0;                     //Estado desactivado
        }
        if (Estado == 1 && ON == 1){
            PORTDbits.RD4 = ~PORTDbits.RD4; //Cambiamos de estado el LED Rojo
            PORTDbits.RD5 = 1;
            PORTDbits.RD6 = 1;
            PORTCbits.RC5 = 1;
            ON = 0;
            __delay_ms(500);
        }

    }
    return;
}
```

El segundo ejemplo será generar una señal de emergencia SOS en el código morse, se activará al oprimir un pulsador y se desactivará cuando se oprima otro botón.
SOS código morse (. . . - - - . . .), la salida será por el LED rojo RD4, el mensaje se activará con el pulsador 1 (RD3) y se apagará con el pulsador 4 (RD0).

- Pulso corto: 200 ms
- Pulso largo (raya): 500 ms
- Espacio entre pulsos: 200 ms
- Pausa entre dos señales SOS: 1000ms

En este ejemplo mandaremos a llamar funciones fuera del main, estas funciones no ayudan a ordenar la actividades o procesos que requiera nuestro proyecto. Al usar funciones externas nos permite tener una estética en el main, y si estas tareas se repitan solo se llame la función en lugar de repetir el mismo proceso en el main.
Podemos tener archivos de cabecera, como librerías, pero eso es un tema que se retomara más adelante.
Código fuente:

```
#include <xc.h>
#include <stdio.h>
#include "configuracion_bits.h" // Llamar cabecera de bits de configuración

int Estado = 0;
void punto (void);              //Tarea de la función punto
```

```
void raya (void);                //Tarea de la función raya
void pausa (void);               //Tarea de la función pausa
void consultarDetener (void);

void main(void) {
   ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
   TRISA = 0X03;           //Se configura 0-1 como entradas, 2-5 como salidas
   TRISE = 0X00;           //Todo como salidas
   /*Configuración PORT D*/
   TRISDbits.TRISD0 = 0;   // Salida ResDigit 4
   TRISDbits.TRISD1 = 1;   // Entrada Push 4
   TRISDbits.TRISD2 = 1;   // Entrada Push 3
   TRISDbits.TRISD3 = 1;   // Entrada Push 1
   TRISDbits.TRISD4 = 0;   // Salida LED Rojo
   TRISDbits.TRISD5 = 0;   // Salida LED Verde
   TRISDbits.TRISD6 = 0;   // Salida LED Azul
   TRISDbits.TRISD7 = 0;   // Salida Control LCD DB4
   /*Configuración PORT C*/
   TRISCbits.TRISC0 = 0;   // Salida ResSegmento E
   TRISCbits.TRISC1 = 0;   // Salida ResSegmento F
   TRISCbits.TRISC2 = 0;   // Salida ResSegmento G
   TRISCbits.TRISC3 = 0;   // Salida ResSegmento Punto
   TRISCbits.TRISC4 = 1;   // Entrada Push 2
   TRISCbits.TRISC5 = 0;   // Salida Ánodo RGB
   TRISCbits.TRISC6 = 0;   // RX colocar en bajo
   TRISCbits.TRISC7 = 0;   // TC colocar en bajo
   // Apagamos todos los puertos
   PORTA = 0x00;
   PORTB = 0x00;
   PORTC = 0x00;
   PORTD = 0x00;
   PORTE = 0x00;
   PORTDbits.RD4 = 1;      // Apagar LED Rojo
   PORTDbits.RD5 = 1;      // Apagar LED Verde
   PORTDbits.RD6 = 1;      // Apagar LED Azul
   PORTCbits.RC5 = 0;      // Apagar ánodo
   while(1){
      if(PORTDbits.RD3 == 1 || Estado == 1){     //Si el Push1 es activado encendemos LED Rojo
         Estado = 1;
         punto ();
         consultarDetener ();
         punto ();
         consultarDetener ();
         punto ();
         consultarDetener ();
         raya ();
         consultarDetener ();
         raya ();
         consultarDetener ();
         raya ();
         consultarDetener ();
         punto ();
         consultarDetener ();
         punto ();
         consultarDetener ();
         punto ();
         consultarDetener ();
         pausa ();
         consultarDetener ();
      }
   }
   return;
}

void consultarDetener(void){
   if(PORTDbits.RD1 == 1){     //Si el Push4 es activado Apagamos el sistema
      PORTD = 0x70;
```

```
        PORTC = 0x00;
        Estado = 0;
    }
}
void punto (void){
    if (Estado == 1){
        PORTD = 0x70;
        PORTC = 0x00;
        __delay_ms(200);
        PORTD = 0x60;
        PORTC = 0x20;
        __delay_ms(200);
        if(PORTDbits.RD1 == 1){
            PORTD = 0x70;
            PORTC = 0x00;
            Estado = 0;
        }
    }
    return;
}
void raya (void){
    if (Estado == 1){
        PORTD = 0x70;
        PORTC = 0x00;
        __delay_ms(200);
        PORTD = 0x60;
        PORTC = 0x20;
        __delay_ms(500);
        if(PORTDbits.RD1 == 1){
            PORTD = 0x70;
            PORTC = 0x00;
            Estado = 0;
        }
    }
    return;
}

void pausa (void){
    if (Estado == 1){
        PORTD = 0x70;
        PORTC = 0x00;
        __delay_ms(1000);
        if(PORTDbits.RD1 == 1){
            PORTD = 0x70;
            PORTC = 0x00;
            Estado = 0;
        }
    }
    return;
}
```

Notas:

6.2 Display de 7 segmentos

Los visualizadores de siete segmentos (llamados display) es una forma de representar caracteres, estos display cuentan un arreglo de LED. Si se utiliza un display con un solo digito tendrá el aspecto como el de la imagen siguiente, sin importar tienen dos puntos como común (COM) ya sea para ánodo o cátodo.

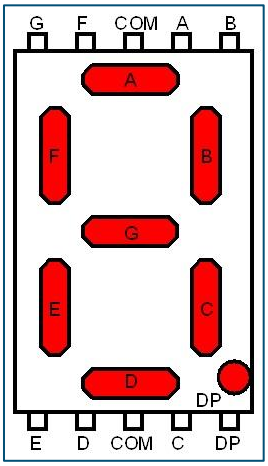


Imagen 6-3 Configuración básica

Existen dos configuraciones para los display, los de ánodo común y los de cátodo común, según estén conectados entre sí los diodos.
En la siguiente imagen se muestra un display de 7 segmentos de un digito, con las dos configuraciones.

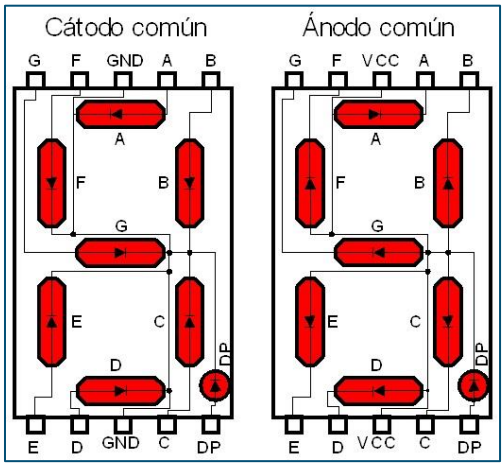


Imagen 6-4 Tipos de display

En la imagen siguiente se muestra la estructura del display implementado en esta tarjeta. [1]

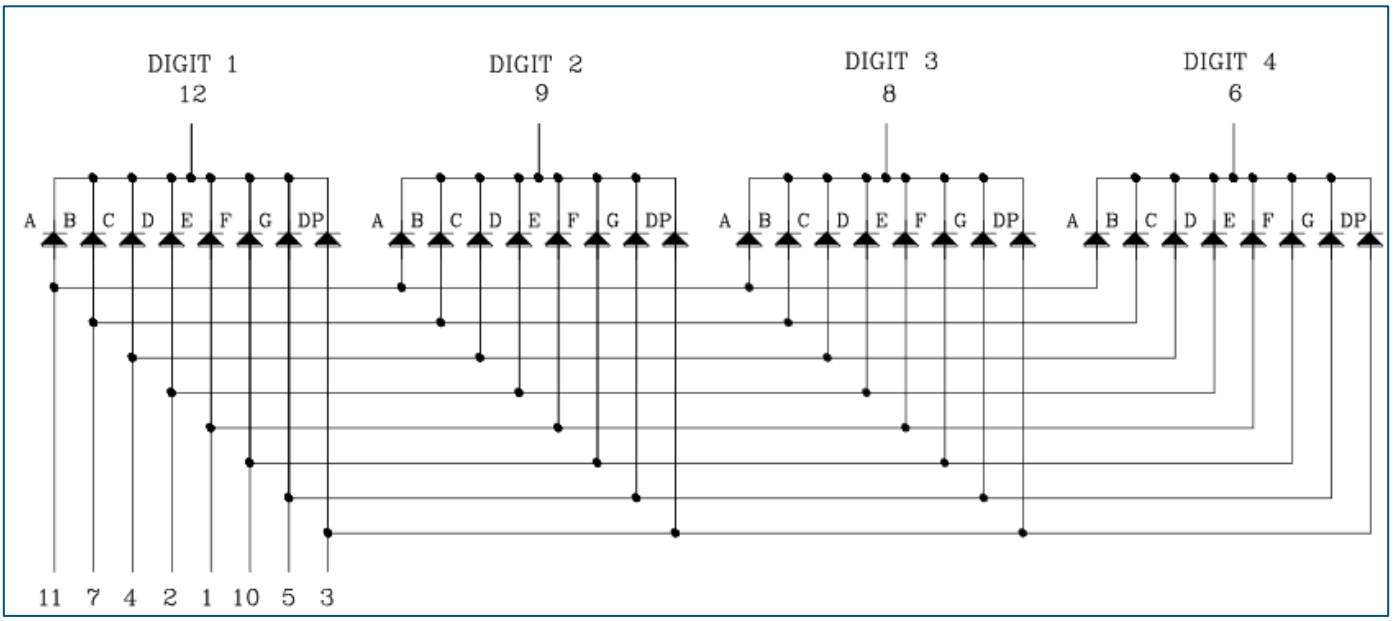


Imagen 6-5 Diagrama interno display

En la siguiente imagen se muestra la distribución de los pines, y el número del digito correspondiente.

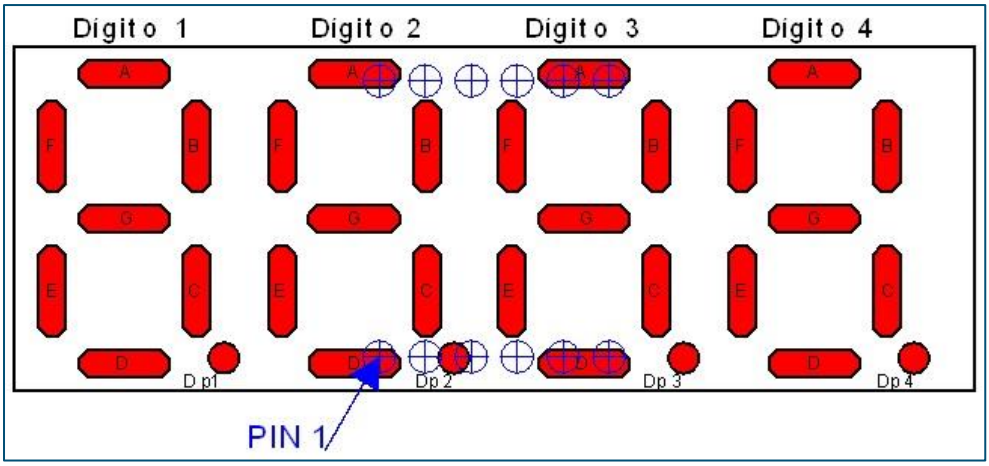


Imagen 6-6 Display 4 dígitos

[1] https://optoelectronics.liteon.com/upload/download/DS-30-96-124/C5723HR.pdf

Las salidas del microcontrolador correspondientes para cada segmento están mostradas en la siguiente imagen.

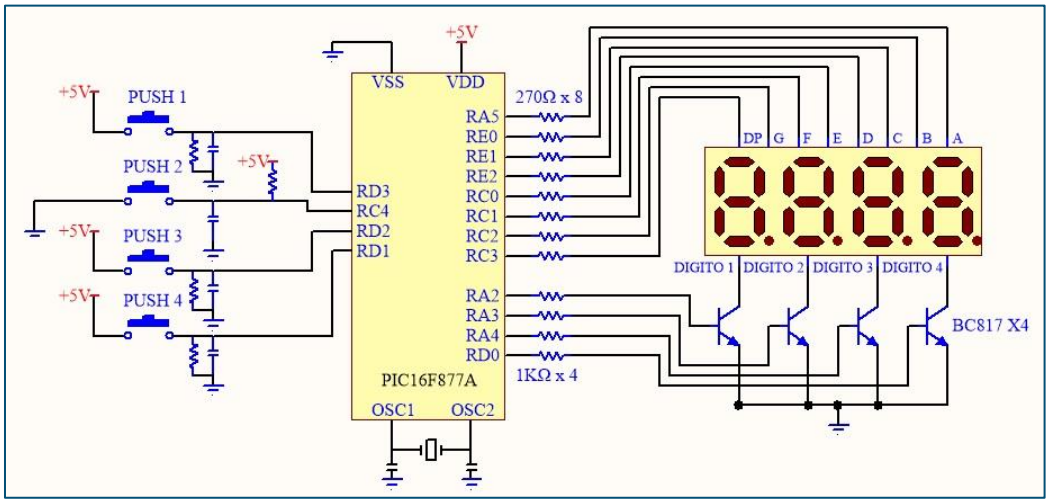


Imagen 6-7 Diagrama de conexión display

Para identificar de mejor manera podemos revisar la siguiente tabla donde se marcan las salidas

Tabla 6-1 Salidas

Display	Puerto
Segmento A	RA5
Segmento B	RE0
Segmento C	RE1
Segmento D	RE2
Segmento E	RC0
Segmento F	RC1
Segmento G	RC2
Segmento DP	RC3
Digito 1	RA2
Digito 2	RA3
Digito 3	RA4
Digito 4	RD0

Con los datos de la tabla anterior, podemos saber que pines serán de salida, ahora vamos a realizar la tabla para cada puerto, de este modo configuramos las entradas y las salidas de cada puerto.

Imagen 6-8 Configuración de puertos

Puerto A									
Puerto	7	6	5	4	3	2	1	0	Valor
Aplicación	x	x	Seg. A	Digito 3	Digito 2	Digito1	Temp.	Potenciómetro	0x03
I/O	x	x	0	0	0	0	1	1	

Puerto C									
Puerto	7	6	5	4	3	2	1	0	Valor

Aplicación	Tx	Rx	Ánodo	Push 2	Seg. D.P.	Seg. G	Seg. F	Seg. E	0x10
I/O	0	0	0	1	0	0	0	0	

Puerto D									
Puerto	7	6	5	4	3	2	1	0	Valor
Aplicación	DB4	LED Azul	LED Verde	LED Rojo	Push 1	Push 3	Push 4	Digito 4	0x0E
I/O	0	0	0	0	1	1	1	0	

Puerto E									
Puerto	7	6	5	4	3	2	1	0	Valor
Aplicación	x	x	x	x	x	Seg. D	Seg. C	Seg. B	0x00
I/O						0	0	0	

Conociendo todas las salidas ahora es necesario crear una tabla donde se muestre las combinaciones para mostrar los caracteres en el display, usaremos como ejemplo la siguiente tabla. Puede que existan más combinaciones para más caracteres, pero, solo mostraremos valores del 0 al 16 en hexadecimal, lo cual sería del 0 al F.

Tabla 6-2 Código 7 segmentos 0 a F

	RC3	RC2	RC1	RC0	RE2	RE1	RE0	RA5	Valor HEX	Port C	Port E	Port A
Hex	D.P.	G	F	E	D	C	B	A				
0	0	0	1	1	1	1	1	1	0x3F	0x03	0x07	0x20
1	0	0	0	0	0	1	1	0	0x06	0x00	0x03	0x00
2	0	1	0	1	1	0	1	1	0x5B	0x05	0x05	0x20
3	0	1	0	0	1	1	1	1	0x4F	0x04	0x07	0x20
4	0	1	1	0	0	1	1	0	0x66	0x06	0x03	0x00
5	0	1	1	0	1	1	0	1	0x6D	0x06	0x06	0x20
6	0	1	1	1	1	1	0	1	0x7D	0x07	0x06	0x20
7	0	0	0	0	0	1	1	1	0x07	0x00	0x03	0x20
8	0	1	1	1	1	1	1	1	0x7F	0x07	0x07	0x20
9	0	1	1	0	0	1	1	1	0x67	0x06	0x03	0x20
A	0	1	1	1	0	1	1	1	0x77	0x07	0x03	0x20
B	0	1	1	1	1	1	0	0	0x7C	0x07	0x06	0x00
C	0	0	1	1	1	0	0	1	0x39	0x03	0x04	0x20
D	0	1	0	1	1	1	1	0	0x5E	0x05	0x07	0x00
E	0	1	1	1	1	0	0	1	0x79	0x07	0x04	0x20
F	0	1	1	1	0	0	0	1	0x71	0x07	0x00	0x20

Para encender cada uno de los dígitos, es necesario activar el pin correspondiente, el digito uno será el que está más a la izquierda, mientras que el digito 4 es el que estará más a la derecha.
Para mostrar cualquier carácter en un solo digito, se debe de activar el deseado y apagar los demás, de lo contrario los 4 dígitos mostraran el mismo carácter. Es por esta razón que el capítulo se ha divido en 2 subtemas, uno para el control de un solo digito, y uno para controlar los 4 dígitos.

6.2.1 Control de 1 display

El manejo de la visualización de caracteres en display de 7 segmentos es de muy gran importancia, debido a que cuando se cuenta con más de 1 digito se puede tener conflicto al mostrar los caracteres. Esto porque no se apaga correctamente los dígitos que no se desean ocupar, y es posible ver otros caracteres.

6.2.1.1 Usar digito 4

Realizar un contador de 0 a 16 en decimal (0-F hex) utilizando el Push 1 para incrementar y el Push 4 para decrementar. Cuando el contador supere el valor 16(F) se reiniciará el valor a 0, si el contador es mayor a cero se podrá reducir el valor, pero si llega a ser igual a 0 no podremos reducir dicho valor. El valor se verá reflejado en el digito 4.

Código fuente:

```
#include <xc.h>
#include <stdio.h>
#include "configuracion_bits.h"

unsigned char codigoPORTA[] = {0x20,0x00,0x20,0x20,0x00,0x20,0x20,0x20,0x20,0x20,0x20,0x00,0x20,0x00,0x20,0x20};
unsigned char codigoPORTC[] = {0x03,0x00,0x05,0x04,0x06,0x06,0x07,0x00,0x07,0x06,0x07,0x07,0x03,0x05,0x07,0x07};
unsigned char codigoPORTE[] = {0x07,0x03,0x05,0x07,0x03,0x06,0x06,0x03,0x07,0x03,0x03,0x06,0x04,0x07,0x04,0x00};

#define DIGITO_1    0x04
#define DIGITO_2    0x08
#define DIGITO_3    0x10
#define DIGITO_4    0x01

int contador = 0;
unsigned char flagPush1 = 0;
unsigned char flahPush4 = 0;
void mostrarNumero (void);

void main(void) {
    /*Configuración de los puertos
     *
     */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC

    TRISA = 0X03;           //Se configura 0-1 como 1, 2-5 como 0
    TRISC = 0x10;           //Solo como entrada RC4, los demas salidas
    TRISD = 0x0E;           //Entradas RD1-RD3, las demas salidas
    TRISE = 0X00;           //Todo como salidas

    //Todos los puertos en alto
    PORTA = 0xFF;
    PORTB = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    PORTE = 0xFF;
    __delay_ms(1000);       // Esperamos 1000ms = 1 seg

    // Todos los puertos en bajo
    PORTA = 0x00;
    PORTB = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTE = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
```

```
    __delay_ms(1000);       // Esperamos 1000ms = 1 seg

     while(1){
        if(PORTDbits.RD3 == 1 && flagPush1 == 0){   //Si el PUSH 1 es presionado y la bandera es cero
           flagPush1 = 1;              //bandera es igual a 1
           contador ++;                //incrementamos el contador
           if (contador >=16)          //Si el contador es mayor o igual a 0
              contador = 0;            //El contador igualamos a 0
        }
        if (PORTDbits.RD3 == 0){       //Si no se oprime el Push 1
           flagPush1 = 0;              //Bandera igual a 0
        }

        if(PORTDbits.RD1 == 1 && flahPush4 == 0){   //Si el Pushb 4 es pulsado y la bandera es cero
           flahPush4 = 1;              //La bandera igual a 1
           if(contador >0)             //Si el contador es mayor a cero
              contador --;             //Decrementamos el contador
        }
        if (PORTDbits.RD1 == 0){       //Si no se pulsa el Push 4
           flahPush4 = 0;              //La bandera igual a 0
        }
        mostrarNumero();               //Llamamos la funcion
    }
}

void mostrarNumero (void){
    PORTA = codigoPORTA[contador];     //Mostrar por el PuertoA el valor del contador
    PORTC = codigoPORTC [contador];    //Mostrar por el PuertoC el valor del contador
    PORTE = codigoPORTE [contador];    //Mostrar por el PuertoE el valor del contador
    PORTD = 0x70 + DIGITO_4;
    return;
}
```

Notas:

6.2.1.2 Usar cualquier digito 1 al 3

Ya que hemos realizado el manejo de un solo display, ahora utilizaremos cualquiera de los otros 3 dígitos, esto con el propósito de controlar correctamente las salidas correspondientes para los segmentos y las salidas de los dígitos.

Si bien vimos en la tabla de Salidas, en el puerto A tenemos las salidas para los otros dígitos, es posible que al momento de manipular estas salidas los resultados no sean lo que esperábamos. Para ello solo debemos de sumar el valor correspondiente a la salida para el carácter con el valor correspondiente para el digito a usar.

Ejemplo:

En el código fuente tenemos los siguientes datos

```
#define DIGITO_1    0x04
#define DIGITO_2    0x08
#define DIGITO_3    0x10
#define DIGITO_4    0x01
```

En el ejemplo anterior usábamos el Puerto D para mostrar el carácter en el digito 4, pero ahora deseamos mostrar el valor por otro digito, para ello debemos de hacer lo siguiente.

Al momento de mostrar el valor, como lo tenemos en el ejemplo anterior, le debemos de agregar el valor del Digito a mostrar, de este modo solo se activará el digito que queremos.

- PORTA = ***DIGITO_2*** **+** codigoPORTA[contador]; //Con el cambio

Y si más a delante queremos encender el LED, podemos hacer lo mismo, pero en el puerto D.

Ahora realicemos el proyecto.

Código fuente:

```
#include <xc.h>
#include <stdio.h>
#include "configuracion_bits.h"

unsigned char codigoPORTA[] = {0x20,0x00,0x20,0x20,0x00,0x20,0x20,0x20,0x20,0x20,0x20,0x00,0x20,0x00,0x20,0x20};
unsigned char codigoPORTC[] = {0x03,0x00,0x05,0x04,0x06,0x06,0x07,0x00,0x07,0x06,0x07,0x07,0x03,0x05,0x07,0x07};
unsigned char codigoPORTE[] = {0x07,0x03,0x05,0x07,0x03,0x06,0x06,0x03,0x07,0x03,0x03,0x06,0x04,0x07,0x04,0x00};

#define DIGITO_1    0x04
#define DIGITO_2    0x08
#define DIGITO_3    0x10
#define DIGITO_4    0x01

int contador = 0;
unsigned char flagPush1 = 0;
unsigned char flahPush4 = 0;
void mostrarNumero (void);

void main(void) {
    /*Configuración de los puertos
     *
     */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC

    TRISA = 0X03;           //Se configura 0-1 como 1, 2-5 como 0
    TRISC = 0x10;           //Solo como entrada RC4, los demas salidas
    TRISD = 0x0E;           //Entradas RD1-RD3, las demas salidas
    TRISE = 0X00;           //Todo como salidas
```

```
    //Todos los puertos en alto
    PORTA = 0xFF;
    PORTB = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    PORTE = 0xFF;
    __delay_ms(500);       // Esperamos 500ms

    // Todos los puertos en bajo
    PORTA = 0x00;
    PORTB = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTE = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

     while(1){
       if(PORTDbits.RD3 == 1 && flagPush1 == 0){   //Si el PUSH 1 es presionado y la bandera es cero
          flagPush1 = 1;              //bandera es igual a 1
          contador ++;               //incrementamos el contador
          if (contador >=16)          //Si el contador es mayor o igual a 0
             contador = 0;           //El contador igualamos a 0
       }
       if (PORTDbits.RD3 == 0){       //Si no se oprime el Push 1
          flagPush1 = 0;              //Bandera igual a 0
       }

       if(PORTDbits.RD1 == 1 && flahPush4 == 0){   //Si el Pushb 4 es pulsado y la bandera es cero
          flahPush4 = 1;              //La bandera igual a 1
          if(contador >0)            //Si el contador es mayor a cero
             contador --;            //Decrementamos el contador
       }
       if (PORTDbits.RD1 == 0){       //Si no se pulsa el Push 4
          flahPush4 = 0;              //La bandera igual a 0
       }
       mostrarNumero();               //Llamamos la funcion
    }
    return;
}

void mostrarNumero (void){
    PORTA = codigoPORTA[contador] + DIGITO_1;      //Mostrar por el PuertoA el valor del contador y activar Digito 1
    PORTC = codigoPORTC [contador];     //Mostrar por el PuertoC el valor del contador
    PORTE = codigoPORTE [contador];     //Mostrar por el PuertoE el valor del contador
    return;
}
```

Realizar los cambios para mostrar el valor en el digito 2 y 3.

Notas:

6.2.2 Control con 4 display

Hasta el momento ya conocemos el manejo de los delay, lectura de puertos, y manejo de salidas. Hemos manejado un solo digito del display, así que es el momento de controlar los cuatro dígitos, independientemente.

Debemos de activar cada uno de los dígitos a una frecuencia la cual sea visible para el ojo humano, esta frecuencia debe de ser entre los 40 Hz y 200 Hz. Cada elemento debe de estar seleccionado una cuarta parte del del tiempo, si consideramos la frecuencia de 50 Hz tenemos un tiempo de 20ms, ahora la cuarta parte es de 5ms, por lo tanto, cada digito debe de estar activado por 5ms.

Mostrar en los 4 display el valor 7853

Código fuente:

```
#include <xc.h>
#include <stdio.h>
#include "configuracion_bits.h"

unsigned char codigoPORTA[] = {0x20,0x00,0x20,0x20,0x00,0x20,0x20,0x20,0x20,0x20,0x20,0x00,0x20,0x00,0x20,0x20};
unsigned char codigoPORTC[] = {0x03,0x00,0x05,0x04,0x06,0x06,0x07,0x00,0x07,0x06,0x07,0x07,0x03,0x05,0x07,0x07};
unsigned char codigoPORTE[] = {0x07,0x03,0x05,0x07,0x03,0x06,0x06,0x03,0x07,0x03,0x03,0x06,0x04,0x07,0x04,0x00};

#define DIGITO_1    0x04
#define DIGITO_2    0x08
#define DIGITO_3    0x10
#define DIGITO_4    0x01

unsigned char flagPush1 = 0;
unsigned char flahPush4 = 0;

int numero = 0;
int millares = 0;
int centenas = 0;
int decenas = 0;
int unidades = 0;
void mostrarNumero(void);

void mostrarUnidades(void);
void mostrarDecenas(void);
void mostrarCentenas(void);
void mostrarMillares(void);

int tiempo = 5;

void main(void) {
    /*Configuración de los puertos
     *
     */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC

    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISC = 0x10;          //Solo como entrada RC4, los demas salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demas salidas
    TRISE = 0X00;          //Todo como salidas

    //Todos los puertos en alto
    PORTA = 0xFF;
    PORTB = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
```

```
    PORTE = 0xFF;
    __delay_ms(500);       // Esperamos 500ms

    // Todos los puertos en bajo
    PORTA = 0x00;
    PORTB = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTE = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    numero = 7853;          //Valor del número a mostrar
    /*
    * Proceso para dividir el número en 4 digitos
    */
    millares = numero / 1000;
    centenas = (numero-(millares * 1000))/100;
    decenas  = (numero- ((millares * 1000) + (centenas * 100)))/10;
    unidades =  numero-((millares * 1000) + (centenas * 100) + (decenas *10));
    while(1){

        mostrarUnidades();      //Mostramos el valor de las unidades
        __delay_ms(tiempo);     //delay entre display
        mostrarDecenas();       //Mostramos el valor de las decenas
        __delay_ms(tiempo);     //delay entre display
        mostrarCentenas();      //Mostramos el valor de las centenas
        __delay_ms(tiempo);     //delay entre display
        mostrarMillares();      //Mostramos el valor de las millares
        __delay_ms(tiempo);     //delay entre display
    }
    return;
}

void mostrarUnidades(void){
    PORTA = codigoPORTA[unidades];      //Mostrar por el PuertoA el valor de las unidades
    PORTC = codigoPORTC [unidades];     //Mostrar por el PuertoC el valor de las unidades
    PORTE = codigoPORTE [unidades];     //Mostrar por el PuertoE el valor de las unidades
    PORTD = 0x70 + DIGITO_4;            // Apagamos el puerto de los LED y activamos digito 4
   return;
}
void mostrarDecenas(void){
    PORTA = codigoPORTA[decenas] + DIGITO_3;      //Mostrar por el PuertoA el valor de las decenas
    PORTC = codigoPORTC [decenas];     //Mostrar por el PuertoC el valor de las decenas
    PORTE = codigoPORTE [decenas];     //Mostrar por el PuertoE el valor de las decenas
    PORTD = 0x70;                      // Apagamos el puerto de los LED y apagamos digito 4
    return;
}
void mostrarCentenas(void){
    PORTA = codigoPORTA[centenas] + DIGITO_2;      //Mostrar por el PuertoA el valor de las centenas
    PORTC = codigoPORTC [centenas];     //Mostrar por el PuertoC el valor de las centenas
    PORTE = codigoPORTE [centenas];     //Mostrar por el PuertoE el valor de las centenas
    PORTD = 0x70;                      // Apagamos el puerto de los LED y apagamos digito 4
    return;
}
void mostrarMillares(void){
    PORTA = codigoPORTA[millares] + DIGITO_1;      //Mostrar por el PuertoA el valor de los millares
    PORTC = codigoPORTC [millares];     //Mostrar por el PuertoC el valor de los millares
    PORTE = codigoPORTE [millares];     //Mostrar por el PuertoE el valor de los millares
    PORTD = 0x70;                      // Apagamos el puerto de los LED y apagamos digito 4
    return;
}
```

Puedes cambiar el valor del número y se deberá de mostrar en los 4 segmentos.

Notas:

6.2.3 Contador de 0 a 9999

Ya con lo aprendido en los capítulos anteriores y tomando la base del código de capitulo anterior, realizaremos el programa para que se muestre en el display los valores del 0 al 9999.

Debemos de tener en cuenta que mientras se esté usando los delay, el microcontrolador quedara en un estado de pausa, por lo que no podremos usar un delay para esperar un segundo y que después lo muestre. Lo recomendable es hacer operaciones para conocer exactamente cuánto tiempo a transcurrido con el valor del tiempo que se espera en los delay.

En otras palabras, hacemos lo siguiente:

Tenemos el valor de tiempo que será igual a 5 ms, este tiempo se espera entre cada activación de digito, por lo que al final de activar los 4 dígitos se habrá tomado un tiempo de 5ms x 4 = 20ms. Los 20ms es el tiempo que tarda en dar una vuelta, pero ahora queremos conocer cuantas vueltas serán necesarias para que transcurra un segundo, así que será dividir el segundo entre el tiempo por vuelta 1000ms/20ms = 50, son 50 vueltas y serán el segundo transcurrido. Por cada segundo transcurrido aumentaremos el valor del número, y así se aumentará hasta llegar a los 9999 segundos.
Podemos detener el contador si deseamos a 60 o cualquier otro valor.

En este ejemplo si se modifica el valor del tiempo se ajustará automáticamente para que se calculen las vueltas necesarias para el valor de un segundo.

Código fuente:

```
#include <xc.h>
#include <stdio.h>
#include "configuracion_bits.h"

unsigned char codigoPORTA[] = {0x20,0x00,0x20,0x20,0x00,0x20,0x20,0x20,0x20,0x20,0x20,0x00,0x20,0x00,0x20,0x20};
unsigned char codigoPORTC[] = {0x03,0x00,0x05,0x04,0x06,0x06,0x07,0x00,0x07,0x06,0x07,0x07,0x03,0x05,0x07,0x07};
unsigned char codigoPORTE[] = {0x07,0x03,0x05,0x07,0x03,0x06,0x06,0x03,0x07,0x03,0x03,0x06,0x04,0x07,0x04,0x00};

#define DIGITO_1    0x04
#define DIGITO_2    0x08
#define DIGITO_3    0x10
#define DIGITO_4    0x01

unsigned char flagPush1 = 0;
unsigned char flahPush4 = 0;

int numero = 0;
int millares = 0;
int centenas = 0;
int decenas = 0;
int unidades = 0;
void mostrarNumero(void);

void mostrarUnidades(void);
void mostrarDecenas(void);
void mostrarCentenas(void);
void mostrarMillares(void);

int tiempo = 5;
```

```
int contador = 0;
int valorMinuto = 0;
int muestras = 0;


void main(void) {
    /*Configuración de los puertos
    *
    */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC

    TRISA = 0X03;           //Se configura 0-1 como 1, 2-5 como 0
    TRISC = 0x10;           //Solo como entrada RC4, los demas salidas
    TRISD = 0x0E;           //Entradas RD1-RD3, las demas salidas
    TRISE = 0X00;           //Todo como salidas

    //Todos los puertos en alto
    PORTA = 0xFF;
    PORTB = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    PORTE = 0xFF;
    __delay_ms(500);       // Esperamos 500ms

    // Todos los puertos en bajo
    PORTA = 0x00;
    PORTB = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTE = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    numero = 0;             //Valor del  numero a mostrar
    valorMinuto = tiempo * 4;   //Obtenemos el valor del tiempo por ciclo
    muestras = (1000 / valorMinuto);    //Obtenemos el valor de muestras para 1 segundo

    while(1){
        /*
         * Proceso para dividir el número en 4 digitos
         */
        millares = numero / 1000;
        centenas = (numero-(millares * 1000))/100;
        decenas  = (numero- (millares * 1000 + centenas * 100))/10;
        unidades =  numero-(millares * 1000 + centenas * 100 + decenas *10);

        mostrarUnidades();      //Mostramos el valor de las unidades
        __delay_ms(tiempo);     //delay entre display
        mostrarDecenas();       //Mostramos el valor de las decenas
        __delay_ms(tiempo);     //delay entre display
        mostrarCentenas();      //Mostramos el valor de las centenas
        __delay_ms(tiempo);     //delay entre display
        mostrarMillares();      //Mostramos el valor de las millares
        __delay_ms(tiempo);     //delay entre display

        contador ++;            //Incrementamos las vueltas del proceso
        if (contador >= muestras){  //Si se cumplen las vueltas a las calculadas
            contador = 0;       //Reiniciamos las vueltas
            numero ++;          //Incrementamos el valor del número
        }
        if (numero >=10000){    //Si llegamos al valor deceado
            contador = 0;       //Reiniciamos las vueltas
            numero = 0;         //Reiniciamos el contador
        }
```

```
    }
    return;
}

void mostrarUnidades(void){
    PORTA = codigoPORTA[unidades];      //Mostrar por el PuertoA el valor de las unidades
    PORTC = codigoPORTC [unidades];     //Mostrar por el PuertoC el valor de las unidades
    PORTE = codigoPORTE [unidades];     //Mostrar por el PuertoE el valor de las unidades
    PORTD = 0x70 + DIGITO_4;            // Apagamos el puerto de los LED y activamos el digito 4
   return;
}
void mostrarDecenas(void){
    PORTA = codigoPORTA[decenas] + DIGITO_3;      //Mostrar por el PuertoA el valor de las decenas
    PORTC = codigoPORTC [decenas];      //Mostrar por el PuertoC el valor de las decenas
    PORTE = codigoPORTE [decenas];      //Mostrar por el PuertoE el valor de las decenas
    PORTD = 0x70;                       // Apagamos el puerto de los LED y apagamos digito 4
    return;
}
void mostrarCentenas(void){
    PORTA = codigoPORTA[centenas] + DIGITO_2;      //Mostrar por el PuertoA el valor de las centenas
    PORTC = codigoPORTC [centenas];     //Mostrar por el PuertoC el valor de las centenas
    PORTE = codigoPORTE [centenas];     //Mostrar por el PuertoE el valor de las centenas
    PORTD = 0x70;                       // Apagamos el puerto de los LED y apagamos digito 4
    return;
}
void mostrarMillares(void){
    PORTA = codigoPORTA[millares] + DIGITO_1;      //Mostrar por el PuertoA el valor de los millares
    PORTC = codigoPORTC [millares];     //Mostrar por el PuertoC el valor de los millares
    PORTE = codigoPORTE [millares];     //Mostrar por el PuertoE el valor de los millares
    PORTD = 0x70;                       // Apagamos el puerto de los LED y apagamos digito 4
    return;
}
```

Notas:

6.2.4 Con punto decimal

Para mostrar valores con punto decimal es necesario activar el segmento correspondiente para el punto, en el display cuenta con 4 puntos decimales, los cuales esta conectados entre sí y este segmento va conectado al puerto RC3. Si queremos que se muestre debemos de sumar al Puerto C el valor del RC3, y al mismo tiempo activar el digito que queremos mostrar el punto.

Por ejemplo:

Si deseamos mostrar el valor de 124.3, el valor que debemos de activa con el punto decimal debe de ser el 4, así que en el código debemos de mostrar el número 4 más el valor correspondiente al punto decimal, y activar el digito 3.

Para tener un mejor control de números con punto decimal lo recomendable es recorrer el punto decimal tantas veces como se dese. Es decir, si tenemos un valor de 124.3 recorremos el punto un lugar por lo tanto lo multiplicamos por diez, así que nos queda 1243. O si tenemos el valor de 12.34 lo podemos multiplicar por diez o cien.

Para activar el punto decimal debemos de mandar un pulso alto por el Puerto RC3, el RC3 toma el valor de 8 en decimal. Tal y como se hizo para activar los dígitos por el puerto A, debemos de sumar el valor del carácter a mostrar más el valor del punto decimal.

Código fuente:

```
#include <xc.h>
#include <stdio.h>
#include "configuracion_bits.h"

unsigned char codigoPORTA[] = {0x20,0x00,0x20,0x20,0x00,0x20,0x20,0x20,0x20,0x20,0x20,0x00,0x20,0x00,0x20,0x20};
unsigned char codigoPORTC[] = {0x03,0x00,0x05,0x04,0x06,0x06,0x07,0x00,0x07,0x06,0x07,0x07,0x03,0x05,0x07,0x07};
unsigned char codigoPORTE[] = {0x07,0x03,0x05,0x07,0x03,0x06,0x06,0x03,0x07,0x03,0x03,0x06,0x04,0x07,0x04,0x00};

#define DIGITO_1    0x04
#define DIGITO_2    0x08
#define DIGITO_3    0x10
#define DIGITO_4    0x01

#define PuntoDecimal 0x08

unsigned char flagPush1 = 0;
unsigned char flahPush4 = 0;

int numero = 0;
int millares = 0;
int centenas = 0;
int decenas = 0;
int unidades = 0;


void mostrarUnidades(void);
void mostrarDecenas(void);
void mostrarCentenas(void);
void mostrarMillares(void);

int tiempo = 5;

void main(void) {
```

```
    /*Configuración de los puertos
    *
    */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC

    TRISA = 0X03;           //Se configura 0-1 como 1, 2-5 como 0
    TRISC = 0x10;           //Solo como entrada RC4, los demas salidas
    TRISD = 0x0E;           //Entradas RD1-RD3, las demas salidas
    TRISE = 0X00;           //Todo como salidas

    //Todos los puertos en alto
    PORTA = 0xFF;
    PORTB = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    PORTE = 0xFF;
    __delay_ms(500);       // Esperamos 500ms

    // Todos los puertos en bajo
    PORTA = 0x00;
    PORTB = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTE = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    numero = (321.3* 10);
    while(1){
        /*
         * Proceso para dividir el número en 4 digitos
         */
        millares = numero / 1000;
        centenas = (numero-(millares * 1000))/100;
        decenas  = (numero- (millares * 1000 + centenas * 100))/10;
        unidades =  numero-(millares * 1000 + centenas * 100 + decenas *10);

        mostrarUnidades();      //Mostramos el valor de las unidades
        __delay_ms(tiempo);     //delay entre display
        mostrarDecenas();       //Mostramos el valor de las decenas
        __delay_ms(tiempo);     //delay entre display
        mostrarCentenas();      //Mostramos el valor de las centenas
        __delay_ms(tiempo);     //delay entre display
        mostrarMillares();      //Mostramos el valor de las millares
        __delay_ms(tiempo);     //delay entre display
    }
    return;
}

void mostrarUnidades(void){
    PORTA = codigoPORTA[unidades];      //Mostrar por el PuertoA el valor de las unidades
    PORTC = codigoPORTC [unidades];     //Mostrar por el PuertoC el valor de las unidades
    PORTE = codigoPORTE [unidades];     //Mostrar por el PuertoE el valor de las unidades
    PORTD = 0x70 + DIGITO_4;            // Apagamos el puerto de los LED y activamos el digito 4
   return;
}
void mostrarDecenas(void){
    PORTA = codigoPORTA[decenas] + DIGITO_3;      //Mostrar por el PuertoA el valor de las decenas
    PORTC = codigoPORTC [decenas] + PuntoDecimal;     //Mostrar por el PuertoC el valor de las decenas
    PORTE = codigoPORTE [decenas];     //Mostrar por el PuertoE el valor de las decenas
    PORTD = 0x70;                       // Apagamos el puerto de los LED y apagamos digito 4
    return;
}
void mostrarCentenas(void){
```

```
    PORTA = codigoPORTA[centenas] + DIGITO_2;      //Mostrar por el PuertoA el valor de las centenas
    PORTC = codigoPORTC [centenas];     //Mostrar por el PuertoC el valor de las centenas
    PORTE = codigoPORTE [centenas];     //Mostrar por el PuertoE el valor de las centenas
    PORTD = 0x70;                       // Apagamos el puerto de los LED y apagamos digito 4
    return;
}
void mostrarMillares(void){
    PORTA = codigoPORTA[millares] + DIGITO_1;      //Mostrar por el PuertoA el valor de los millares
    PORTC = codigoPORTC [millares];     //Mostrar por el PuertoC el valor de los millares
    PORTE = codigoPORTE [millares];     //Mostrar por el PuertoE el valor de los millares
    PORTD = 0x70;                       // Apagamos el puerto de los LED y apagamos digito 4
    return;
}
```

Notas:

6.3 Control de pantalla LCD

El uso de las pantallas LCD (sigla del inglés liquid-crystal display) se ha implementado en varias áreas tanto como en la industria, como en proyectos escolares o de medianas empresas, ya que su uso es agradable.
En el mercado existen una gama de diferentes LCD, dependiendo de la aplicación que se requiera. Para nuestro caso usaremos una LCD de 16x2, esto significa que tenemos 16 caracteres por dos filas, la comunicación la realizaremos en bus de 4 bits, en la siguiente imagen se muestra el diagrama ce conexión entre la pantalla y el microcontrolador.

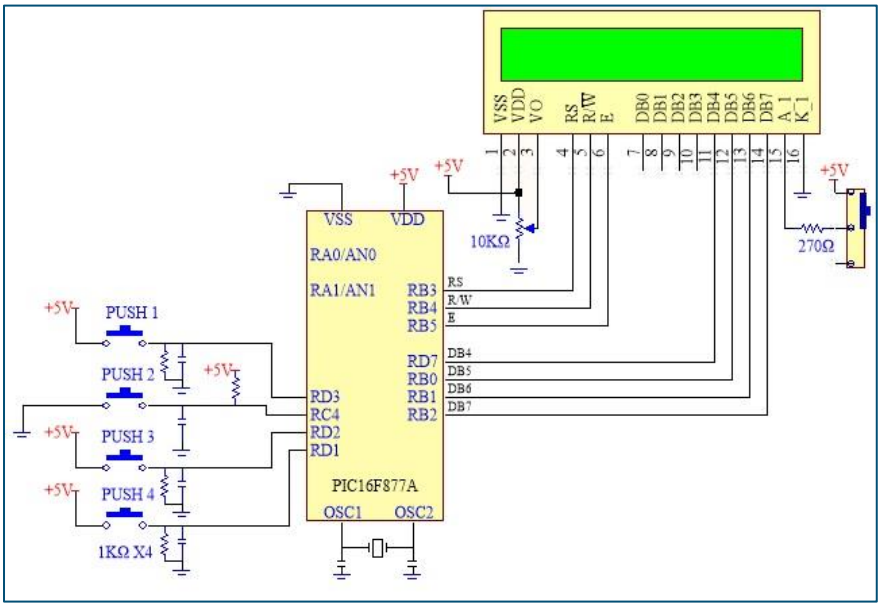


Imagen 6-9 Diagrama display LCD

Como se puede apreciar en el diagrama los puertos usados para el manejo de la pantalla son por los puertos “B” y “D”, identificamos los pines a usar y obtenemos la siguiente tabla.

Tabla 6-3 Salidas LCD

Puerto	Aplicación
RB0	DB5
RB1	DB6
RB2	DB7
RB3	RS
RB4	R/W
RB5	E
RD7	DB4

La descripción de cada uno de los pines son los siguientes[2]

- Bus de datos: desde DB4 a DB7
- Bus de control:
- Enable (E): Cuando está e nivel bajo el display está deshabilitado.
- Read/Write (R/W): Cuando está a nivel bajo los datos se escriben en el LCD. Cuando está en alto, los datos se leen del LCD.
- Register Select (RS): Sirve para distinguir entre las instrucciones y caracteres. A nivel bajo se envían instrucciones y a nivel alto caracteres.

[2] PROGRAMACIÓN DE MICROCONTROLADORES PIC EN LENGUAJE C, Alfaomega, pp 55.

El display contiene tres bloques de memoria:[3]

- DDRAM – Display Data RAM
- CGRAM – Character Generator RAM
- CGROm – Character Generator ROM

Para mandar un carácter al display, se escribe en la DDRAM. La CGROM contiene las matrices de puntos del juego de caracteres por defecto.
También es posible generar nuevos caracteres definidos por el usuario mediante la CGRAM, una memoria de 64 bytes. Cada carácter necesita 8 bytes para almacenarse por lo que disponemos de 8 caracteres definidos por el usuario. Para almacenar el mapa de un carácter, debemos iniciar la dirección de la CGRAM y escribir datos al display.

Antes de acceder a la DDRAM después de la definición del carácter, el programa debe establecer la posición de la DRAM donde escribir el nuevo carácter.

Cuando se envía una instrucción o un dato al display es necesario esperar un cierto tiempo antes de enviar el siguiente, para que el display pueda procesarlo.
Esto se puede hacer monitorizando el flag de Busy o respetando los tiempos de ejecución indicados por el fabricante.

6.3.1 Librería de LCD

Como hemos visto en los ejemplos anteriores la configuración de los puertos la hemos realizado en el mismo main, pero a partir de este capítulo las funciones y los puertos comienzan a aumentar, por lo que es necesario recurrir a la creación de cabeceras con .h y .c, para ver cómo se crean estos archivos ir al siguiente capítulo en el paso VIII.
Necesitamos construir dos archivos .h, uno para la LCD y otro para la configuración de los puertos, adicionalmente un archivo .c para el manejo de la LCD.

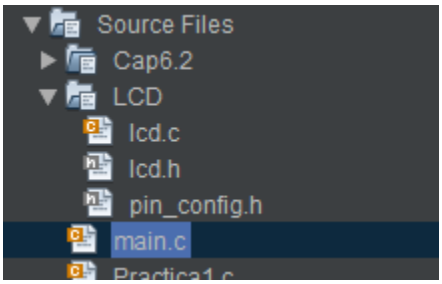


Imagen 6-10 Cabeceras

Puedes obtener los archivos en el siguiente enlace, solo copia el código en tus archivos.
En el pin_config.h encontraras los comandos para la pantalla, donde puedes mover el cursor, apagar el cursor, limpiar la pantalla, etc.

6.3.2 Hola Mundo en LCD

Con las librerías previamente cargadas mostraremos nuestro primer mensaje por la pantalla LCD.

[3] PROGRAMACIÓN DE MICROCONTROLADORES PIC EN LENGUAJE C, Alfaomega, pp 55.

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "lcd.h"
#include "lcd_pin_config.h"


void main(void) {
    /*Configuración de los puertos
    *
    */

    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demas salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demas salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    Lcd_Init();            //Inicializamos la LCD
    delay(5);
    Lcd_Cmd (LCD_CLEAR);    //Limpiamos la pantalla
    delay(5);
    Lcd_Cmd (LCD_CLEAR);    //Limpiamos la pantalla
    Lcd_Cmd (LCD_CURSOR_OFF);     //Si queremos que el cursor se apague mandamos esta instrucción
    Lcd_Out (1,3,"Hola");   //Colocar el cursor en fila 1 columna 3, escribir Hola
    Lcd_Out (2,5, "Mundo"); //Colocar el cursor en fila 2 columna 5, escribir Mundo
    while(1){
    }

    return;
}
```

6.3.3 Reloj

Realizar un reloj que se muestre en el display, para este ejemplo usaremos el delay, solo para fines demostrativos, más adelante implementaremos temporizadores para no detener el proceso del microcontrolador.

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "lcd.h"
#include "lcd_pin_config.h"
#include <stdio.h>
```

```
unsigned char segundos = 0;
unsigned char minutos = 0;
unsigned char horas = 0;
unsigned char buffer[10];


void main(void) {
    /*Configuración de los puertos
    *
    */

    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;           //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;           //Solo como entrada RC4, los demas salidas
    TRISD = 0x0E;           //Entradas RD1-RD3, las demas salidas
    TRISE = 0X00;           //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

        Lcd_Init();                                 //Inicializamos la LCD
    Lcd_Cmd (LCD_CLEAR);                //Limpiamos la pantalla
    Lcd_Cmd (LCD_CURSOR_OFF);    //Apagamos el cursor
    Lcd_Out (1,5,"Hola");                   //Colocar el cursor en fila 1 columna 3, escribir Hola
    Lcd_Out (2,5, "MCU");                   //Colocar el cursor en fila 2 columna 5, escribir Mundo
    __delay_ms(2000);
    Lcd_Cmd (LCD_CLEAR);              //Limpiamos la pantalla
    Lcd_Out(1,5,"Tiempo");
            while(1){
        Lcd_Out(1,5,"Tiempo");
        sprintf(buffer,"%d:%d:%d",horas,minutos,segundos);
        Lcd_Out2(2,3,buffer);
        __delay_ms(1000);
        segundos++;
        if(segundos >= 60){
            Lcd_Cmd (LCD_CLEAR);            //Limpiamos la pantalla
            minutos++;
            segundos = 0;
        }
        if(minutos >=60){
            Lcd_Cmd (LCD_CLEAR);            //Limpiamos la pantalla
            horas++;
            minutos =0;
        }
        if(horas >=24){
            Lcd_Cmd (LCD_CLEAR);            //Limpiamos la pantalla
            horas = 0;
        }
    }
    return;
}
```

Notas:

7 Temporizadores

Los temporizadores en un PIC, junto a su sistema de interrupciones, ayudan a construir un sistema que realizan varias tareas dependientes del tiempo simultaneo.
El temporizador contador funciona como con el reloj del sistema, pero tambien puede ser utilizado para realizar conteos, por lo que también se le llama contador.

Un ejemplo es la implementación de un cronómetro, con los temporizadores se puede obtener la cuenta precisa del tiempo y por otro se puede refrescar los display o las pantallas de manera periódica, sin la necesidad de interrumpir el proceso o de implementar los delay.
Para realizar estas tareas el microcontrolador debe de usar un temporizador que le indique el momento exacto en que debe iniciar cada rutina.

Otras aplicaciones que se le pueden dar a los temporizadores son para crear retardos, realizar tareas periódicas, generar señales PWM o contar pulsos externos, etc. Al tratarse de dispositivos de hardware independientes de la CPU, todas estas funciones tienen lugar de manera simultánea, y a la vez que el micro puede estar haciendo otras tareas.

El PIC16F877A que se ocupa consta de tres temporizadores, Timer0, Timer1 y Timer2.
En la siguiente imagen se muestra el diagrama interno de los temporizadores para el PCI16F877A.
http://ww1.microchip.com/downloads/en/DeviceDoc/39582C.pdf

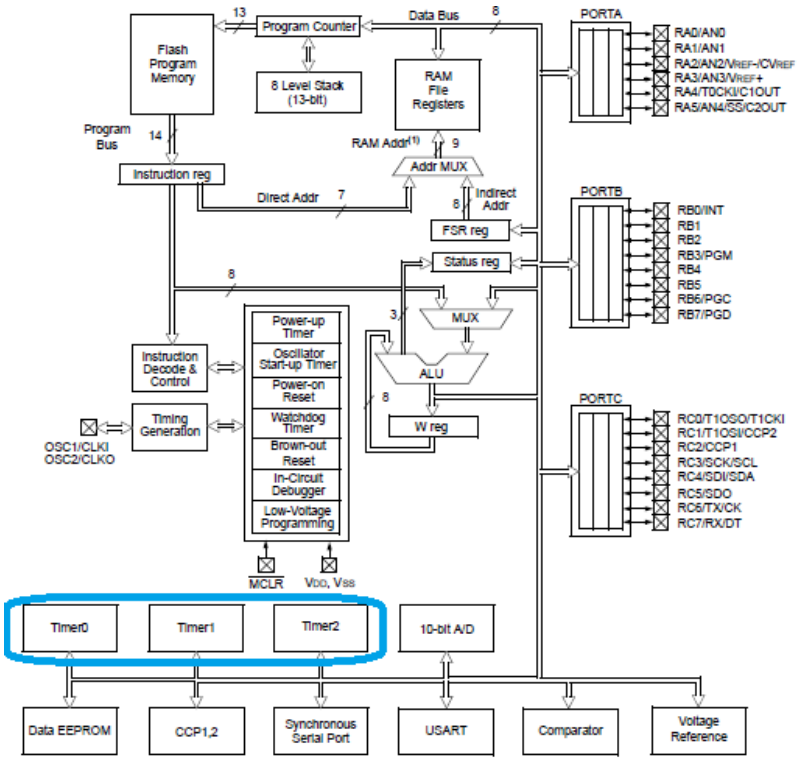


Imagen 7-1 Temporizador

La siguiente tabla proporciona los detalles de los tres temporizadores.

Tabla 7-1 Temporizador

Temporizador	Talla	Registro de control	Cuenta Registro	Retraso mínimo	Retraso máximo
TIMER0	8 bits	OPTION_REG	TMR0	0.2usec	13.107ms
TIMER1	16 bits	T1CON	TMR1H, TMR1L	0.2usec	104,857 ms
TIMER2	8 bits	T2CON	TMR2	0.2usec	819usec

Cálculo del temporizador

La frecuencia del oscilador PIC se divide por 4 y luego se envía al controlador. Ahora, esta frecuencia se puede dividir aún más por Prescaler para generar el rango de retardos. El tiempo para incrementar la cuenta del temporizador en uno (tick del temporizador) se puede determinar cómo se indica a continuación.

tick = (Prescaler / (Fosc / 4)
tick = (Prescaler / (20Mhz / 4))
tick = (Prescaler * 4) / Fosc

Ahora, el valor del temporizador para el retraso requerido se puede calcular como se indica a continuación.

Delay = TimerCount * tick
Count = (Delay / tick)
RegValue = TimerMax- Count
RegValue = TimerMax- (Delay / tick) = TimerMax - (Delay / ((Prescaler * 4) / Fosc))
RegValue = TimerMax - ((Retraso * Fosc) / (Prescaler * 4))

La siguiente tabla proporciona la fórmula para los tres temporizadores.

Temporizador	Talla	Fórmula para el cálculo de la demora
TIMER0	8 bits	**RegValue = 256 - ((Retraso * Fosc) / (Predivisor * 4))**
TIMER1	16 bits	**RegValue = 65536 - ((Retraso * Fosc) / (Predivisor * 4))**
TIMER2	8 bits	**RegValue = 256 - ((Retraso * Fosc) / (Predivisor * 4))**

7.1 Temporizador 0[4]

[5]El módulo TMR0 es un temporizador / contador de 8 bits con las siguientes características:

[4] https://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf
[5] https://exploreembedded.com/wiki/PIC16f877a_Timer

- Temporizador / contador de 8 bits
- Legible y escribible
- Prescaler programable por software de 8 bits
- Selección de reloj interno o externo
- Interrupción por desbordamiento de FFh a 00h
- Selección de borde para reloj externo

Timer0 registros

La siguiente tabla muestra los registros asociados con el módulo PIC16f877A Timer0.

Registrarse	Descripción
OPTION_REG	Estos registros se utilizan para configurar el predivisor TIMER0, la fuente del reloj, etc.
TMR0	Este registro contiene el valor de conteo del temporizador que se incrementará dependiendo de la configuración prescalar
INTCON	Este registro contiene el indicador de desbordamiento del Timer0 (TMR0IF) y el indicador de habilitación de intrusión correspondiente (TMR0IE).

Prescaler

Solo hay un preescalador disponible que secompartido exclusivamente entre el módulo Timer0 y el Temporizador de Watchdog. Una asignación de preescalador para el el módulo Timer0 significa que no hay preescalador para el Watchdog Timer y viceversa. Este preescalador no es legible o escribible.

El PSA y PS2: bits PS0 (OPTION_REG <3: 0>) determinar la asignación de preescalador y la proporción de preescala.

Cuando se asigna al módulo Timer0, todas las instrucciones escribir en el registro TMR0 (por ejemplo, CLRF1, MOVWF 1, BSF 1, x .... etc.) Borrará el preescalador. Cuando se asigna a WDT, una instrucción CLRWDT borrará el preescalador junto con el temporizador de vigilancia. El prescaler no es legible o escribible.

OPTION_REG							
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
7	6	5	4	3	2	1	0
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

RBPU: NA para temporizadores

INTEDG: NA para temporizadores

T0CS: Bit de selección de fuente de reloj TMR0.

1 = Transición en el pin T0CKI

0 = Reloj de ciclo de instrucción interno (CLKO)

T0SE: Bit de selección de borde de origen TMR0

1 = Incremento en la transición de alto a bajo en el pin T0CKI

0 = Incremento en la transición de bajo a alto en el pin T0CKI

PSA: bit de asignación de preescalador

1 = El preescalador está asignado al WDT

0 = El preescalador está asignado al Timer0

PS2: PS0: bits de selección de velocidad de preescalador

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Nota: Solo hay un preescalador disponible que se comparte exclusivamente entre el módulo Timer0 y el temporizador Watchdog. Una asignación de preescalador para el módulo Timer0 significa que no hay preescalador para el temporizador de vigilancia y viceversa. Este preescaler no es accesible, pero se puede configurar usando los bits de **PS2:PS0** de **OPTION_REG.**

7.1.1 Utilizar el temporizador sin interrupción

Realizar el código para establecer el temporizador para apagar y encender un LED cada segundo, este temporizador debe de contar con el prescaler a 256 y el tiempo sea de 10 ms.

Encender cualquier de los LED's del RGB.

Cálculos de retardo para 10ms a 16MHz con prescaler 256:

$$\text{RegValue} = 256 - \left(\frac{Retraso * Fosc}{Prescaler * 4}\right) = 256 - \left(\frac{10ms * 16MHz}{256 * 4}\right) = 256 - 156.25 = 99.75 \approx 100$$

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"

void main(void) {
    /*Configuración de los puertos
    *
    */

    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;           //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;           //Solo como entrada RC4, los demas salidas
    TRISD = 0x0E;           //Entradas RD1-RD3, las demas salidas
    TRISE = 0X00;           //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
```

```
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    unsigned char cont;                         // Contador de desbordamiento de TMR0
    OPTION_REG=0xC7;                            // TMR0 como temporizador con predivisor P=256
    TMR0 = 100;                                 // Para que Td sean 10ms
    while(1){
       if(T0IF){
          T0IF=0;         //Desactivar el FLAG de TMR0
          TMR0=100;       //Recargar TMR0
          cont++;         //Incrementar contador de desbordamiento de TMR0
          if(cont==100){         //Si se han llegado a 100 desbordamiento (1 segundo)
             PORTDbits.RD5 = ~PORTDbits.RD5;     // Cambia estado de RB1
             PORTCbits.RC5 = ~PORTCbits.RC5;    //Cambia Estado del Anodo
             cont=0;     // Reinicia contador de desbordamientos
          }
       }
    }
    return;
}
```

Para validar que nuestro programa proporciona el tiempo de 1 segundo podemos conectar un osciloscopio al pin de salida del microcontrolador ya sea RD5 o RC5, en la tarjeta estará la serigrafia para ubicarlo.

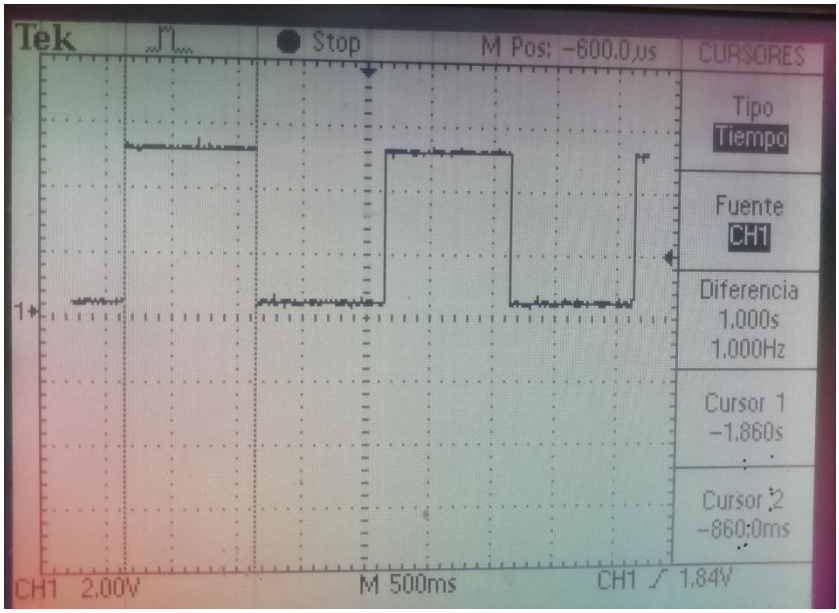


Imagen 7-2 Pulso de 1Hz (1segundo)

7.1.2 Utilizar el temporizador con interrupción

En este modo lo que implica es que en lugar de estar preguntando por el estado del temporizador este se actualizara automáticamente, aunque el microcontrolador este atendiendo otras tareas.
Para explicar el manejo de las interrupciones usaremos el display de 7 segmentos, con un solo digito, mostraremos valores del 0 al 9, reestableciendo cada vez que llegue al 10.
En comparación con el manejo del display del capítulo 6.2 no manejaremos los delay, el conteo será externamente son interrumpir el proceso.

Nota: Para estos ejercicios será necesario agregar las librerías que se van a utilizar, sin estas librerías es posible que el código arroje errores.

Código fuente:

```
/#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "control_7segmentos.h"

// DEFINICIÓN DE VARIABLES
int cont = 0;                           //Contador de desbordamiento de TMR0
unsigned char valorDigito = 0;          //Contador del dígito a mostrar
int timeValue = 252;                    //configuración del Td

void main(void) {
    /*Configuración de los puertos
    *
    */

    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    setup_timer0(RTCC_INTERNAL|RTCC_DIV256);    // TMR0 como temporizador predivisor=256
    set_timer0(240);                                    // Para que Td sean 1ms
    ei();                           //Habilitar interrupciones
    T0IE=1;                         // Habilitar interrupción de TMR0
    mostrarNumero(valorDigito);

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

```
// INTERRUPCIÓN por Timer0
void __interrupt () isr(void){
    if(T0IF){
        T0IF=0;                                         //Desactivar el FLAG de la interrupción de TMR0
        set_timer0(240);                                //Recargar TMR0
        cont++;                                         //Incrementar contador de interrupciones
        if(cont>=1000){
            valorDigito++;                      //Decremento índice lista
            if(valorDigito >= 10){
                valorDigito = 0;                //Siguiente al último de la lista
            }
            mostrarNumero(valorDigito);
            cont=0;                         //Ponemos a cero el contador de interrupciones
        }
    }
}
```

Notas:

7.2 Temporizador 1

El módulo de temporizador TMR1 es un temporizador / contador de 16 bits con las siguientes características:

- Temporizador / contador de 16 bits con dos registros de 8 bits TMR1H / TMR1L
- Legible y escribible
- Prescaler programable por software hasta 1: 8
- Selección de reloj interno o externo
- Interrupción por desbordamiento de FFFFh a 00h
- Selección de borde para reloj externo

Timer1 puede funcionar en uno de dos modos:

- Como temporizador
- Como contador

En el modo temporizador, Timer1 incrementa cada instrucción ciclo. En el modo Contador, aumenta con cada aumento borde de la entrada de reloj externo.

Registros Timer1

La siguiente tabla muestra los registros asociados con el módulo PIC16f877A Timer1.

Tabla 7-2 Registros Timer1

Registro	Descripción
T1CON:	Estos registros se utilizan para configurar el prescaler TIMER1, la fuente del reloj, etc.
TMRIH	Este registro contiene los 8 bits más altos del valor del temporizador. TMR1H y TMR1L se utilizan en pareja para incrementar de 0000 - FFFFh
TMRIL	Este registro contiene los 8 bits más bajos del valor del temporizador. TMR1H y TMR1L se utilizan en pareja para incrementar de 0000 - FFFFh
PIR1	Este registro contiene el indicador de desbordamiento del Timer1 (TMR1IF).
PIE1	Este registro contiene el indicador de habilitación de interrupción del temporizador 1 (TMR1IE).

Tabla 7-3 T 1CON

T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)							
7	6	5	4	3	2	1	0
-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

bit 7-6 **No implementado**: leer como '0'

bit 5-4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits

11 = 1: 8 valor de preescala

10 = 1: 4 valor de preescala

01 = 1: 2 valor de preescala
00 = 1: 1 valor de preescala

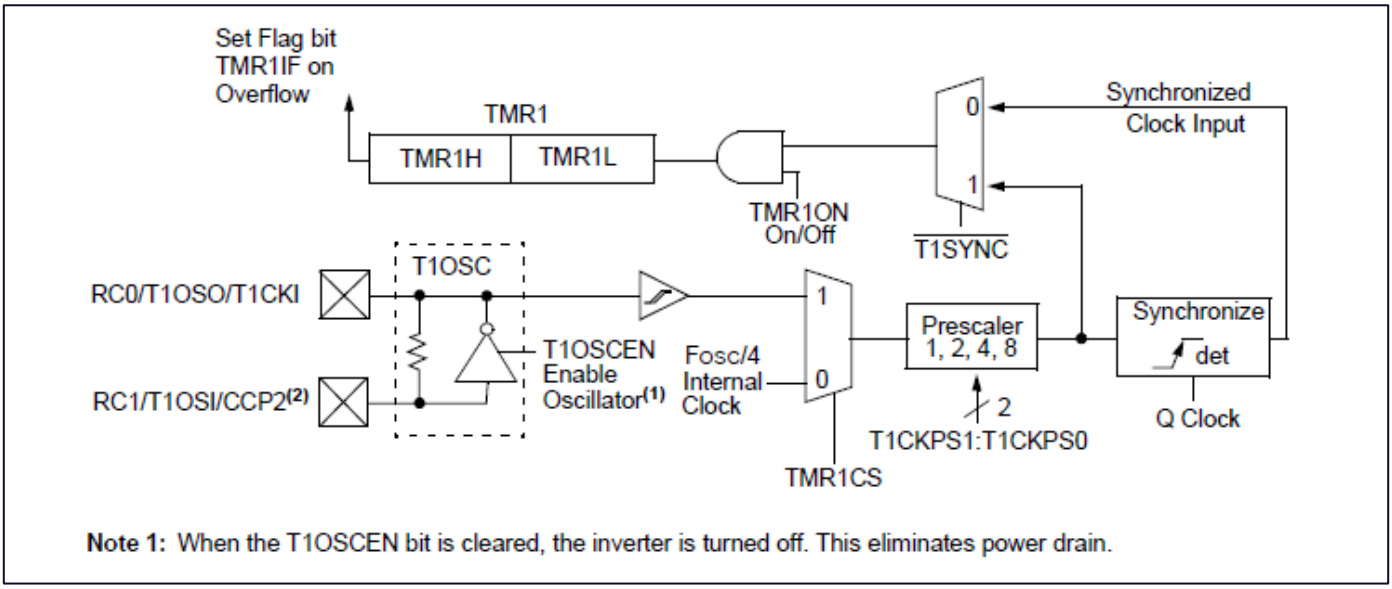
bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = oscilador habilitado
0 = El oscilador está apagado (el inversor del oscilador está apagado para eliminar el consumo de energía

bit 2 **T1SYNC**: Timer1 Entrada de reloj externo Bit de control de sincronización
Cuando TMR1CS = 1:
1 = No sincronizar entrada de reloj externo
0 = sincronizar entrada de reloj externo
Cuando TMR1CS = 0:
Este bit se ignora. Timer1 usa el reloj interno cuando TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit
1 = Reloj externo desde el pin RC0 / T1OSO / T1CKI (en el borde ascendente)
0 = Reloj interno (FOSC / 4)

bit 0 **TMR1ON**: Timer1 On bit
1 = Habilita Timer1
0 = Detiene el temporizador 1

Imagen 7-3 Diagrama a bloques del TIMER1



La siguiente formula es la que se utiliza para conocer los valores que debemos de asignar para los temporizadores.

RegValue = 65536- ((Delay * Fosc) / (Predivisor * 4))

7.2.1 Utilizar el temporizador sin interrupción

Mostrar en el display de 7 segmentos las cifras hexadecimales de 0a F de manera indefinida, las cifras cambiaran cada 500ms.

Para obtener los valores correspondientes al TMR1H y TMR1L debemos de ocupar la formula anterior descrita en el inicio de este capítulo.

Cálculos de retardo para 100ms a 16Mhz con Predivisor como 8:

$$\boldsymbol{RegValue} = \boldsymbol{65536} - \left(\frac{\boldsymbol{Delay} * \boldsymbol{Fosc}}{\boldsymbol{Predivisor} * \boldsymbol{4}}\right)$$

$$= \boldsymbol{65536} - \left(\frac{\boldsymbol{100ms} * \boldsymbol{16Mhz}}{\boldsymbol{8} * \boldsymbol{4}}\right) = \boldsymbol{15536} = \boldsymbol{0x3CB0}.$$

Lo que nos da lo siguiente:

$$\boldsymbol{TMR1 = 15536 = 0x3CB0 \rightarrow TMR1H = 0x3C\ y\ TMR1L = 0XB0}$$

Necesitamos ahora la configuración del T1CON, si revisamos los datos en la Tabla 7-3 T 1CON, debemos configurar que la preescala debe de ser 8, **T1OSCEN** = 0, **T1SYNC** = 0, **TMR1CS** = 0 y **TMR1ON** = 1. Lo que nos da como resultado lo siguiente $T1CON = 0b00110001 = 0x31$.

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "control_7segmentos.h"

//DEFINICIÓN DE VARIABLES
unsigned char valorDigito = 0;  //Contador del dígito a mostrar
int cont = 0;

void main(void) {
    /*Configuración de los puertos
    *
    */

    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms
    set_timer1(0x3CB0);                         //Ajusta el los registros contadores del Timer1
            setup_timer1(T1_ON|T1_INT|T1_DIV8);       //Timer 1 ON con contador interno y predivisor 8
    while(1){
        mostrarNumero(valorDigito);     //Muestra el valor en el display
        if(TMR1IF){                                 //Si la interrupción la ha generado TMR1, ejecutamos el código
            TMR1IF=0;              //Desactivamos el FLAG de la interrupción de TMR1
            set_timer1(0x3CB0);                         //Ajusta el los registros contadores del Timer1
            cont ++;
            if(cont >= 10){
                valorDigito++;                                  //Incrementa índice de la lista de códigos
```

```
                cont = 0;
            }
        }
                    if(valorDigito>=16){                    //Si se ha barrido toda la lista
                        valorDigito=0;          //Resetear índice de la lista
                    }
            }
}
```

7.2.2 Utilizar el temporizador con interrupción

Se pretende realizar un letrero con la palabra HOLA que se vaya desplazando de derecha a izquierda con una cadencia de 1segundo.

En la siguiente tabla se escribe el valor correspondiente para las letras HOLA.

	RC3	RC2	RC1	RC0	RE2	RE1	RE0	RA5	Valor HEX	Port C	Port E	Port A
Hex	D.P.	G	F	E	D	C	B	A				
H	0	1	1	1	0	1	1	0	0x76	0x07	0x03	0x00
O	0	0	1	1	1	1	1	1	0x3F	0x03	0x07	0x20
L	0	0	1	1	1	0	0	0	0x38	0x03	0x04	0x00
A	0	1	1	1	0	1	1	1	0x77	0x07	0x03	0x20

Actualizar cada digito entre 5ms o 10ms, para que sea visible para el ojo, evitar el uso de delay o algún otro retardo, todo deberá de ser por las interrupciones del timer1.

Usando la formula descrita en este capítulo del Timer 1, sustituimos los valores, delay de 5ms, Fosc 16MHz, predivisor 8.

$$TMR1 = 65536 - \left(\frac{Delay * Fosc}{(Predivisor * 4}\right)$$

$$= 65536 - \left(\frac{5ms * 4Mhz}{8 * 4}\right)$$

$$TMR1 = 63036 \rightarrow 0xF63C \rightarrow TMR1H = 0xF6 \; y \; TMR1L = 0x3C$$

La configuración del T1CON queda de la siguiente manera Timer 1 ON con contador interno y predivisor 8 0x31

$$T1CON = 0b00110001 = 0x31.$$

Consultar los anexos para las librerías faltantes

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "control_7segmentos.h"

//DEFINICIÓN DE VARIABLES
unsigned char valorDigito = 0;  //Contador del dígito a mostrar
int cont = 0;
unsigned char contLetras = 0;
unsigned char valorDeDigito = 0;


void main(void) {
    /*Configuración de los puertos
    *
    */

    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    TMR1H = 0xF6;
            TMR1L = 0x3C;
            T1CON = 0x31;
            //    set_timer1(0xEC78);               //Ajusta el los registros contadores del Timer1
```

```
        //    setup_timer1(T1_ON|T1_INT|T1_DIV8);  //Timer 1 ON con contador interno y predivisor 8 0x31

        GIE = 1;                                    //Interrupciones Globales habilitadas
        PEIE = 1;                                           //Interrupción por TMR1 habilitada
        TMR1IE = 1;                                 //Interrupción por TMR1 habilitada


    while(1){
        }
    return;
}

void __interrupt () isr(void){
        if(TMR1IF){                                         //Si la interrupción la ha generado TMR1, ejecutamos el código
                TMR1IF=0;           //Desactivamos el FLAG de la interrupción de TMR1
      TMR1H = 0xF6;
      TMR1L = 0x3C;
                //      set_timer1(0xEC78);           //Ajusta el los registros contadores del Timer1
                cont++;
                if (valorDeDigito == 0){
                        mostrarDigito1(contLetras);     //Mostrar el carácter para el Digito 1
                }
                if (valorDeDigito == 1){
                        mostrarDigito2(contLetras);     //Mostrar el carácter para el Digito 2
                }
                if (valorDeDigito == 2){
                        mostrarDigito3(contLetras);     //Mostrar el carácter para el Digito 3
                }
                if (valorDeDigito == 3){
                        mostrarDigito4(contLetras);     //Mostrar el carácter para el Digito 4
                }
                valorDeDigito ++;               //Incrementamos el valor del digito a mostrar
                if (valorDeDigito>=4)            //Si el valor del Digito es mayor a 4
                        valorDeDigito = 0;          //Reestablecemos el valor

                if(cont >=200){     //Si se cumple la cuenta de 5ms*200 = 1s
                        cont =0;
                        contLetras++;   //Desplazamos el carácter a mostrar
                        if (contLetras >=4)
                                contLetras = 0;
                }
        }
}
```

Notas:

7.3 Temporizador 2

Timer2 es un temporizador de 8 bits con un preescalador y un postscaler. Se puede utilizar como base de tiempo PWM para Modo PWM de los módulos CCP.

El módulo TImer2 es un temporizador / contador de 8 bits con las siguientes características:

- Temporizador / contador de 8 bits
- Legible y escribible
- Prescaler / PostScaler programable por software hasta 1:16
- Interrupción por desbordamiento de FFh a 00h

Registros Timer2

La siguiente tabla muestra los registros asociados con el módulo PIC16f877A Timer0.

Tabla 7-4 Registros Timer2

Registrarse	Descripción
T2CON	Estos registros se utilizan para configurar el prescalar TIMER2, la fuente del reloj, etc.
TMR2	Este registro contiene el valor de conteo del temporizador que se incrementará dependiendo de la configuración prescalar
PIR1	Este registro contiene el indicador de desbordamiento del Timer2 (TMR2IF).
PIE1	Este registro contiene el indicador de habilitación de interrupción de Timer2 (TMR2IE).

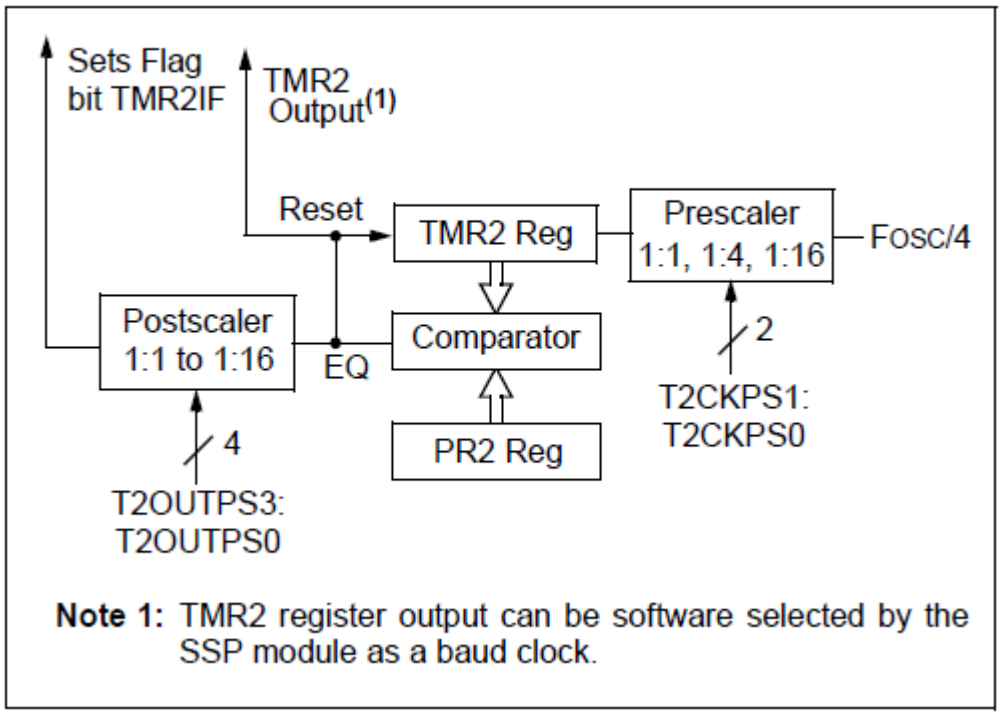
Tabla 7-5 T2CON

T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)							
7	6	5	4	3	2	1	0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

bit 7 **No implementado**: leer como ‘0’
bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Salida Post-DIVISOR Seleccionar bits
0000 = 1:1 Post-DIVISOR
0001 = 1:2 Post-DIVISOR
0010 = 1:3 Post-DIVISOR
•••
1111 = 1:16 Post-DIVISOR
bit 2 **TMR2ON**: Timer2 en bit
1 = Timer2 está encendido
0 = Timer2 está apagado
bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Selleccionar bits

00 = Pre-DIVISOR es 1
01 = Pre-DIVISOR es 4
1x = Pre-DIVISOR es 16

TIMER2 BLOCK DIAGRAM



Generación de un retardo de 1 segundo con Timer2:
Como el timer2 es de 8 bits y admite un prescaler 1:16, no es posible generar directamente el retardo de 1 segundo. El retardo máximo con un prescaler 1:16 será:

$$Delay = 256 * \frac{(Prescaler * 4)}{Fosc} = 256 * (\frac{(16 * 4)}{16Mhz}) = 1.024\text{ms}$$

Ahora se pueden generar 400µs usando temporizadores que se usarán para incrementar un contador 2,500 veces para obtener un retraso de 1 segundo.
Cálculos de retardo para 400µsec @ 16Mhz con Prescalar como 16:

$$RegValue = 256 - \left(\frac{Delay * Fosc}{Prescalar * 4}\right) = 256 - (\frac{400\mu s * 16Mhz}{16 * 4})$$
$$= 256 - 100 = 156$$

La Fórmula para calcular el valor de Timer2 es

$$Timer2 = Td = P1 * (N_{PR2} + 1) * P2 * Ti$$

Donde:

- N_{PR2} el valor que hay que cargar en el registro PR2
- $P1$ el factor de división del pre-divisor (**P1 = 1,4,16)**.
- $P2$ el factor de división del post-divisor **(P = 1,2,3,…,16)**.
- Ti el periodo de los pulsos internos de entrada al módulo **(Ti =4/FOSC = 4 * Tosc)**.

$$N_{PR2} = \left(\frac{Td}{P1 * P2 * Ti}\right) - 1$$

7.3.1 Utilizar el temporizador sin interrupción

Programar el Timer2, sin interrupción para obtener por RD4 una señal de 50Hz. El Timer2 debe activar la bandera TMR2IF cada milisegundo, siendo la frecuencia de oscilador principal de 16MHz.

Para corroborar el funcionamiento se puede conectar un osciloscopio al puerto RD5.

La temporización o tiempo de desbordamiento del Timer2 es de $Td = P1 * (N_{PR2} + 1) * P2 * T$

El Timer2 se configura con el registro T2CON, ver la Tabla 7-5 T2CON.
Con T2CON = 0b01001101, el Timer2 actúa como temporizador (Ti = 4/16MHz = 0.25µs), con P1 = 4 y P2 = 10, y habilitado.
Para Td = 1ms, el valor que se tiene que cargar al PR2 es igual al N_{PR2}, por lo tanto.

$$N_{PR2} = \left(\frac{Td}{P1 * P2 * Ti}\right) - 1 = \left(\frac{1ms}{4 * 10 * \left(\frac{4}{16MHz}\right)}\right) - 1 = 100 - 1 = 99$$

Código fuente:

```
// ARCHIVOS DE DEFINICIONES
#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "control_7segmentos.h"

// DEFINICION DE VARIABLES
int x;

// DEFINICION DE FUNCIONES
//Función de 1 ms
void DELAY1ms(void){
        while(TMR2IF == 0);                     //Espera mientras no se desborde el Timer 2
        TMR2IF=0;                               //Desactivar el FLAG de TMR2
}


void main(void) {
   /*Configuración de los puertos
   *
   */

   ADCON1 = 0x07;       // Deshabilitar PORTA Y PORTE de ADC
   TRISA = 0X03;        //Se configura 0-1 como 1, 2-5 como 0
   TRISB = 0X00;
   TRISC = 0x10;        //Solo como entrada RC4, los demás salidas
   TRISD = 0x0E;        //Entradas RD1-RD3, las demás salidas
   TRISE = 0X00;        //Todo como salidas

   PORTA = 0xFF;
   PORTE = 0xFF;
   PORTC = 0xFF;
   PORTD = 0xFF;
   __delay_ms(500);
   PORTA = 0x00;
   PORTE = 0x00;
   PORTC = 0x00;
   PORTD = 0x00;
```

```
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    T2CON = 0b01001101;                         //TMR2 con P1=4 y P2=10 y encendido
            PR2 = 99;                                           //Carga para 1 ms
            while(1){
                    for(x=0;x<10;x++){                  //Espera durante
                            DELAY1ms();                         // 10 ms
                    }
                    PORTDbits.RD4 = ~PORTDbits.RD4;         //Cambiar estado LED Verde Puerto RD4
                    PORTCbits.RC5 = ~PORTCbits.RC5;  //Cambiar estado de ánodo
            }

    return;
}
```

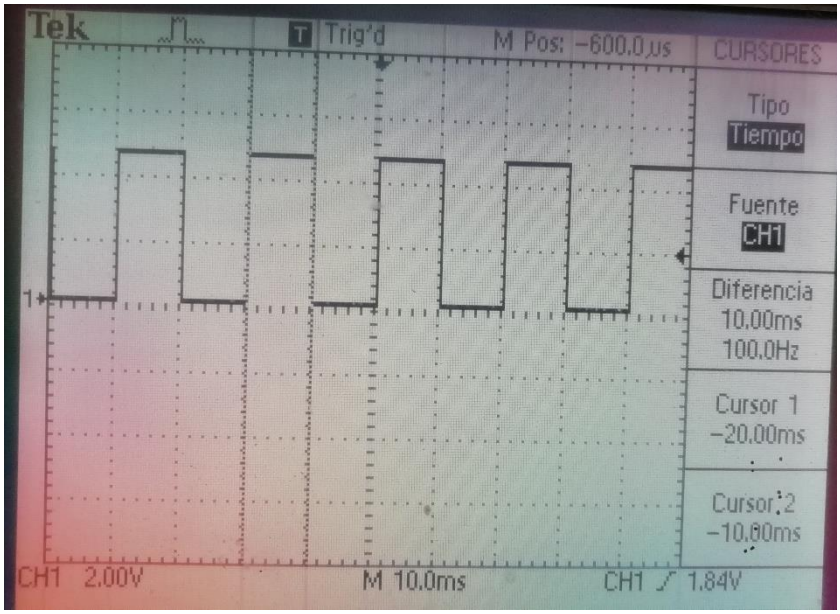


Imagen 7-4 Tiempo de pulso 10ms

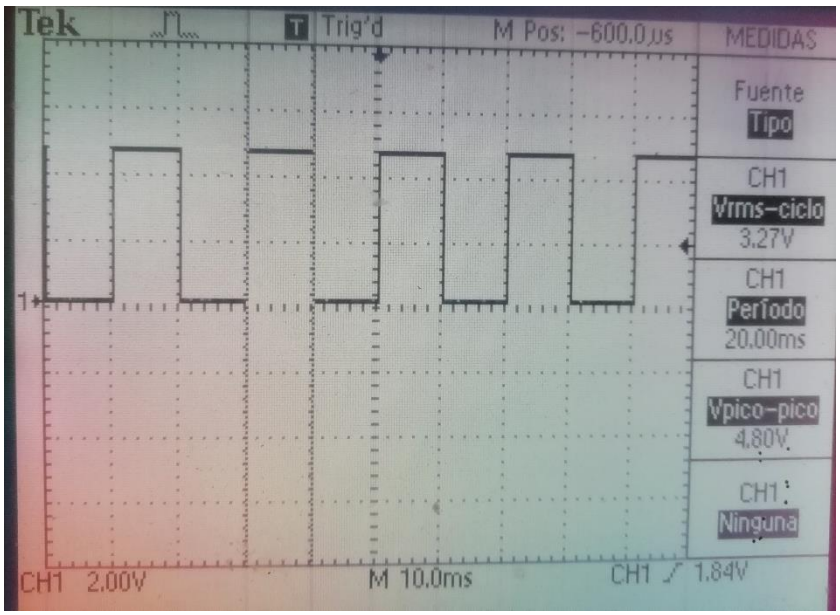


Imagen 7-5 Ciclo completo 20ms

Notas:

7.3.2 Utilizar el temporizador con interrupción

Se pretende realizar un cronómetro digital que cuente segundos y minutos, solo la cuenta llegara de 0 a 59 minutos y se reiniciara. Se quiere disponer de tres pulsadores uno para inicie el cronómetro, otro para parar el conteo y el ultimo para reiniciar el cronómetro.

Utilizar el temporizador 2 con interrupciones para conseguir la cadencia de 1ms para actualizar cada display, después de 1000 vueltas de 1ms se incrementa el conteo de 1 segundo, al llegar a los 60 segundos tendremos 1 minuto.

El Timer2 se configura con el registro T2CON.
Con T2CON =0b01001101 el Timer2 actúa como temporizador (Ti = 4/16MHz = 0.25µs).
Con P1 = 4 y P2= 10 y habilitado.

Para Td = 5ms, el valor que hay que cargar el PR2 es N_{PR2} = 99.

$$N_{PR2} = \left(\frac{Td}{P1 * P2 * Ti}\right) - 1 = \left(\frac{1ms}{4 * 10 * \left(\frac{4}{16MHz}\right)}\right) - 1 = 100 - 1 = 99$$

Código fuente:

```
// ARCHIVOS DE DEFINICIONES
#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "control_7segmentos.h"

// DEFINICION DE VARIABLES
#define PAUSA   0
#define START   1

unsigned char segundos = 0;
unsigned char minutos = 0;
unsigned char estado = PAUSA;
int cont = 0;
unsigned char valorDeDigito = 0;
unsigned char flagPush1 = 0;
unsigned char flagPush2 = 0;

void reiniciarCrono(void);

// PROGRAMA PRINCIPAL
void main(void) {
    /*Configuración de los puertos */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
```

```
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    dividirMinutos (minutos);
    dividirSegundos (segundos);
    ei();                                   //Interrupciones Globales habilitadas
            PEIE=1;                                 //Interrupción por TMR2 habilitada
            TMR2IE=1;                               //Interrupción por TMR2 habilitada
            setup_timer2(T2_ON|T2_PRED_DIV4|T2_POST_DIV10); //TIMER2 con P1=4 y P2=10 y encendido
            set_timer2(99);                 //Carga para 1ms

    apagarDigito ();

            while(1){
       dividirMinutos (minutos);
       dividirSegundos (segundos);
       if(PORTDbits.RD3 == 1 && flagPush1 == 0){   //Si el PUSH 1 es presionado y iniciamos el cronómetro
          flagPush1 = 1;             //bandera es igual a 1
          estado = START;
       }
       if (PORTDbits.RD3 == 0){       //Si no se oprime el Push 1
          flagPush1 = 0;             //Bandera igual a 0
       }
       if(PORTCbits.RC4 == 0 && flagPush2 == 0){   //Si el PUSH 2 es presionado pausamos el cronómetro
          flagPush2 = 1;             //bandera es igual a 1
          estado = PAUSA;
       }
       if (PORTCbits.RC4== 1){       //Si no se oprime el Push 1
          flagPush2 = 0;             //Bandera igual a 0
       }

       if(PORTDbits.RD2 == 1 && flagPush2 == 0){   //Si el PUSH 3 es presionado reiniciamos el cronómetro
          flagPush2 = 1;             //bandera es igual a 1
          reiniciarCrono();
       }
       if (PORTDbits.RD2 == 0){       //Si no se oprime el Push 3
          flagPush2 = 0;             //Bandera igual a 0
       }

            }
}

void __interrupt () isr(void){
    if(TMR2IF){
       TMR2IF=0;                                                //Desactiva el FLAG de la interrupción de TMR2
//       set_timer2(99);           //Carga para 1ms
       if(estado == START){       //Si el cronómetro está en marcha
          cont ++;
          if(cont >= 1000){       //Contador igual a 100 se cumple 1 segundo
             cont = 0;          //Reiniciamos el contador
             segundos++;        //incrementamos el contador de segundos
             if(segundos >= 60){ //Si el contador de segundos llega a 60 es igual a 1 minuto
                segundos = 0;   //Se reinicia el contador de segundos
                minutos ++;     //Incrementamos el contador de minutos
                if(minutos >=60){   //Si el contador de minutos llega a 60 se cumple una hora
                   minutos = 0;    //Se reinicia el contador de minutos
                }
             }
          }
       }
       if (valorDeDigito == 0){
                              mostrarUnidades(unidadesSegundos);
                    }
                    if (valorDeDigito == 1){
```

```
                        mostrarDecenas(decenasSegundos);
                }
                if (valorDeDigito == 2){
                        mostrarCentenas(unidadesMinuto);
        PORTC = PORTC + PUNTO_DECIMAL;
                }
                if (valorDeDigito == 3){
                        mostrarMillares(decenasMinuto);
                }
                valorDeDigito ++;               //Incrementamos el valor del digito a mostrar
                if (valorDeDigito>=4)            //Si el valor del Digito es mayor a 4
                        valorDeDigito = 0;              //Reestablecemos el valor

    }
}

void reiniciarCrono(void){
    cont = 0;
    segundos = 0;
    minutos = 0;
    return;
}
```

Para revisar que nuestro proyecto tiene los tiempos exactos, podemos conectar el osciloscopio al puerto RA2, en este pin se encuentra la salida para activar el digito 1, al conectar el osciloscopio notaremos los pulsos de activación para el digito, y un espacio entre pulso el cual representa el tiempo de activación de los otros dígitos. Como hemos establecido un tiempo de 1ms tendremos la activación de cada digito de 1ms (Imagen 7-6), al sumar los 4 dígitos debemos de obtener 4ms (Imagen 7-7). Mientras que en el osciloscopio veremos un pulso en alto de 1ms y un pulso bajo de 3ms (Imagen 7-8), tal y como se muestran en las siguientes imágenes.

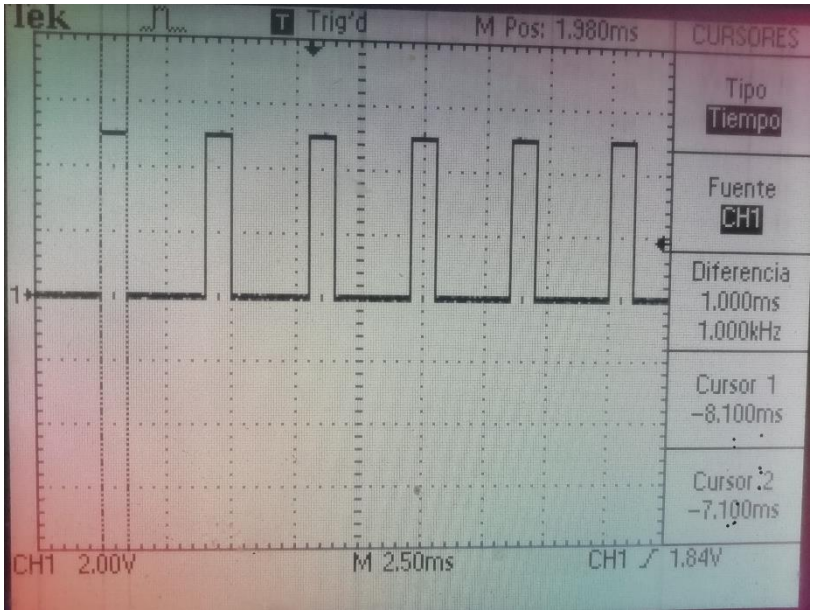


Imagen 7-6 Pulso de activación 1ms

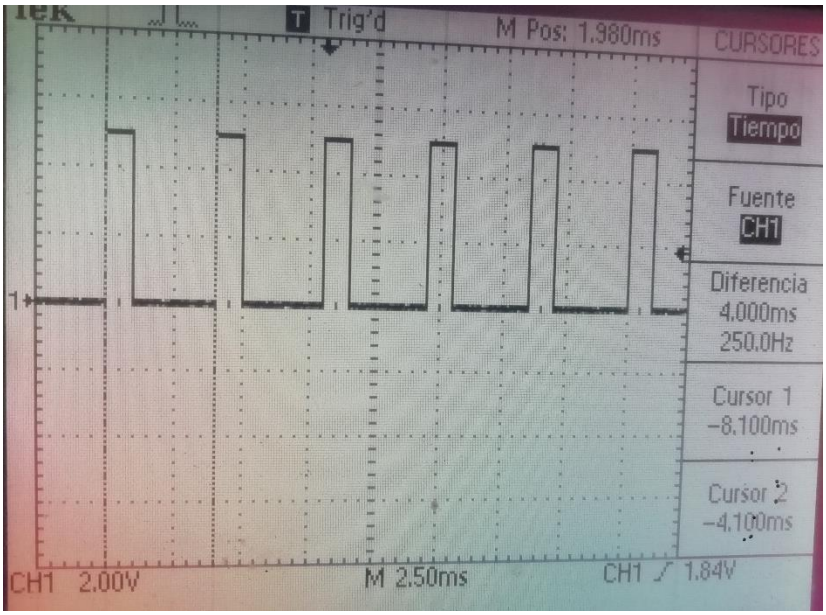


Imagen 7-7Tiempo total 4ms

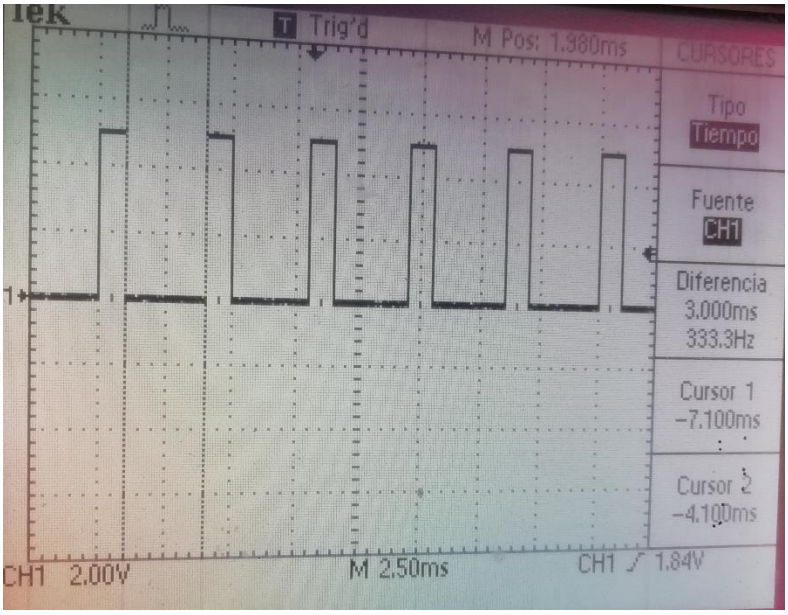


Imagen 7-8 Tiempo de los otros 3 dígitos 3ms

Notas:

7.4 WATCHDOG

La función del Watchdog es impedir que código o el sistema entre en un bucle infinito, provocando un RESET cuando sucede esto.
El Watchdog es un oscilador RC interno independientemente del oscilador principal del microcontrolador, por lo que funciona incluso cuando se detiene la ejecución mediante la instrucción SLEEP.

Se habilita o deshabilita mediante el bit de configuración WDTE, el cual no puede ser deshabilitado por software. Colocando el bit PSA a “1” se habilita el postdivisor del Watchdog, mediante él se puede establecer períodos entre 0.018 y 2.30 segundos.

Cuando el Watchdog se desborda, es decir alcanza el tiempo establecido, se resetea el microcontrolador, por lo que antes de que esto suceda se debe borrar mediante la instrucción CLRWDT.

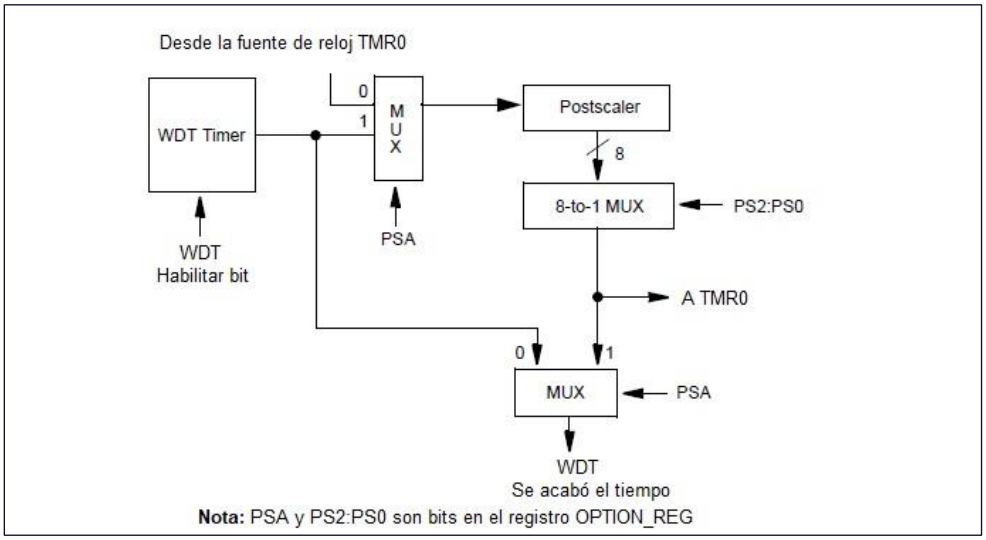


Imagen 7-9 Representación a bloques del Watchdog

En la siguiente tabla se muestra los registros para la configuración del OPTION_REG del Watchdog.

Tabla 7-6 Registro asociado al Watchdog

RESUMEN DE LOS REGISTROS DEL TEMPORIZADOR DE WATCHDOG								
Registro	7	6	5	4	3	2	1	0
OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

La siguiente tabla contiene los tiempos para el Watchdog, dependiendo del tiempo que se requiera es la configuración que debemos de colocar.

Tabla 7-7 Desbordamiento del WDT

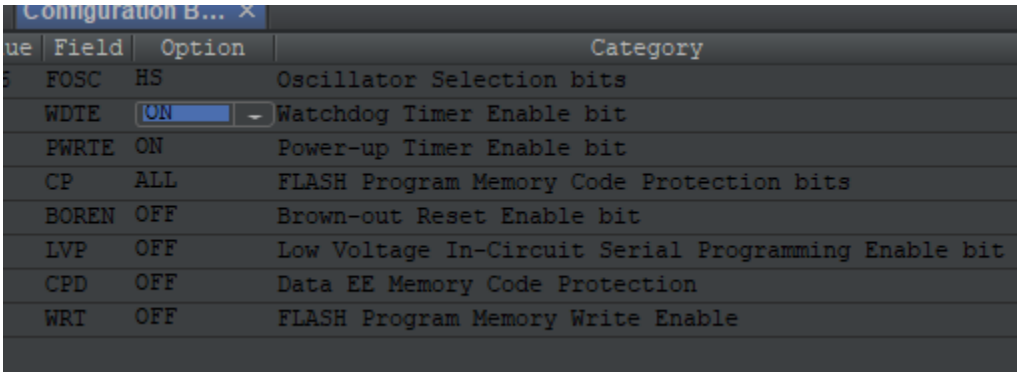
Desbordamiento del WDT	
PS2:PS0	Tiempo de desbordamiento (ms)
000	18
001	36
010	72
011	144
100	288
101	576
110	1152
111	2304

7.4.1 Uso de los TIMER y el Watchdog

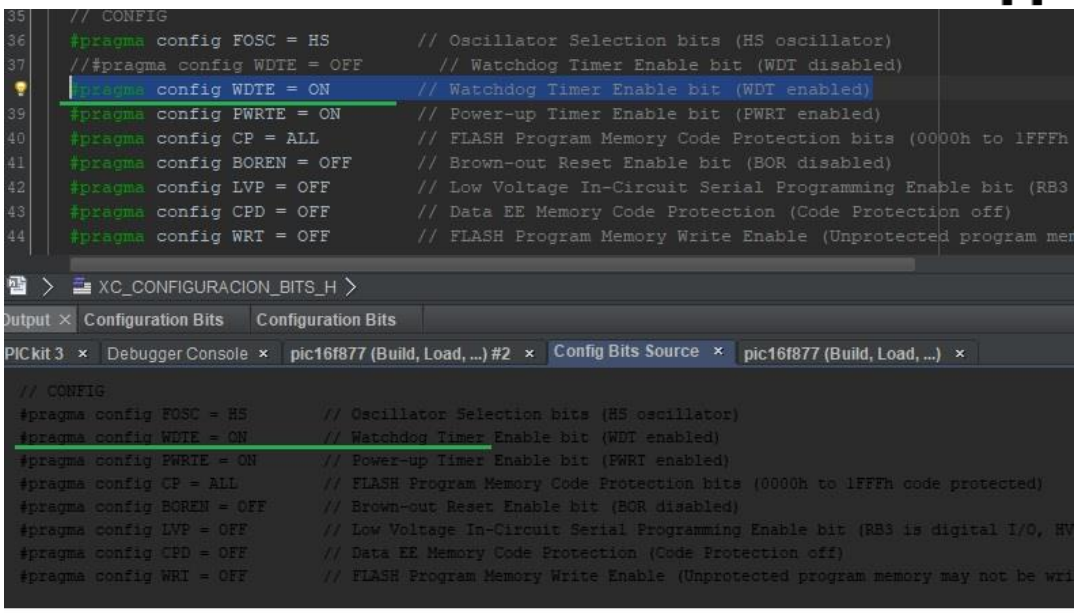
Se desea realizar un programa que, utilizando el pulsador conectado a RD3 un digito del display 7 segmentos, haga lo siguiente.

Al principio estarán apagados todos los segmentos y continuaran así hasta que se actúa sobre el pulsador, en ese momento comenzara un conteo de 0 a 9, cada 1s, por lo que en total serán 10 segundos la cuenta, al final de los 10 segundos se retorna el estado inicial, esto con el uso del Watchdog, meteremos el proceso en bucle infinito para que el mismo Watchdog reinicie el proceso. Actualizar el número cada 50ms, por lo tanto, tendremos dos timers el Timer0 y el Timer1, el Timer0 será el que actualice el digito cada 50 ms, mientras que el Timer1 será un contador de 500ms para realizar el conteo del tiempo y cuando llegue a los 10s entrará el bucle infinito.

Para activar el Watchdog debemos de configurarlo, esta configuración las podemos realizar tal y como realizamos la configuración de bits en el capítulo Configuración de Bits, pero ahora activaremos el Watchdog tal y como se muestra a continuación.



Generamos el código y copiamos la parte del WDTE a la cabecera de configuración de bits.



Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "control_7segmentos.h"

// DEFINICION DE VARIABLES
#define PAUSA   0
#define START   1

unsigned char cont1 = 0;                                    //Contador de desbordamiento de TMR1
unsigned char contadorTime1 = 0;
unsigned char valor = 9;
#define OFF 0
#define ON  1
unsigned char estado = OFF;
unsigned char flagPush1 = OFF;
// PROGRAMA PRINCIPAL
void main(void) {
    /*Configuración de los puertos */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms
    // Configuración  del TIMER 0 para producir interrupciones cada 10 ms.
    OPTION_REG=0xC7;                //TIMER0 como temporizador con predivisor P=256 WDT = 128
```

```
    TMR0=99;                                    //Para que Td de TMR0 sean 10ms y empiece el TMR0
    GIE=1;                                      //Habilitar interrupciones
    T0IE=1;                                     // Habilitar interrupción de TMR0
    T0IF=0;                                     //Desactivar el FLAG de TMR0
            while(1){
       CLRWDT();                                //Reset del Watchdog
       if(PORTDbits.RD3 == 1 && flagPush1 == OFF){   //Si el PUSH 1 es presionado y iniciamos el cronómetro
          flagPush1 = ON;            //bandera es igual a 1
          estado = ON;

       // Configuración  del TIMER1 para producir interrupciones cada 100 ms.
          T1CON=0x30;                           //Configurar TIMER 1 como temporizador, con P=8 y parado
          PEIE=1;                               //Habilitar interrupción de TMR1
          TMR1IE=1;                             //Habilitar interrupción de TMR1
          TMR1IF=0;                             //Desactivar el FLAG de TMR1
          TMR1H=0x3C;                           //Para que Td de Timer 1 sean 100 ms.
          TMR1L=0xB0;
          TMR1ON=1;                             //Activado el TIMER 1
       }
       if (PORTDbits.RD3 == 0){       //Si no se oprime el Push 1
          flagPush1 = OFF;             //Bandera igual a 0
       }
            }
}

void __interrupt () isr(void){
   if(T0IF){
                    T0IF=0;                                 //Desactivar el FLAG de la interrupción de TMR0
                    TMR0=99;                                    //Recargar TMR0
       mostrarNumero(valor);
            }
            if(TMR1IF == 1){
                    TMR1IF=0;                                   //Desactivar el FLAG de la interrupción de TMR1
       TMR1H=0x3C;                          //Para que Td de Timer 1 sean 100 ms.
       TMR1L=0xB0;
       contadorTime1 ++;
       if(contadorTime1 >= 10){ //10 interrupciones igual a 1 segundo
          contadorTime1 = 0;
          valor --;       //Incrementar valor del digito
       }
                    cont1++;                    //Incrementar contador de interrupciones de Timer 1.
                    if(cont1==100){             //100 interrupciones cada 100 ms da lugar a 10s
                            TMR1ON=0;                   //Desactivar el TIMER
                            GIE=0;                      //Deshabilitar interrupciones
                            while(1);           //Bucle infinito
             //Como ya trascurrieron los 10 segundos, nos quedamos en este bucle infinito
             //sin resetear el Watchdog hasta que desborde y reinicie el microcontrolador.
             //Tras 18 mseg (valor típico) el micro se reseteará por desbordamiento del Watchdog
                    }
            }
}
```

Notas:

8 Módulos analógicos

El módulo del convertidor A/D dispone de las siguientes características:[6]

- El convertidor genera un resultado binario de 10 bits utilizando el método de aproximaciones sucesivas y almacena los resultados de conversión en los registros ADC (ADRESL y ADRESH);
- Dispone de 14 entradas analógicas separadas;
- El convertidor A/D convierte una señal de entrada analógica en un número binario de 10 bits;
- La resolución mínima o calidad de conversión se puede ajustar a diferentes necesidades al seleccionar voltajes de referencia Vref- y Vref+.

La conversión de una señal de entrada analógica da como resultado un número digital correspondiente de 10 bits. El módulo A/D tiene una entrada de referencia de alto y bajo voltaje que es seleccionable a alguna combinación de VDD, VSS, RA2 o RA3. Con 5v como Vref, la resolución de Pic16f877a ADC se puede determinar de la siguiente manera:

$$resolución\ de\ ADC\ = \frac{Vref}{2^{10}-1}\ = \frac{5}{1023}\ =\ 0.004887\ =\ 4.887mv$$

La siguiente tabla muestra los pines de entrada ADC multiplexados con otros pines GPIO.
El pin ADC se puede habilitar configurando el registro ACON1 correspondiente.
Cuando se selecciona la función ADC para un pin, entonces otras señales digitales se desconectan de los pines de entrada ADC.[7]

Tabla 8-1 Pin ADC

Canal adc	Pin Pic16f877a	Función Pin
0	RA0	AN0
1	RA1	AN1
2	RA2	AN2 / VREF-
3	RA3	AN3 / VREF +
4	RA5	AN4
2	RE0	AN5
3	RE1	AN6
4	RE2	AN7

El convertidor A/D tiene la característica única de poder operar mientras el dispositivo está en modo de suspensión.
El módulo A / D tiene cuatro registros. Estos registros son:

[6] **https://www.mikroe.com/ebooks/microcontroladores-pic-programacion-en-c-con-ejemplos/modulos-analogicos**
[7] **https://exploreembedded.com/wiki/ADC_Using_PIC16F877A**

- Registro alto de resultado A/D (ADRESH)
- Registro bajo de resultado A/D (ADRESL)
- Registro de control A/D0 (ADCON0)
- Registro de control A/D1 (ADCON1)

La siguiente tabla muestra los registros asociados con PIC16F877A ADC.

Tabla 8-2 Registros

Registro	Descripción
ADCON0	Se usa para encender el ADC, seleccionar la frecuencia de muestreo y también iniciar la conversión.
ADCON1	Se usa para configurar los pines GPIO para ADC
ADRESH	Mantiene el byte más alto del resultado de ADC
ADRESL	Mantiene el byte más bajo del resultado de ADC

Ahora veamos cómo configurar los registros individuales para la comunicación UART.

Tabla 8-3 ADCON0

ADCON0 REGISTER (ADDRESS 1Fh)							
R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	U-0	R/W -0
7	6	5	4	3	2	1	0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/ $\overline{DONE}$	-	ADON

bit 7-6 **ADCS1:ADCS0**: Bits de selección de reloj de conversión A/D

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Conversión de reloj
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (reloj derivado del oscilador interno A/D RC)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (reloj derivado del oscilador interno A/D RC)

bit 5-3 **CHS2:CHS0**: Bits de selección de canal analógico

000 = Canal 0 (AN0)
001 = Canal 1 (AN1)
010 = Canal 2 (AN2)
011 = Canal 3 (AN3)

100 = Canal 4 (AN4)
101 = Canal 5 (AN5)
110 = Canal 6 (AN6)
111 = Canal 7 (AN7)

bit 2 $\boldsymbol{GO/\overline{DONE}}$: bit de estado de conversión A/D
Cuando ADON = 1:
1 = conversión A/D en curso (la configuración de este bit inicia la conversión A/D que se borra automáticamente por hardware cuando se completa la conversión A/D)
0 = Conversión A/D no en curso

bit 1 **No implementado**: leer como '0'

bit 0 **ADON**: A/D en bit
1 = El módulo convertidor A/D está encendido
0 = El módulo convertidor A/D está apagado y no consume corriente de funcionamiento

Tabla 8-4 ADCON1

ADCON1 REGISTER (ADDRESS 9Fh)							
R/W -0	R/W -0	U-0	U-0	R/W -0	R/W -0	R/W -0	R/W -0
7	6	5	4	3	2	1	0
ADFM	ADCS2	-	-	PCFG3	PCFG2	PCFG1	PCFG0

bit 7 **ADFM**: Formato de resultado A/D Seleccione bit
1 = Justificado a la derecha. Los seis (6) bits más significativos de ADRESH se leen como '0'.
0 = Justificado a la izquierda. Seis (6) bits menos significativos de ADRESL se leen como '0'.

bit 6 **ADCS2**: bit de selección de reloj de conversión A/D
Compruebe ADCS1: ADCS0 del registro ADCON0.

bit 5-4 **No implementado**: leer como '0

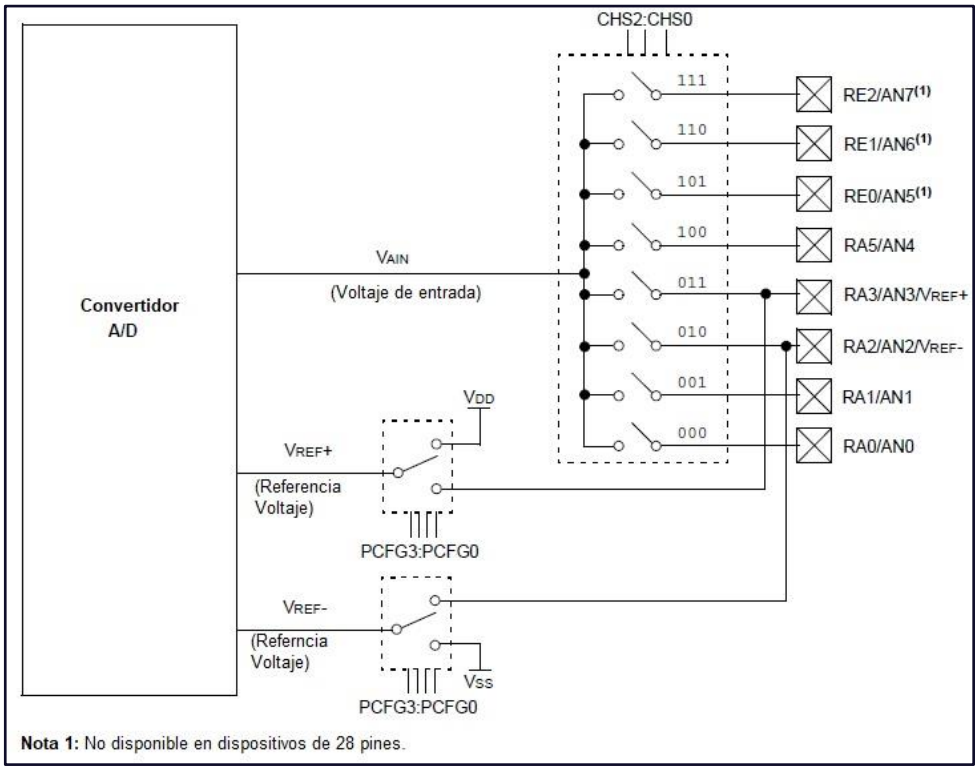
bit 3-0 **PCFG3: PCFG0**: Bits de control de configuración del puerto A/D

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A= Entrada analógica D = Digital I/O

C / R = # de canales de entrada analógica / # de referencias de voltaje A/D

Imagen 8-1 Diagrama a bloque A/D



El tiempo de conversión A / D por bit se define como TAD. La conversión A / D requiere un mínimo de 12 TAD por conversión de 10 bits. La fuente del reloj de conversión A/D se selecciona mediante software. Las siete posibles opciones para TAD son:

- 2 T OSC
- 4 T OSC
- T OSC
- 16 T OSC
- 32 T OSC
- 64 T OSC
- Oscilador RC del módulo A/D interno (2-6μs)

Para conversiones A/D correctas, el reloj de conversión A/D (TAD) se debe seleccionar para asegurar un tiempo TAD mínimo de 1.6μs.

La tabla muestra los tiempos TAD resultantes derivados de las frecuencias de operación del dispositivo y la fuente de reloj A/D seleccionada.

Tabla 8-5 TAD VS FRECUENCIAS MÁXIMAS DE FUNCIONAMIENTO DEL DISPOSITIVO (DISPOSITIVOS ESTÁNDAR (F))

Fuente de reloj AD (TAD)		Frecuencia máxima del dispositivo
Operación	ADCS2: ADCS1: ADCS0	
2 TOSC	000	1.25 MHz
4 TOSC	100	2.5 MHz
8 TOSC	001	5 MHz
16 TOSC	101	10 MHz
32 TOSC	010	20 MHz
64 TOSC	110	20 MHz
RC(1,2,3)	x11	(**Nota 1)**

Nota **1**: La fuente RC tiene un tiempo T AD típico de 4 μs, pero puede variar entre 2-6μs.

2: Cuando las frecuencias del dispositivo son superiores a 1 MHz, la fuente de reloj de conversión RC A / D solo se recomienda para la operación de suspensión.

3: Para dispositivos de voltaje extendido (LF),

8.1 Lectura de tensión analógica

La configuración básica de lectura para los canales analógicos es la siguiente.
Código fuente:

```
void ADC_Init()
{
   ADCON0 = 0x81;   // ADCS1=1, ADCS0=0, Canal 0, Módulo encendido
   ADCON1 = 0x80;   // Justificado a la derecha, FOSC/32
}

unsigned int ADC_Read(unsigned char channel)
{
   if(channel > 7)
     return 0;

   ADCON0 &= 0xC5;
   ADCON0 |= channel<<3;
   __delay_ms(2);
   GO_nDONE = 1;
   while(GO_nDONE);
   return ((ADRESH<<8)+ADRESL);
}
```

Llamamos el ADC_Init(), en el main, esto configura el módulo del ADC.
En el ADC_Read(# Canal a leer), en esta función se leerá el canal que se requiera leer, desde el 0 al 7, y nos devolverá el resultado del ADC.

La lectura recibida esta basada en la resolución del microcontrolador por lo tanto la tenemos que convertir, usando la siguiente formula.

$$resolución\ de\ ADC\ = \frac{Vref}{2^{10}-1}\ = \frac{5}{1023}\ =\ 0.004887\ =\ 4.887mv$$

$$V_{ADC} = ADC * Res_{ADC}$$

Por ejemplo:

- Si el ADC nos regresa un valor de 1023 este valor lo tenemos que multiplicar por la resolución del ADC, que en este caso es de 4.887mv, así que el resultado es de 4.999V, un valor muy cercano a los 5V.
- Ahora, supongamos que la lectura del ADC es de 250, por lo que a la entrada del canal analógico es de 1.22 V.

8.1.1 Lectura de Trimpot

Se pretende tomar lectura del potenciómetro o trimpot que se encuentra en la tarjeta SIRIUS P16, la salida del trimpot se encuentra conectado al puerto A, en el pin RA0. En la siguiente imagen se muestra el diagrama de conexión del trimpot.

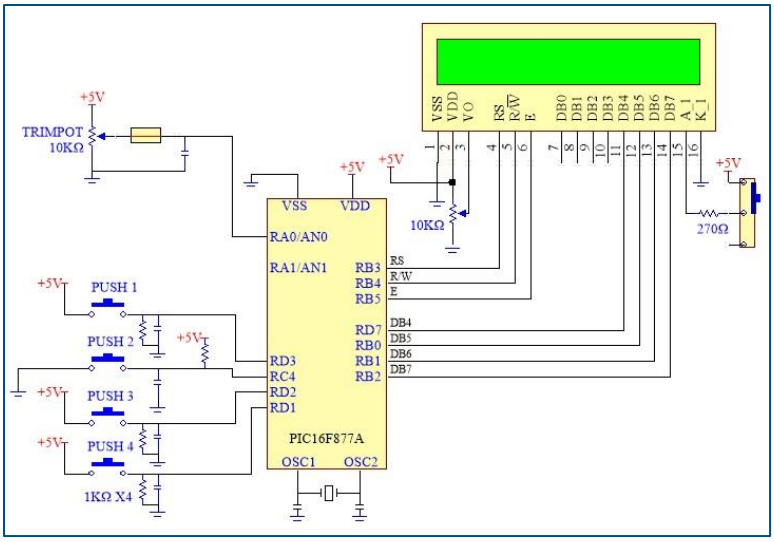


Imagen 8-2 Conexión Trimpot

Para ubicar que Trimpot es el que está conectado al puerto RA0 revisar la Imagen 4-1 Tarjeta el cual es el componente 9.- Potenciómetro para señal analógica.

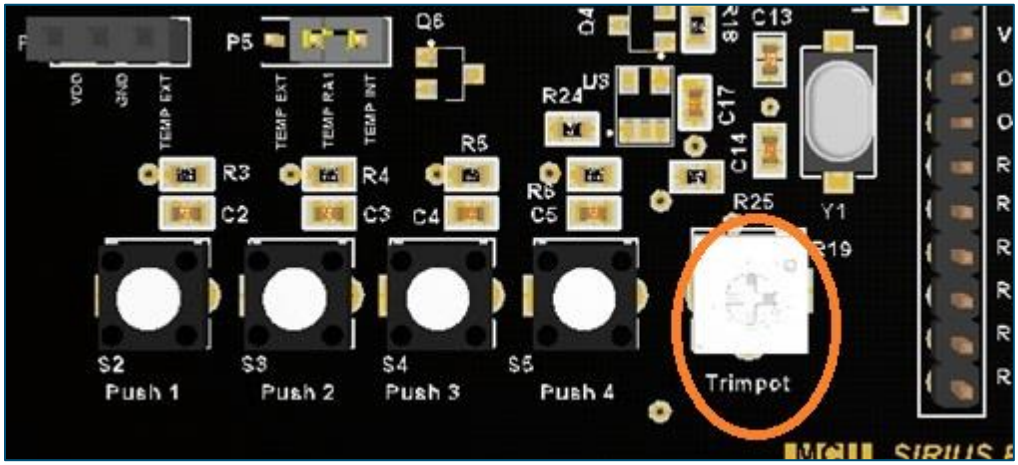


Imagen 8-3 Ubicación del Trimpot

Ya que tenemos ubicado el Trimpot podemos hacer el programa para tomar la lectura de este puerto, solo lo leeremos y mostraremos el valor en el LCD, en los siguientes dos capítulos le aplicaremos las fórmulas para desplegar el valor correcto.

Nota: Para validar el resultado se recomienda colocar un multímetro en la entrada de la señal RA0 ubicada en el 7.header de conexión.

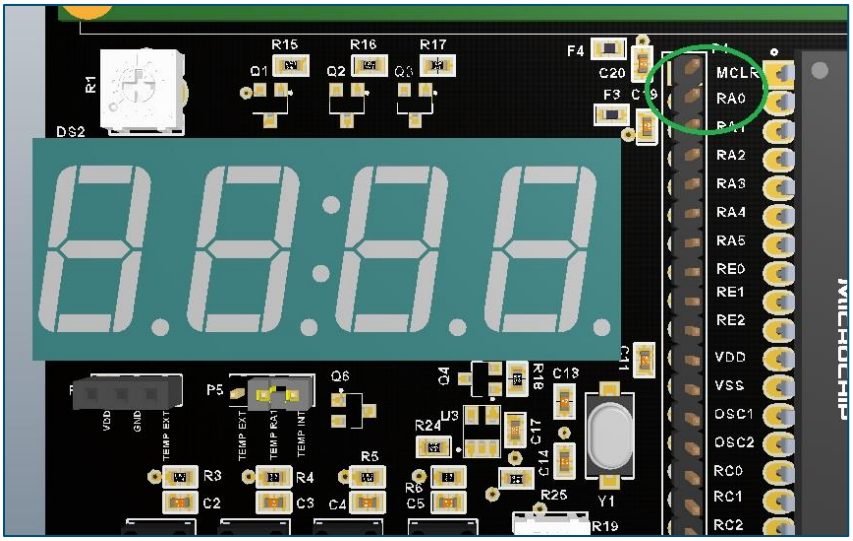


Imagen 8-4 Header de conexión

La conexión del multímetro debe de ser el positivo en RA0 y el negativo en VSS (GND), el multímetro debe de estar en la opción de VC.D.

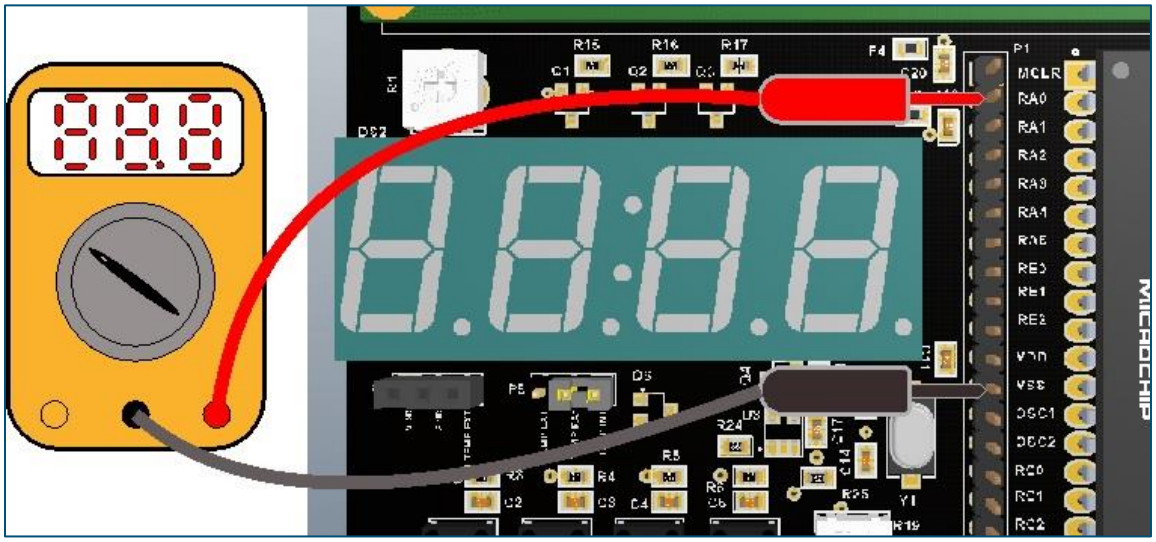


Imagen 8-5 Multímetro y RA0

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "adc.h"
#include "lcd.h"
#include <stdlib.h>


// DEFINICION DE VARIABLES
unsigned char contadorDisplay = 0;
char buffer[80];

int adc = 0;
unsigned char  voltaje = 0;

// PROGRAMA PRINCIPAL
void main(void) {
    /*Configuración de los puertos */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    Lcd_Init();               //Inicializamos la LCD
    Lcd_Cmd (LCD_CLEAR);       //Limpiamos la pantalla
    Lcd_Cmd (LCD_CURSOR_OFF);
    Lcd_Out (1,5,"Hola");       //Colocar el cursor en fila 1 columna 3, escribir Hola
    Lcd_Out (2,5, "MCU");       //Colocar el cursor en fila 2 columna 5, escribir Mundo
    __delay_ms(1000);
    Lcd_Cmd (LCD_CLEAR);       //Limpiamos la pantalla

    Lcd_Out(1,3,"Valor ADC");          //Escribimos en la fila 1 columna 3

    ADC_Init();               //Iniciamos el ADC
    while(1){
        adc = ADC_Read(0);              //Leer el valor del canal AN0
        itoa(buffer,adc,10);            //Unir el valor en un buffer
        Lcd_Out(2,0,"          ");     //Borrar Linea
        Lcd_Out2(2,5,buffer);           //Mostrar el valor en la fila 2 columna 5
    }
    return;
}
```

Puedes ir ajustando el trimpot desde lo más bajo a los más alto, te deberá de mostrar un valor de 0 a 1023.

Notas:

8.1.1.1 Desplegar lectura en display 7 segmentos

Mostraremos el resultado de la lectura del canal AN0 por los 4 dígitos del display, ya sabemos cómo leer un canal analógico y sabemos manejar la pantalla LCD.

Mostraremos solo por los dos dígitos el valor de la lectura, si requiere más decimales puede modificar el código y así mostrar hasta tres valores después del punto decimal.

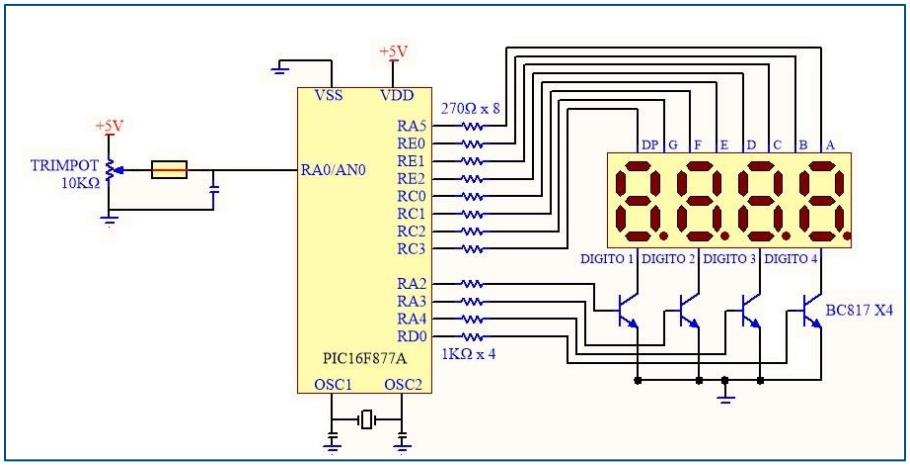


Imagen 8-6 Trimpot Y display 7 segmentos

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "control_7segmentos.h"
#include "adc.h"


// DEFINICION DE VARIABLES
unsigned char contadorDisplay = 0;
char buffer[80];

int adc = 0;
unsigned char  voltaje = 0;

// PROGRAMA PRINCIPAL
void main(void) {
    /*Configuración de los puertos */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
```

```
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

    ADC_Init();               //Iniciamos el ADC
    // Configuración  del TIMER 0 para producir interrupciones cada 50 ms.
    OPTION_REG = 0xC7;              //TIMER0 como temporizador con predivisor P=256
    TMR0 = 99;                                  //Para que Td de TMR0 sean 10ms y empiece el TMR0
    GIE=1;                                      //Habilitar interrupciones
    T0IE=1;                                     // Habilitar interrupción de TMR0
    T0IF=0;

    apagarDigito ();
    float x = 0;
    while(1){
        adc = ADC_Read(0);              //Leer el valor del canal AN0
        x = adc * (RESOLUCION_ADC /1000);
        voltaje = x *10;
        manejadorDelValor (voltaje);
    }
    return;
}


// INTERRUPCIÓN por Timer0
void __interrupt () isr(void){
            if(T0IF){
        T0IF=0;            //Desactivar el FLAG de TMR0
        TMR0 = 99;           //Recargar TMR0 10ms
        if(contadorDisplay == 0){       //Contador = 0 Mostrar digito 1
            mostrarUnidades(unidades);
        }
        if(contadorDisplay == 1){       //Contador = 1 Mostrar digito 2
            mostrarDecenas(decenas);//Contador = 1 Mostrar digito 2
            PORTC = PORTC + PUNTO_DECIMAL;
        }
        if(contadorDisplay == 2 ){      //Contador = 2 Mostrar digito 3
            mostrarCentenas(centenas);
        }
        if(contadorDisplay == 3){       //Contador = 3 Mostrar digito 4
            mostrarMillares(millares);
        }
        contadorDisplay ++;     //Contador del digito a mostrar
        if(contadorDisplay >= 2)    //Si el contador es mayor o igual a 4, se reinicia
            contadorDisplay = 0;
    }
}
```

Comparar el resultado con la lectura del multímetro conectado al puerto RA0 del heder de conexión

Notas:

8.1.1.2 Desplegar lectura en display LCD

Mostraremos el resultado de la lectura del canal AN0 por la LCD, ya sabemos como leer un canal analógico y sabemos manejar la pantalla LCD.

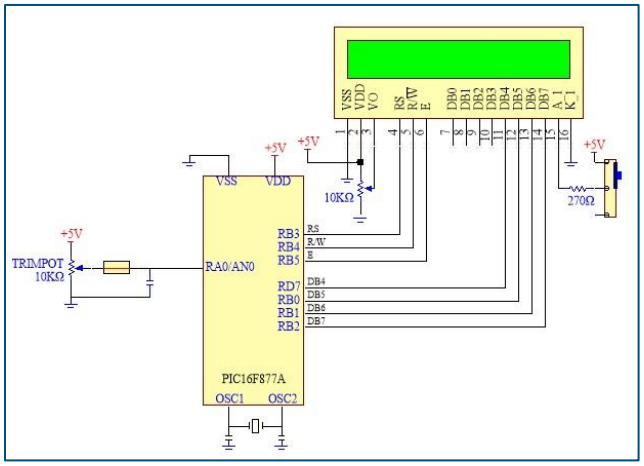


Imagen 8-7 Trmpot y LCD

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "adc.h"
#include "lcd.h"
#include <stdio.h>
#include <stdlib.h>


// DEFINICION DE VARIABLES
char buffer[80];
int adc = 0;
unsigned char  voltaje = 0;

// PROGRAMA PRINCIPAL
void main(void) {
    /*Configuración de los puertos */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

        Lcd_Init();               //Inicializamos la LCD
    Lcd_Cmd (LCD_CLEAR);       //Limpiamos la pantalla
```

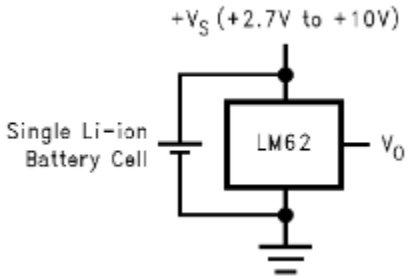
```
    Lcd_Cmd (LCD_CURSOR_OFF);
    Lcd_Out (1,5,"Hola");       //Colocar el cursor en fila 1 columna 3, escribir Hola
    Lcd_Out (2,5, "MCU");       //Colocar el cursor en fila 2 columna 5, escribir Mundo
    __delay_ms(1000);
    Lcd_Cmd (LCD_CLEAR);        //Limpiamos la pantalla
    ADC_Init();                 //Iniciamos el ADC

            while(1){
       float voltaje = 0;
       adc = ADC_Read(0);               //Leer el valor del canal AN0
       itoa(buffer,adc,10);             //Unir el valor en un buffer
       Lcd_Out(1,0,"ADC = ");
       Lcd_Out(1,5,"     ");
       Lcd_Out2(1,5,buffer);            //Mostrar el valor en la fila 2 columna 5
       voltaje = adc * (RESOLUCION_ADC /1000);
       Lcd_Out(2,0,"Voltaje = ");
       Lcd_Out(2,9,"     ");
       sprintf(buffer,"%f",voltaje);
       Lcd_Out2(2,9,buffer);            //Mostrar el valor en la fila 2 columna 5
    }
    return;
}
```

Notas:

8.1.2 Lectura de Temperatura

La tarjeta cuenta un sensor de temperatura interno, este sensor es un LM62 la temperatura de operación es de 0°C a 90°C, con un voltaje de operación de 2.7V a 10V.
Este sensor cuenta con una resolución de 15.6mV/ºC de acuerdo con la hoja de datos tenemos la siguiente información:



$$V_{out} = \left(+15.6\frac{mV}{ºC} * T\ ºC\right) + 480mV$$

Temperatura	Típico Vout
+90 ºC	+ 1884mV
+70 ºC	+ 1572mV
+25 ºC	+ 870mV
0 ºC	+ 480mV

Con estos datos podemos calcular la temperatura que se encuentra en nuestra tarjeta, siguiendo los siguientes pasos.

Tomemos como ejemplo para una temperatura de 25ºC, para esta temperatura tenemos un voltaje de salida de 870mv, ahora este valor lo debe de convertir el ADC, conociendo que el microcontrolador tiene una resolución de 4.887mV tenemos el siguiente valor.

$$ADC = \frac{V_{in}}{Resolución} = \frac{870mV}{4.887mV} = 178.023\ \approx 178$$

Nuestro valor del ADC es de 178, ahora este valor es el que nos estará entregando el convertidor de ADC, por lo que debemos de regresar ese valor a voltaje.

$$V_{out} = ADC * Resolución = 178 * 4.887mV = 869.88mv$$

Tenemos el valor de voltaje, es muy cercano al valor que nos muestra la table de temperatura. Con este valor debemos de sustituir los datos en la fórmula que esta al principio de ese capítulo, la cual es la siguiente:

$$V_{out} = \left(+15.6\frac{mV}{ºC} * T\ ºC\right) + 480mV$$

Despejamos T que es igual a Temperatura y obtenemos el siguiente resultado

$$Temp = \frac{V_{out} - 480mV}{15.6mV/ºC} = \frac{869mV - 480mV}{15.6mV/ºC} = 24.93\ ºC$$

Obtenemos una temperatura de 24.93ºC que es muy cercana a los 25ºC que tomamos como ejemplo.

En la tarjeta encontraremos el sensor de temperatura revisando el tema Contenido del kit de desarrollo SIRIUS P16, el sensor es el componente 13, recordar que el componente 11 es el header donde se debe de hacer el puente para seleccionar la señal que se requiere leer, en este

caso el puente debe de estar entre “TEMP INT” (Temperatura interna) y “TEMP RA1” (Temperatura al puerto RA1), la otra configuración es para una temperatura externa o cualquier otra señal.

O vista de una manera más sencilla, revise la siguiente imagen.
El circulo en color naranja resalta el sensor de temperatura interna, mientras que el circulo en color verde resalta el puente para medir la temperatura interna o externe.

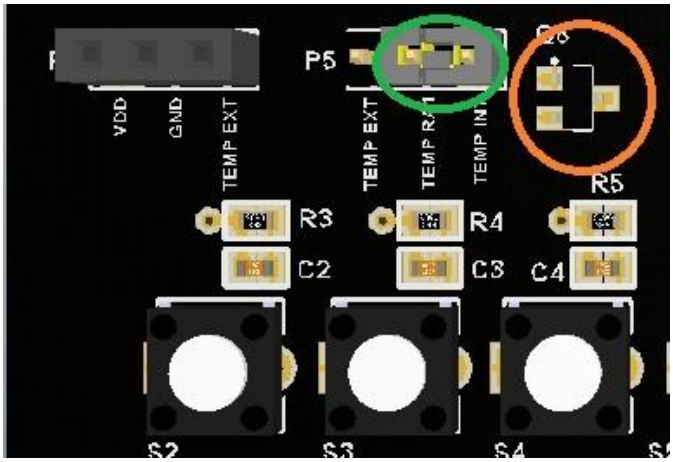


Imagen 8-8 Sensor de temperatura interna

8.1.2.1 Desplegar lectura en display 7 segmentos

Se requiere leer el valor que entrega el sensor de temperatura interno, dicho valor se deberá de ver reflejado en el display de 7 segmentos. Considerar los cálculos para obtener un valor próximo a la temperatura.
Se recomienda utilizar un multímetro que lea la temperatura o un termómetro, esto con el objetivo de comparar los resultados.

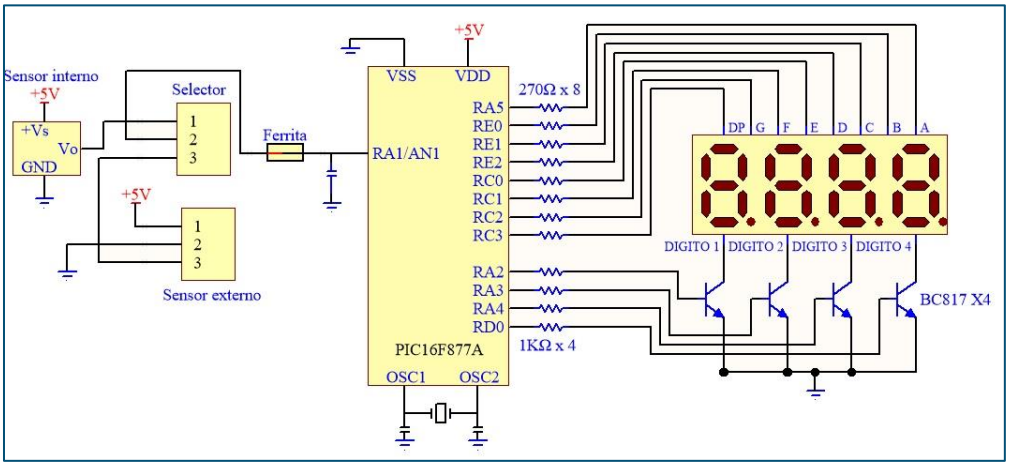


Imagen 8-9 Diagrama sensor de temperatura

Código fuente:

```
#include <htc.h>                                    // Incluimos libreria del micro a usar.
#include "analog.h"
#include "control_7segmentos.h"
#define _XTAL_FREQ 4000000                          //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

#define CONSTANTE_LM62 480
#define RESOLUCION_LM62 15.6
```

```
unsigned char contadorDisplay = 0;
char buffer[80];

int adc = 0;
int Vin = 0;
int temperatura = 0;


float calcular_Temperatura (int valor);
// PROGRAMA PRINCIPAL
void main (void){
   TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
   TRISC = 0x10;          //Solo como entrada RC4, los demas salidas
   TRISD = 0x0E;          //Entradas RD1-RD3, las demas salidas
   TRISE = 0X00;          //Todo como salidas

   apagarDigito();
   ADC_Init();            //Iniciamos el ADC
   // Configuración  del TIMER 0 para producir interrupciones cada 50 ms.
   OPTION_REG=0xC7;              //TIMER0 como temporizador con predivisor P=256
   TMR0=60;                                 //Para que Td de TMR0 sean 50ms y empiece el TMR0
   GIE=1;                                   //Habilitar interrupciones
   T0IE=1;                                  // Habilitar interrupción de TMR0
   T0IF=0;                                  //Desactivar el FLAG de TMR0

   float x = 0;
   while(1){
      adc = ADC_Read(1);            //Leer el valor del canal AN0
      x = adc * (RESOLUCION_ADC /1000);
      Vin = x *1000;
      temperatura = (calcular_Temperatura(Vin))*10;
      manejadorDelValor (temperatura);
   }
   return;
}

float calcular_Temperatura (int valor){
   float temp = 0;
   temp = (valor - CONSTANTE_LM62)/RESOLUCION_LM62;
   return temp;
}

// INTERRUPCIÓN por Timer0
void __interrupt () isr(void){
        if(T0IF){
      T0IF=0;          //Desactivar el FLAG de TMR0
      TMR0=60;         //Recargar TMR0
      if(contadorDisplay == 0){      //Contador = 0 Mostrar digito 1
         mostrarUnidades(unidades);
      }
      if(contadorDisplay == 1){      //Contador = 1 Mostrar digito 2
         mostrarDecenas(decenas);//Contador = 1 Mostrar digito 2
         PORTC = PORTC + PUNTO_DECIMAL;
      }
      if(contadorDisplay == 2 ){     //Contador = 2 Mostrar digito 3
         mostrarCentenas(centenas);
      }
      if(contadorDisplay == 3){      //Contador = 3 Mostrar digito 4
         mostrarMillares(millares);
      }
      contadorDisplay ++;     //Contador del digito a mostrar
      if(contadorDisplay >= 3)    //Si el contador es mayor o igual a 4, se reinicia
         contadorDisplay = 0;
   }
}
```

8.1.2.2 Desplegar lectura en display LCD

Usar el display LCD para mostrar el valor de la temperatura

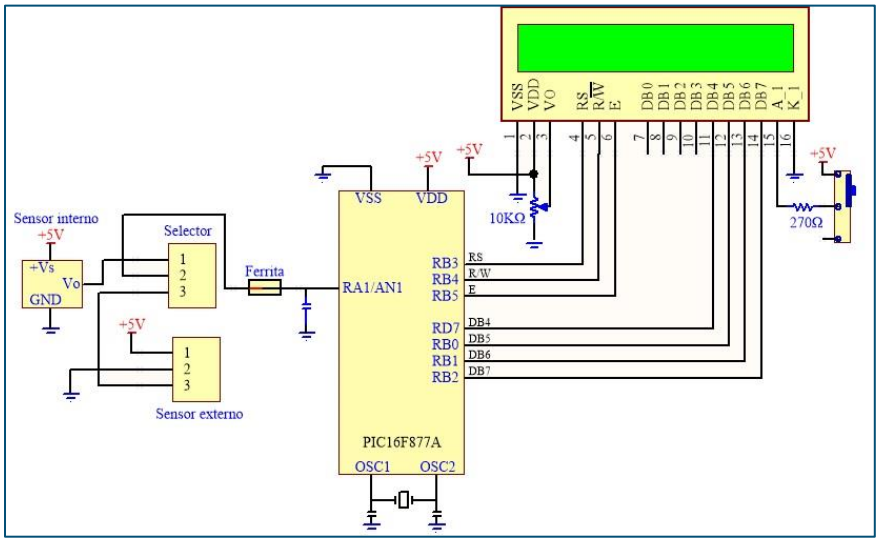


Imagen 8-10 Sensor de temperatura y LCD

Código fuente:

```
#include <htc.h>                                        // Incluimos libreria del micro a usar.
#include "pin_config.h"            // Incluimos libreria de la configuracion de puertos
#include "analog.h"
//#include "control_7segmentos.h"
#include "lcd.h"
#include <stdio.h>
#define _XTAL_FREQ 4000000                              //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

#define CONSTANTE_LM62 480
#define RESOLUCION_LM62 15.6
unsigned char contadorDisplay = 0;
char buffer[80];

int adc = 0;
int Vin = 0;
float temperatura = 0;
float x = 0;
int d;
int f;

float calcular_Temperatura (int valor);
// PROGRAMA PRINCIPAL
void main (void){
    Lcd_Init();                 //Inicializamos la LCD
    Lcd_Cmd (LCD_CLEAR);        //Limpiamos la pantalla
    Lcd_Cmd (LCD_CURSOR_OFF);
    Lcd_Out (1,5,"Hola");       //Colocar el cursor en fila 1 columna 3, escribir Hola
    Lcd_Out (2,5, "MCU");       //Colocar el cursor en fila 2 columna 5, escribir Mundo
    __delay_ms(1000);
    Lcd_Cmd (LCD_CLEAR);        //Limpiamos la pantalla

    ADC_Init();                 //Iniciamos el ADC


    while(1){
        adc = ADC_Read(1);              //Leer el valor del canal AN0
        x = adc * (RESOLUCION_ADC /1000);
        Vin = x *1000;
```

```
        x = (Vin - CONSTANTE_LM62)/RESOLUCION_LM62;

        d = (int)x;
        x = (x -d) * 100;
        f = (int)x;
        sprintf(buffer,"Temp = %d.%d C",d,f);
        Lcd_Out2(2,0,buffer);               //Mostrar el valor en la fila 2 columna 5
    }
    return;
}
```

8.1.3 Lectura de dos señales analógicas

Se pretende leer dos señales analógicas individualmente, una señal será por el canal AN0 el cual está conectado al Trimpot, y la otra señal será por el canal AN1 el cual recibe la señal de nuestro sensor de temperatura. Mostrar los resultados en el display LCD.
Usar los cálculos correspondientes para obtener los resultados finales.

Código fuente:

```
#include <htc.h>                                        // Incluimos libreria del micro a usar.
#include "pin_config.h"            // Incluimos libreria de la configuracion de puertos
#include "analog.h"
//#include "control_7segmentos.h"
#include "lcd.h"
#include <stdio.h>
#define _XTAL_FREQ 4000000                              //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

#define CONSTANTE_LM62 480
#define RESOLUCION_LM62 15.6
unsigned char contadorDisplay = 0;
char buffer[80];

int adc = 0;
int Vin = 0;
float temp= 0;
float x = 0;
int d;
int f;


float calcular_Temperatura (int valor);
int float_to_int (float valor);
// PROGRAMA PRINCIPAL
void main (void){
    Lcd_Init();               //Inicializamos la LCD
    Lcd_Cmd (LCD_CLEAR);        //Limpiamos la pantalla
    Lcd_Cmd (LCD_CURSOR_OFF);
    Lcd_Out (1,5,"Hola");       //Colocar el cursor en fila 1 columna 3, escribir Hola
    Lcd_Out (2,5, "MCU");       //Colocar el cursor en fila 2 columna 5, escribir Mundo
    __delay_ms(1000);
    Lcd_Cmd (LCD_CLEAR);        //Limpiamos la pantalla

    ADC_Init();                 //Iniciamos el ADC


    while(1){
        float voltaje = 0;
        f = 0;
        d = 0;
        adc = ADC_Read(0);              //Leer el valor del canal AN0
        voltaje = adc * (RESOLUCION_ADC /1000); //convertimos el valor del ADC a voltaje
        d = (int) voltaje;              //Asignamos el valor entero a la variable d
        f = (int) ((voltaje - d)*100);  //Restamos el valor entero y multiplicamos los decimales por 100
                                        //Asignamos el valor entero a la variable f

        Lcd_Out(1,0,"Voltaje = ");      //Mostramos el mensaje en la pantalla
        sprintf(buffer,"%d.%d V",d,f);    //Agregamos los valores d y f al buffer
        Lcd_Out2(1,10,buffer);                          //Mostrar el valor en la fila 1 columna 10

        adc = ADC_Read(1);              //Leer el valor del canal AN1
        x = adc * (RESOLUCION_ADC /1000);   //Convertimos el valor del ADC a voltaje
        Vin = x *1000;                  //convertimos volts a mVolts
        temp = (Vin - CONSTANTE_LM62)/RESOLUCION_LM62;  //Obtenemos el valor de la temperatura
        d = (int) temp;                 //Asignamos el valor entero a la variable d
        f = (int) ((temp - d)*100);     //Restamos el valor entero y multiplicamos los decimales por 100
```

```
                                    //Asignamos el valor entero a la variable f
        sprintf(buffer,"Temp = %d.%d C",d,f);   //Asignamos valores al buffer
        Lcd_Out2(2,0,buffer);               //Mostrar el valor en la fila 2 columna 0
    }
    return;
}
```

8.1.4 Lectura de sensor externo

Ya conocemos como leer lecturas analógicas, pero ahora leeremos lecturas de algún sensor externo, este sensor puede ser cualquiera, ya sea de presión, temperatura, humedad, sónicos, etc.

En este ejemplo usaremos un sensor de temperatura externa, un LM35, para conectar cualquier sensor ver siguiente imagen, el sensor debe de ir en el Header P6 (circulo anaranjado), identificando la alimentación y la entrada del sensor. Para leer dicho sensor es necesario cambiar de posición el puente del Header P5 (circulo verde), el puente debe de ir entre TEMP EXT y TEMP RA1.

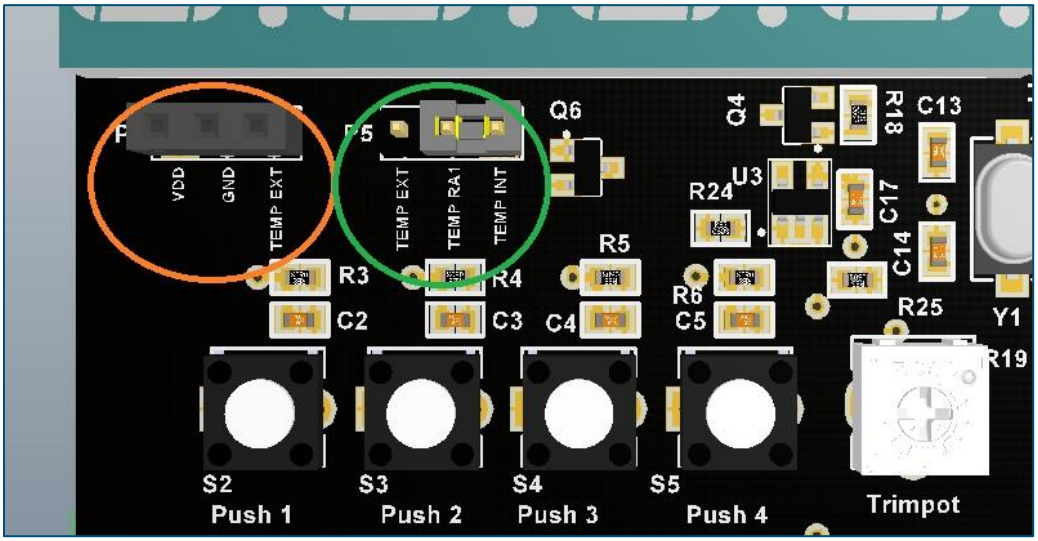


Imagen 8-11 Sensor externo

Con estos cambios el sensor de temperatura interna quedara fuera para que pueda trabajar el sensor externo.

Dependiendo del sensor que se coloque tendremos que revisar la resolución o ganancia que tenga el sensor a usa, para este ejemplo el LM35 cuenta con una resolución de 10mV/ºC, en comparación con los 15.6mV/ºC con el sensor interno.

8.1.4.1 Desplegar lectura en display LCD

Leer la señal del sensor externo y mostrarlo en el display LCD.

Código fuente:

```
#include <htc.h>                                        // Incluimos libreria del micro a usar.
#include "pin_config.h"          // Incluimos libreria de la configuracion de puertos
#include "analog.h"
//#include "control_7segmentos.h"
#include "lcd.h"
#include <stdio.h>
#define _XTAL_FREQ 4000000                              //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

#define RESOLUCION_SENSOR_EXTERNO 10
char buffer[80];

int adc = 0;
int Vin = 0;
float temp= 0;
```

```
float x = 0;
int d;
int f;


float calcular_Temperatura (int valor);
int float_to_int (float valor);
// PROGRAMA PRINCIPAL
void main (void){
    Lcd_Init();                 //Inicializamos la LCD
    Lcd_Cmd (LCD_CLEAR);        //Limpiamos la pantalla
    Lcd_Cmd (LCD_CURSOR_OFF);
    Lcd_Out (1,5,"Hola");       //Colocar el cursor en fila 1 columna 3, escribir Hola
    Lcd_Out (2,5, "MCU");       //Colocar el cursor en fila 2 columna 5, escribir Mundo
    __delay_ms(1000);
    Lcd_Cmd (LCD_CLEAR);        //Limpiamos la pantalla

    ADC_Init();                 //Iniciamos el ADC


    while(1){
        adc = ADC_Read(1);                  //Leer el valor del canal AN1
        x = adc * (RESOLUCION_ADC /1000);   //Convertimos el valor del ADC a voltaje
        Vin = x *1000;                      //convertimos volts a mVolts
        temp = (Vin)/RESOLUCION_SENSOR_EXTERNO;  //Obtenemos el valor de la temperatura
        d = (int) temp;                 //Asignamos el valor entero a la variable d
        f = (int) ((temp - d)*100);     //Restamos el valor entero y multiplicamos los decimales por 100
                                        //Asignamos el valor entero a la variable f
        sprintf(buffer,"Temp = %d.%d C",d,f);   //Asignamos valores al buffer
        Lcd_Out2(2,0,buffer);               //Mostrar el valor en la fila 2 columna 0
    }
    return;
}
```

9 Modulación de Anchura de Pulsos (PWM)

9.1 Módulo PWM

La generación de señales PWM es una herramienta vital en cada arsenal de ingenieros integrados, son muy útiles para muchas aplicaciones como controlar la posición del servomotor, cambiar algunos circuitos integrados electrónicos de potencia en convertidores / inversores e incluso para un simple control de brillo de LED. En los microcontroladores PIC, las señales PWM se pueden generar utilizando los módulos Comparar, Capturar y PWM (CCP) configurando los Registros requeridos.

El PIC16F877 puede generar señales PWM solo en los pines RC1 y RC2, si usamos los módulos CCP. Pero podemos encontrar situaciones en las que necesitemos más pines para tener la funcionalidad PWM. En estos escenarios, podemos programar los pines GPIO para producir señales PWM utilizando módulos de temporizador. De esta manera podemos generar tantas señales PWM con cualquier pin requerido. También hay otros trucos de hardware como el uso de un IC multiplexor, pero ¿por qué invertir en hardware cuando se puede lograr lo mismo mediante la programación? Así que aquí aprenderemos cómo convertir un pin PIC GPIO en un pin PWM.

9.1.1 Generación de pulsos

Antes de entrar en detalles, repasemos un poco qué son las señales PWM. La modulación de ancho de pulso (PWM) es una señal digital que se usa con mayor frecuencia en los circuitos de control. Esta señal se establece en alto (5v) y bajo (0v) en un tiempo y velocidad predefinidos. El tiempo durante el cual la señal permanece alta se denomina "tiempo de activación " y el tiempo durante el cual la señal permanece baja se denomina "tiempo de inactividad". Hay dos parámetros importantes para un PWM como se describe a continuación:

Ciclo de trabajo del PWM (Duty Cycle)

El porcentaje de tiempo en el que la señal PWM permanece ALTA (a tiempo) se llama ciclo de trabajo (duty cycle). Si la señal está siempre encendida, está en un ciclo de trabajo del 100% y si siempre está apagada, es un ciclo de trabajo del 0%.

Ciclo de trabajo = tiempo de encendido / (tiempo de encendido + tiempo de apagado)

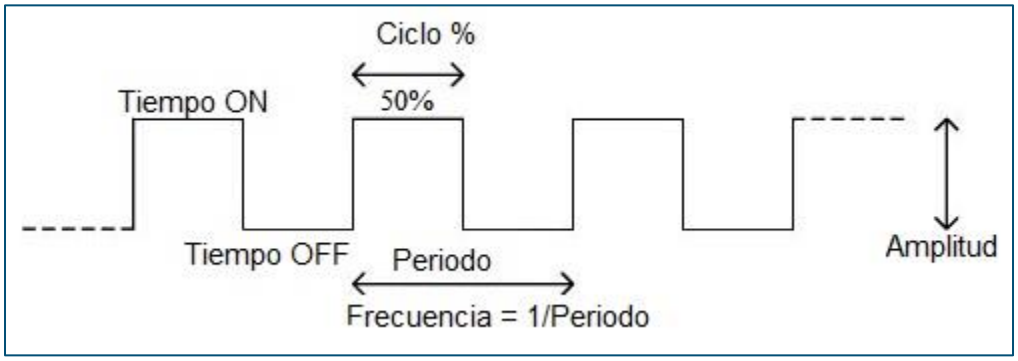


Imagen 9-1 Ciclo de trabajo

Frecuencia de un PWM

La frecuencia de una señal PWM determina qué tan rápido un PWM completa un período. Un período es el ENCENDIDO y APAGADO completo de una señal PWM como se muestra en la figura anterior.

Cálculo del ciclo de trabajo para PWM

Para generar una señal PWM en un pin GPIO, simplemente tenemos que encenderlo y apagarlo durante un tiempo predefinido. Pero no es tan simple como parece. Este tiempo de encendido y apagado debe ser preciso para cada ciclo, por lo que simplemente no podemos usar las funciones de retardo, por lo que empleamos un módulo de temporizador y usamos las interrupciones del temporizador. También tenemos que considerar el ciclo de trabajo y la frecuencia de la señal PWM que generamos. Los siguientes nombres de variables se utilizan en el programa para definir los parámetros.

Nombre de la variable	Se refiere a
PWM_Frequency	Frecuencia de la señal PWM
T_TOTAL	Tiempo total necesario para un ciclo completo de PWM
T_ON	A tiempo de la señal PWM
T_OFF	Tiempo de apagado de la señal PWM
Duty_cycle	Ciclo de trabajo de la señal PWM

Así que ahora, hagamos los cálculos.

Estas son las fórmulas estándar donde la frecuencia es simplemente el recíproco del tiempo. El valor de la frecuencia debe ser decidido y establecido por el usuario en función de los requisitos de su aplicación.

T_TOTAL = (1 / PWM_ Frequency)

Cuando el usuario cambia el valor del ciclo de trabajo, nuestro programa debe ajustar automáticamente el tiempo T_ON y el tiempo T_OFF de acuerdo con eso. Por lo tanto, las fórmulas anteriores se pueden usar para calcular T_ON en función del valor de Duty_Cycle y T_TOTAL.

T_ON = (Duty_Cycle * T_TOTAL) / 100

Dado que el tiempo total de la señal PWM para un ciclo completo será la suma del tiempo de encendido y apagado. Podemos calcular el tiempo de inactividad T_OFF como se muestra arriba.

T_OFF = T_TOTAL - T_ON

Con estas fórmulas en mente, podemos comenzar a programar el microcontrolador PIC. El programa involucra el módulo de temporizador PIC y el módulo PIC ADC para crear una señal PWM basada con un ciclo de trabajo variable de acuerdo con el valor ADC del POT.

9.1.2 Manejo de LED RGB

Para este caso usaremos un Timer de 0.1ms por el TMR0, usando un Prescaler de 256 y el cristal de 16MHz.

RegValue = $256 - (\frac{Retraso * Fosc}{Prescaler * 4}) = 256 - \left(\frac{0.1ms * 16MHz}{256 * 4}\right) = 256 - 1.95 = 254.04 \approx 254$

Dentro del ciclo while infinito, tenemos que calcular el valor de tiempo de encendido (T_ON) del ciclo de trabajo. El tiempo de encendido y el ciclo de trabajo varían según la posición del POT, por lo que lo hacemos repetidamente dentro del ciclo while como se muestra a continuación. 0.0976 es el valor que hay que multiplicar por 1024 para obtener 100 y para calcular T_ON lo hemos multiplicado por 10 para obtener el valor en milisegundos.

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "timers.h"
#include "adc.h"
#define PWM_Frequency 0.05 // en KHz (50Hz)
// DEFINICION DE VARIABLES
int POT_val; // variable para almacenar el valor de ADC
int count; // variable de temporizador
int T_TOTAL = (1 / PWM_Frequency) / 10; // calcular el tiempo total a partir de la frecuencia (en milisegundos)) // 2msec
int T_ON = 0; // valor de tiempo
int Duty_cycle; // Valor del ciclo de trabajo

// PROGRAMA PRINCIPAL
void main(void) {
    /*Configuración de los puertos */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 1;      // Encender ánodo
    __delay_ms(500);       // Esperamos 500ms
```

```
    ADC_Init();               //Iniciamos el ADC
    // Configuración  del TIMER 0 para producir interrupciones cada 0.1 ms.
    OPTION_REG = 0xC7;              //TIMER0 como temporizador con predivisor P=256
    TMR0 = 254;                             //Para que Td de TMR0 sean 0.1ms y empiece el TMR0
    GIE=1;                                  //Habilitar interrupciones
    T0IE=1;                                 // Habilitar interrupción de TMR0
    T0IF=0;
    while(1){
        POT_val = ADC_Read(0);              //Leer el valor del canal AN0
        Duty_cycle = (POT_val * 0.0976);
        T_ON = ((Duty_cycle * T_TOTAL)*10 / 100); // Calcular Tiempo_ON usando la unidad de fórmula en milisegundos
        __delay_ms(100);
    }
    return;
}


// INTERRUPCIÓN por Timer0
void __interrupt () isr(void){
            if(T0IF){
        T0IF=0;           //Desactivar el FLAG de TMR0
        PORTDbits.RD6 = ~PORTDbits.RD6; //Cambiar estado de LED Azul
        TMR0 = 254;       //Recargar TMR0 0.1ms
        count++;          // Contar incrementos por cada 0.1ms -> count / 10 dará el valor del conteo en ms
    }
    if (count <= (T_ON) ){
        PORTDbits.RD4 = 0;      // Encender LED Rojo
        PORTDbits.RD5 = 1;      // Apagar LED Verde
    }
    else{
        PORTDbits.RD4 = 1;      // Apagar LED Rojo
        PORTDbits.RD5 = 0;      // Apagar LED Verde
    }
    if (count >= (T_TOTAL*10) )
        count=0;
}
```

10 Módulos de comunicación

En este capítulo se presenta los módulos de comunicación serie que integra el microcontrolador PIC16F877A. Este microcontrolador integra dos módulos de comunicaciones serie independientes que utilizan los terminales asociados al PORTC. Estos módulos son el módulo **USART** (Universal Synchronous/Asynchronous Receiver Transmitter ó Transmisor-Receptor Síncrono/Asíncrono Universal) y el módulo **MSSP** (Master Synchronous Serial Port o Puerto
Serie Síncrono Maestro).

10.1 Módulo USART

El módulo transmisor receptor universal síncrono/asíncrono (USART) es uno de los dos módulos de E/S en serie. También se conoce como interfaz de comunicaciones en serie o SCI (Serial Communications Interface).
El USART se puede configurar como un sistema asíncrono full-duplex que puede comunicarse con dispositivos periféricos, como terminales CRT y computadoras personales, o se puede configurar

como un sistema síncrono half-duplex que puede comunicarse con dispositivos periféricos, convertidores análogos/digital o digital/análogo, memorias EEPROM, etc.
Este módulo permite la comunicación del microcontrolador utilizando los protocolos RS232, RS422 y RS485, siempre y cuando se adapten los niveles de señal utilizados por el microcontrolador normalmente 5V.

El USART se puede configurar en los siguientes modos:

- Asíncrono (dúplex completo (full-duplex))
- Sincrónico: maestro (semidúplex (half-duplex))
- Sincrónico: esclavo (semidúplex (half-duplex))

Dado que le módulo USAR es un módulo periférico integrado en el microcontrolador, este requiere una configuración inicial para su funcionamiento. Una vez configurado, el módulo USART funcionara de forma autónoma requiriendo la atención del microcontrolador sólo cuando sea necesario, ya sea mediante interrupciones o únicamente activando los indicadores “flags”.
La configuración de los módulos de transmisor y receptor se realiza mediante los registros TXSTA, RCSTA y SPBRG, mientras que el envío y recepción de datos se realiza utilizando los registros TXREG y RCREG respectivamente.

Con respecto al módulo transmisor, el bit TXIF Del registro PIR1 se pone a ´0´cada vez que escribimos un nuevo dato sobre el registro TXREG y, pasa a ´1´cuando el contenido de este buffer se copia sobre el TSR Para dar inicio a la transmisión del nuevo carácter. En este momento, si se desea, se puede provocar una interrupción al controlador. el bit TRMT del registro TXSTA, indica el estado del registro desplazamiento TSR. se pone a ´1´cuando se encuentra vacío, sin nada que transmitir.

pueden realizarse transmisiones de 9 bits por cada carácter. En este caso, los 8 bits de menos peso se cargan en el registro TXREG mientras que el noveno (bit 8) se carga en el bit TX9D del registro TXSTA. Esta característica se habilita mediante el bit TX9 del mismo registro.

La siguiente tabla muestra los registros asociados con PIC16F877A UART.[8]

Tabla 10-1 Registros UART

Registro	Descripción
TXSTA	Transmisión de estado y registro de control
RCSTA	Recibir estado y registro de control
SPBRG	Generador de velocidad en baudios USART
TXREG	Registro de transmisión de USART. Contiene los datos que se transmitirán en UART

[8] **https://exploreembedded.com/wiki/Serial_Communication_with_PIC16F877A**

RCREG	Registro de transmisión de USART. Contiene los datos recibidos de UART

Ahora veamos cómo configurar los registros individuales para la comunicación UART.

Tabla 10-2 Registros UART

TXSTA: ESTADO DE TRANSMISIÓN Y REGISTRO DE CONTROL							
7	6	5	4	3	2	1	0
CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D

bit 7 **CSRC**: Bit de selección de fuente de reloj
Modo asincrónico:
No importa
Modo sincrónico:
1 = Modo maestro (reloj generado internamente desde BRG)
0 = modo esclavo (reloj de fuente externa)

bit 6 **TX9**: Habilitar transmisión de 9 bits
1 = Selecciona transmisión de 9 bits
0 = Selecciona transmisión de 8 bits

bit 5 **TXEN**: Transmitir bit habilitado
1 = Transmisión habilitada
0 = Transmisión inhabilitada
Nota: SREN / CREN anula TXEN en el modo Sync.

bit 4 **SYNC**: bit de selección de modo USART
1 = modo síncrono
0 = modo asíncrono

bit 3 **No implementado**: leer como '0''

bit 2 **BRGH**: bit de selección de alta velocidad en baudios
Modo asincrónico:
1 = alta velocidad
0 = velocidad baja
Modo sincrónico:
No se usa en este modo.

bit 1 **TRMT**: Transmitir el bit de estado del registro de cambio
1 = TSR vacío
0 = TSR lleno

bit 0 **TX9D**: noveno bit de transmisión de datos, puede ser un bit de paridad

Tabla 10-3 RCSTA

RCSTA: RECIBIR ESTADO Y REGISTRO DE CONTROL							
7	6	5	4	3	2	1	0

SPEN	RX9	SREN	CREN	AÑADIR	FERR	OERR	RX9D

bit 7 **SPEN**: bit de habilitación del puerto serie
1 = puerto serie habilitado (configura los pines RC7 / RX / DT y RC6 / TX / CK como pines del puerto serie)
0 = puerto serie deshabilitado

bit 6 **RX9**: bit de habilitación de recepción de 9 bits
1 = Selecciona la recepción de 9 bits
0 = Selecciona la recepción de 8 bits

bit 5 **SREN**: bit de habilitación de recepción única
Modo asincrónico:
No importa
Modo síncrono - Maestro:
1 = Habilita la recepción única
0 = Desactiva la recepción única
Este bit se borra una vez finalizada la recepción.
Modo síncrono - esclavo:
No importa

bit 4 **CREN**: bit de habilitación de recepción continua
Modo asincrónico:
1 = habilita la recepción continua
0 = Desactiva la recepción continua
Modo sincrónico:
1 = Habilita la recepción continua hasta que se borra el bit de habilitación CREN (CREN anula SREN)
0 = Desactiva la recepción continua

bit 3 **ADDEN**: bit de habilitación de detección de dirección
Modo asíncrono de 9 bits (RX9 = 1):
1 = Habilita la detección de direcciones, habilita la interrupción y la carga del búfer de recepción cuando RSR <8>
Está establecido
0 = Deshabilita la detección de dirección, se reciben todos los bytes y el noveno bit se puede usar como bit de paridad

bit 2 **FERR**: bit de error de trama
1 = Error de encuadre (se puede actualizar leyendo el registro RCREG y recibir el siguiente byte válido)
0 = Sin error de encuadre

bit 1 **OERR**: bit de error de saturación
1 = Error de rebasamiento (se puede borrar borrando el bit CREN)
0 = Sin error de rebasamiento

bit 0 **RX9D**: noveno bit de datos recibidos (puede ser un bit de paridad, pero debe ser calculado por el firmware del usuario)

Cálculo de la tasa de baudios

El criterio principal para la comunicación UART es su velocidad en baudios. Ambos dispositivos Rx/Tx deben configurarse a la misma velocidad en baudios para una comunicación exitosa. Esto se puede lograr mediante el registro SPBRG. SPBRG es un registro de 8 bits que controla la generación de velocidad en baudios.
Dada la velocidad en baudios y el FOSC deseados, el valor entero más cercano para el registro SPBRG se puede calcular usando la siguiente fórmula.

Tabla 10-4 Fórmula de velocidad en baudios

FÓRMULA DE VELOCIDAD EN BAUDIOS		
SYN	BRGH = 0 (baja velocidad)	BRGH = 1 (alta velocidad)
0	(Asíncrono) BaudRate= FOSC / (64 (X + 1))	BaudRate= FOSC / (16 (X + 1))
1	(Sincrónico) BaudRate= FOSC / (4 (X + 1))	N/A

X = valor en SPBRG (0 a 255)

BRGH = 1 de alta velocidad

$$SPBRG = (Fosc/(16 * BaudRate)) - 1$$

BRGH = 0 de baja velocidad

$$SPBRG = (Fosc/(64 * BaudRate)) - 1$$

Puede resultar ventajoso utilizar la velocidad en baudios alta (BRGH = 1) incluso para relojes en baudios más lentos. Esto se debe a que la ecuación FOSC / (16 (X + 1)) puede reducir el error de velocidad en baudios en algunos casos. La siguiente tabla muestra la lista de BaudRates estándar y los valores SPBRG.

BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz			Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-	0.300	0	207	0.3	0	191
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64	2.404	0.17	25	2.4	0	23
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15	8.929	6.99	6	9.6	0	5
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4	31.250	8.51	1	28.8	0	1
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4	-	-	-	-	-	-
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2	62.500	8.51	0	57.6	0	0
HIGH	1.221	-	255	0.977	-	255	0.610	-	255	0.244	-	255	0.225	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0	62.500	-	0	57.6	-	0

BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz			Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-	1.202	0.17	207	1.2	0	191
2.4	-	-	-	-	-	-	2.441	1.71	255	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31	19.231	0.16	12	19.2	0	11
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21	27.798	3.55	8	28.8	0	7
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18	35.714	6.29	6	32.9	2.04	6
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10	62.500	8.51	3	57.6	0	3
HIGH	4.883	-	255	3.906	-	255	2.441	-	255	0.977	-	255	0.9	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0	250.000	-	0	230.4	-	0

10.1.1 Configurar el módulo USART en modo asíncrono

En este ejemplo se va a configurar el módulo USART para comunicaciones en modo asíncrono de forma genérica. Para ello se creará una macro que permita introducir los valores necesarios en los registros de configuración.

Configuración:

Para la creación de la macro Será necesario definir diferentes constantes que determinen el modo de trabajo del módulo y que permitan ajustar los registros de configuración TXSTA, RCSTA y SPBRG. En este caso se definirá una máscara que ajuste el modo de trabajo en alta velocidad HIGH_SPEED = 0x04 y otra que ajuste el modo de funcionamiento 9 bits NINE_BITS = 0x40. Estas máscaras se utilizarán junto con la función OR (|) para ajustar los registros de configuración TXSTA Y RCSTA. El registro SPBRG que determinará la velocidad del módulo obtiene su velocidad a partir del oscilador principal y el modo de funcionamiento a alta o baja velocidad a partir de las siguientes fórmulas:

$$Baja\ velocidad\ \rightarrow SPBRG = \frac{F_{osc}}{64(X+1)}$$

$$Alta\ velocidad\ \rightarrow SPBRG = \frac{F_{osc}}{16(X+1)}$$

Así pues, la macro para configurar el módulo USART en modo asíncrono tendrá la siguiente estructura:

#define INIT_USART (BAUD, SPEED, NINE)

Para utilizar esta función será suficiente con incluir el código fuente que aparece a continuación dentro del programa o de una forma más elegante incluir el código en un archivo de definiciones denominado usart.h y añadirlo a nuestro programa principal mediante la sentencia
#include "usart.h"
Código fuente:

```
#ifndef XC_USART_H
#define   XC_USART_H

#include <xc.h> // include processor files - each processor file is guarded.
#include "configuracion_bits.h"


// ARCHIVOS DE DEFINICIONES
#define HIGH_SPEED 0x04                         // Modo alta velocidad
#define LOW_SPEED 0                                     // Modo baja velocidad
#define NINE_BITS 0x40                          // Modo transmisión 9 bits
#define EIGHT_BITS 0                                    // Modo transmisión 8 bits
#define RX_PIN TRISC7                           // Define el pin RC7 como pin para recepción de datos
#define TX_PIN TRISC6                           // Define el pin RC6 como pin para transmisión de datos

/* MACRO PARA CONFIGURAR EL MÓDULO USART EN MODO ASÍNCRONO*/
// BAUD: Velocidad de comunicaciones (9600, 19200, 28800, etc.)
// SPEED: Modo alta velocidad para el generador de baudios => TRUE = On
// NINE: Utiliza transmission de 9 bits => FALSE=8bit

void INIT_USART(int BAUD, int SPEED, int NINE); //Inicializacion del puerto UART

void putch (char ch);
void UART_TxChar (char ch);    //Envio de datos por TX
char UART_RxChar();        //Recepción de datos
char UART_Read();

#endif     /* XC_USART_H */
```

El código del archivo uart.c es el siguiente:

Código fuente:

```
#include "usart.h"
void INIT_USART(int BAUD, int SPEED, int NINE){
            RX_PIN = 1;                                             // Define los pines RC6 y RC7 para poder
            TX_PIN = 1;                                             // ser utilizados por el módulo USART
            if (SPEED)SPBRG = ((int)(_XTAL_FREQ/((16UL * BAUD) +16)));          // Ajusta la velocidad del
            else SPBRG = ((int)(_XTAL_FREQ/((64UL * BAUD) +64)));       // generador de baudios
            RCSTA = (NINE|0x90);                        // Carga la configuración del módulo receptor
            TXSTA = (SPEED|NINE|0x20);                  // Carga la configuración del módulo transmisor
}

void putch (char ch){
    while (TXIF == 0 );  // Espere hasta que el registro del transmisor se vacíe
    TXIF = 0 ;          // Borrar la bandera del transmisor
    TXREG = ch;         // carga el char que se transmitirá en el registro de transmisión
}

char  UART_RxChar(){
    while (RCIF == 0 );    // Espere hasta que se reciban los datos
    RCIF = 0 ;            // Borrar la bandera del receptor
    return (RCREG);     // Devuelve los datos recibidos a la función de llamada
}

char UART_Read(){
  while(!RCIF);
  return RCREG;
}
```

Este código solo es la configuración, el cual se agrega al archivo de definición .h, para ver probar el funcionamiento se debe de mandar a llamar la función *INIT_USART(9600,HIGH_SPEED,EIGHT_BITS); desde el programa principal y observar que valores toman los registros de configuración TXSTA, RCSTA y SPBRG.*

10.1.2 Envió de datos utilizando el módulo USART

A partir del ejemplo anterior se usará el código para configurará el módulo USART para crear un programa que se comunique con el puerto serie del ordenador personal y escriba a través de una terminal la típica frase de iniciación en la programación “¡Hola Mundo!”.

Utilizar el conector DB-9 para conectar la placa y el ordenador, la tarjeta SIRIUS P16 ya cuenta con el circuito para convertir las señales, utiliza el integrado MAX232.

Configuración:

Para la configuración del módulo USART se utilizará la función INIT_USART cuyo ejemplo ya se mostró en el capítulo anterior y la cual está incluida en la librería usart.h, a la que se la añadirá la definición de una nueva función que se encargará de enviar un byte de datos a través del módulo USART. Esta nueva función denominada void putch (char ch); tendrá como argumento un dato de tamaño byte y se encargará de transmitir el dato escribiéndolo en el registro TXREG cuando el módulo esté disponible (bit TXIF = ´1´).

El código fuente de la función se incluirá en la librería usart.c y tendrá la siguiente estructura.

```
void putch (char ch){
    while (TXIF == 0 );  // Espere hasta que el registro del transmisor se vacíe
    TXIF = 0 ;          // Borrar la bandera del transmisor
    TXREG = ch;         // carga el char que se transmitirá en el registro de transmisión
}
```

El envío de datos utilizando la función putch se realiza byte a byte, por lo que para enviar una cadena de caracteres sería necesario llamar la función de forma consecutiva por cada carácter a enviar usando un código similar al siguiente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include <stdio.h>                                  // Librería con funciones para el control de entradas y salidas
#include "usart.h"                                  // Librería para el módulo de comunicaciones USART


void main(void){
    char i;
    char a[] = { "Hola Mundo\n\r" };
            INIT_USART(9600,HIGH_SPEED,EIGHT_BITS);       // Configura el módulo USART
            while(1){
        for(i=0;  a[i]!= 0; i++ ){
            putch (a[i]);    // Transmitir el carácter
        }
        __delay_ms(2000);
              }
}
```

Aunque el envío de la cadena de caracteres de la forma indicada anteriormente es completamente válido, existe una librería en C denominada stdio.h que integra funciones que facilitan esta tarea. En particular la función printf se encarga de dar formato a una cadena de caracteres y enviarla a

través del puerto de salida llamando la función putch las veces que sean necesarias hasta completar la cadena.

En este caso se utilizará la sentencia printf("\n\r Hola Mundo");

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include <stdio.h>                                  // Librería con funciones para el control de entradas y salidas
#include "usart.h"                                  // Librería para el módulo de comunicaciones USART


void main(void){
        INIT_USART(9600,HIGH_SPEED,EIGHT_BITS);     // Configura el módulo USART
        while(1){
                printf("\n\r Hola Mundo");          // Escritura utilizando la función printf
                __delay_ms(2000);
        }
}
```

10.1.3 Recepción de datos

El siguiente ejemplo es para comprender como es que se reciben los datos de una terminal, en este ejemplo solicitaremos el nombre y la edad de una persona.

En la terminal aparecer que el PIC estará solicitando el nombre, se quedara esperando el nombre hasta que se ingrese la tecla enter, después de ser ingresada mostrara el mensaje solicitando nuestra edad y de igual forma procesará la información cuando se oprima la tecla enter.

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include <stdio.h>                                  // Librería con funciones para el control de entradas y salidas
#include "usart.h"                                  // Librería para el módulo de comunicaciones USART

void clearBuffer(unsigned char *buffer, unsigned char size);        //Funcion para limpiar buffer
void main(void){
    unsigned char buf[80],buff_edad[3];
    clearBuffer(&buf,80);
    clearBuffer(buff_edad,3);
            INIT_USART(9600,HIGH_SPEED,EIGHT_BITS);        // Configura el módulo USART
            while(1){
                        printf("\n\r Hola! ¿Como te LLamas?");                          // Mensaje de salida
        for (int i = 0; i<80; i++){             //Ciclo para guardar la cedena
            buf[i] = UART_RxChar();                                         // Almacena la respuesta del usuario en buf
            if(buf[i] == '\r'){                 //Si el mensaje recibido es '\r' indica que se oprimio la tecla Enter
                buf[i] = 0x0;
                i = 80;
            }
        }
                        printf("\n\r ¿Cual es tu edad?");        // Mensaje de salida
        for (int i = 0; i<4; i++){          //Ciclo para guardar la cedena
            buff_edad[i] = UART_RxChar();          // Almacena la respuesta del usuario en buff_edad
            if(buff_edad[i] == '\r'){       //Si el mensaje recibido es '\r' indica que se oprimio la tecla Enter
                buff_edad[i] = 0x0;
                i = 4;
            }
        }
                        printf("\n\n\r Hola %s de %s de edad.\n",buf,buff_edad);                      // Respuesta
             }
}

void clearBuffer(unsigned char *buffer, unsigned char size){
    for(int i = 0; i<size; i++){
        buffer[i]= 0;
    }
    return;
}
```

Notas:

10.2 Módulo MSSP (Master Synchronous Serial Port) "I²C"

El módulo MSSP es una interfaz de comunicaciones serie muy utilizado para comunicar el microcontrolador con dispositivos periféricos u otros microcontroladores. Entre los dispositivos que se utilizan habitualmente se pueden encontrar memorias EEPROM, registros de desplazamiento, conversores analógicos/digital, etc.

El módulo MSSP tiene asociados 3 registros:

- El registro de status SSPSTAT
- Y dos registros de control SSPCON Y SSPCON2

Los últimos dos registros determinarán el funcionamiento del mismo en modo I²C o SPI tanto funcionando en modo maestro como en modo esclavo.

Las terminales del microcontrolador asociados al módulo MSSP son las siguientes

Terminal	SPI	I²C
RC3	Serial Clock (SCK)	Serial Clock (SCL)
RC4	Serial Data In (SDI)	Serial Data (SDA)
RC5	Serial Data Out (SDO)	-
RA5	Slave select	-

Pero para esta placa usaremos el módulo I²C ya que la memoria EEPROM que cuenta nuestra tarjeta usa esta comunicación.

El módulo MSSP en modo I2C implementa completamente todas las funciones maestras y esclavas (incluido el soporte de llamadas generales) y proporciona interrupciones en los bits de inicio y parada en el hardware para determinar un bus libre (función multimaestro). El módulo MSSP implementa las especificaciones de modo estándar, así como el direccionamiento de 7 y 10 bits.
Se utilizan dos pines para la transferencia de datos

• Serial Clock (SCL) - RC3 / SCK / SCL
• Serial Data (SDA): RC4 / SDI / SDA

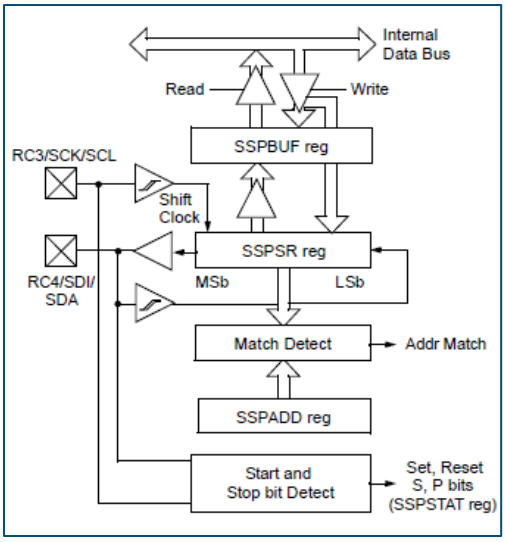


Imagen 10-1Esquema a bloques del I2C

El módulo MSSP tiene seis registros para el funcionamiento de I 2 C.
Estos son:

- Registro de control de MSSP (SSPCON)
- Registro de control 2 de MSSP (SSPCON2)
- Registro de estado de MSSP (SSPSTAT)
- Registro de búfer de transmisión / recepción en serie (SSPBUF)
- Registro de cambio de MSSP (SSPSR): no se puede acceder directamente
- Registro de direcciones MSSP (SSPADD)

SSPCON, SSPCON2 y SSPSTAT son los registros de control y estado en la operación en modo I2C. Los registros SSPCON y SSPCON2 se pueden leer y escribir. Los seis bits inferiores del SSPSTAT son de solo lectura. Los dos bits superiores de SSPSTAT son de lectura / escritura.

SSPSR es el registro de desplazamiento que se utiliza para introducir o sacar datos. SSPBUF es el registro de búfer en el que se escriben o se leen los bytes de datos.
El registro SSPADD contiene la dirección del dispositivo esclavo cuando el SSP está configurado en el modo esclavo I2C. Cuando el SSP está configurado en modo maestro, los siete bits inferiores de SSPADD actúan como el valor de recarga del generador de velocidad en baudios.

Tabla 10-5 SSPSTAT

SSPSTAT: REGISTRO DE ESTADO MSSP (MODO I^2C)							
7	6	5	4	3	2	1	0
SMP	CKE	$D/\bar{A}$	P	S	$R/\bar{W}$	UA	BF

bit 7 **SMP**: bit de control de velocidad de respuesta
En modo maestro o esclavo:
1 = Control de velocidad de respuesta deshabilitado para el modo de velocidad estándar (100 kHz y 1 MHz)
0 = Control de velocidad de respuesta habilitado para modo de alta velocidad (400 kHz)

bit 6 **CKE**: bit de selección de SMBus

En modo maestro o esclavo:

1 = Habilitar entradas específicas de SMBus

0 = Deshabilitar entradas específicas de SMBus

bit 5 $\boldsymbol{D/\overline{A}}$: bit de datos/$\overline{dirección}$

En modo maestro:

Reservado.

En modo esclavo:

1 = Indica que el último byte recibido o transmitido fueron datos

0 = Indica que el último byte recibido o transmitido fue dirección

bit 4 **P**: Detener bit

1 = Indica que se ha detectado un bit de parada por última vez

0 = El último bit de parada no se detectó

Nota: Este bit se borra en Reset y cuando SSPEN se borra.

bit 3 **S**: bit de inicio

1 = Indica que se ha detectado un bit de inicio por última vez

0 = El último bit de inicio no se detectó

Nota: Este bit se borra en Reset y cuando SSPEN se borra.

bit 2 $\boldsymbol{R/\overline{W}}$: lectura / escritura de información de bits (solo modo I 2 C)

En modo esclavo:

1 = Leer

0 = escribir

Nota: Nota: Este bit contiene la información de bits R / W después de la última coincidencia de direcciones. Este bit solo es válido a partir de la coincidencia de direcciones con el siguiente bit de inicio, bit de parada o bit no ACK.

En modo maestro:

1 = Transmisión en curso

0 = La transmisión no está en curso

Nota: O si este bit con SEN, RSEN, PEN, RCEN o ACKEN indicará si el MSSP está en modo inactivo.

bit 1 **UA**: Dirección de actualización (solo modo esclavo de 10 bits)

1 = Indica que el usuario necesita actualizar la dirección en el registro SSPADD

0 = No es necesario actualizar la dirección

bit 0 **BF**: bit de estado de búfer completo

En modo de transmisión:

1 = Recepción completa, SSPBUF está lleno

0 = Recepción no completa, SSPBUF está vacío

En modo de recepción:

1 = Transmisión de datos en curso (no incluye los bits ACK y Stop), SSPBUF está lleno

0 = Transmisión de datos completa (no incluye los bits ACK y Stop), SSPBUF está vacío

Tabla 10-6 SSPCON1

SSPCON1: REGISTRO 1 DE CONTROL MSSP (MODO I^2C)							
7	6	5	4	3	2	1	0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

bit 7 **WCOL**: bit de detección de colisión de escritura

En el modo de transmisión maestra:

1 = Se intentó escribir en el registro SSPBUF mientras las condiciones I2C no eran válidas para que se iniciara una transmisión. (Debe borrarse en el software).

0 = sin colisión

En el modo de transmisión esclava:

1 = El registro SSPBUF se escribe mientras aún está transmitiendo la palabra anterior. (Debe borrarse en el software).

0 = sin colisión

En el modo de recepción (modos maestro o esclavo):

Esta es una parte de "no me importa".

bit 6 **SSPOV**: bit de indicador de desbordamiento de recepción

En modo de recepción:

1 = Se recibe un byte mientras el registro SSPBUF todavía contiene el byte anterior. (Debe borrarse en el software).

0 = sin desbordamiento

En modo de transmisión:

Este es un bit "no importa" en el modo de transmisión.

bit 5 **SSPEN**: bit de habilitación del puerto serie síncrono

1 = Habilita el puerto serie y configura los pines SDA y SCL como pines del puerto serie

0 = Desactiva el puerto serie y configura estos pines como pines del puerto de E / S

Nota: Cuando está habilitado, los pines SDA y SCL deben configurarse correctamente como entrada o salida.

bit 4 **CKP**: Bit de control de liberación de SCK

En modo esclavo:

1 = Liberar reloj

0 = Mantiene el reloj bajo (estiramiento del reloj). (Se utiliza para garantizar el tiempo de configuración de los datos).

En modo maestro:

No se usa en este modo.

bit 3-0 **SSPM3: SSPM0**: bits de selección de modo de puerto serie síncrono

1111 = I^2C Modo esclavo, dirección de 10 bits con interrupciones de bit de inicio y parada habilitadas

1110 = I^2C Modo esclavo, dirección de 7 bits con interrupciones de bit de inicio y parada habilitadas

1011 = I^2C Modo maestro controlado por firmware (esclavo inactivo)

1000 = I^2C Modo maestro, reloj = FOSC / (4 * (SSPADD + 1))

0111 = I^2C Modo esclavo, dirección de 10 bits

0110 = I^2C Modo esclavo, dirección de 7 bits

Nota: Las combinaciones de bits que no se enumeran específicamente aquí están reservadas o implementadas en Solo modo SPI.

SSPADD

Cuando el SSP está configurado en modo maestro, los siete bits inferiores de SSPADD actúan como el valor de recarga del generador de velocidad en baudios.

Para conocer el valor del SSPADD debemos de usar la formula del I²C modo maestro el cual es el siguiente.

$$reloj = \frac{FOSC}{4 * SSPADD + 1}$$

Donde

Reloj. – Es la velocidad

FOSC. – Es el valor del oscilador interno (16MHz)

Por lo tanto, despejamos la variable SSPADD y obtenemos lo siguiente

$$SSPADD = \frac{FOSC}{4 * reloj} - 1$$

Para este ejercicio usaremos un reloj de 100kHz y el cristal de 16MHz.

$$SSPADD = \frac{16MHZ}{4 * 100KHz} - 1 = 40 - 1 = 39 \rightarrow 0x27$$

Tabla 10-7 SSPCON2

SSPCON2: REGISTRO 2 DE CONTROL MSSP (MODO I²C)							
7	6	5	4	3	2	1	0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN

bit 7 **GCEN**: bit de habilitación de llamada general (solo en modo esclavo)

1 = Habilitar interrupción cuando se reciba una dirección de llamada general (0000h) en el SSPSR

0 = Dirección de llamada general deshabilitada

bit 6 **ACKSTAT**: bit de estado de reconocimiento (solo en modo de transmisión maestra)

1 = No se recibió confirmación del esclavo

0 = Se recibió confirmación del esclavo

bit 5 **ACKDT**: Reconocimiento de bits de datos (solo modo de recepción maestra)

1 = No reconocer

0 = Reconocer

Nota: Valor que se transmitirá cuando el usuario inicie una secuencia de reconocimiento al final de una recepción.

bit 4 **ACKEN**: Reconocimiento del bit de habilitación de secuencia (solo modo de recepción maestra)

1 = Iniciar la secuencia de reconocimiento en los pines SDA y SCL y transmitir el bit de datos ACKDT. Borrado automáticamente por hardware.

0 = Confirmar secuencia inactiva

bit 3 **RCEN**: bit de habilitación de recepción (solo en modo maestro)

1 = Habilita el modo de recepción para I²C

0 = Recibir inactivo

bit 2 **PEN**: bit de habilitación de condición de parada (solo modo maestro)
1 = Iniciar condición de parada en los pines SDA y SCL. Borrado automáticamente por hardware.
0 = Condición de parada inactiva

bit 1 **RSEN**: bit habilitado de condición de inicio repetido (solo en modo maestro)
1 = Iniciar la condición de inicio repetido en los pines SDA y SCL. Borrado automáticamente por hardware.
0 = Condición de inicio repetido inactivo

bit 0 **SEN**: Condición de inicio habilitada / estirado habilitado bit
En modo maestro:
1 = Iniciar condición de inicio en los pines SDA y SCL. Borrado automáticamente por hardware.
0 = Condición de inicio inactivo
En modo esclavo:
1 = La extensión del reloj está habilitada tanto para la transmisión esclava como para la recepción esclava (extensión habilitada)
0 = El alargamiento del reloj está habilitado para transmisión esclava solamente (compatibilidad con PIC16F87X)

¿Como trabaja la comunicación I2C?

Para poder reconocer cada uno de los dispositivos conectados a los DOS hilos del bus I2C, a cada dispositivo se le asigna una dirección.

Así en este tipo de comunicaciones el MAESTRO es el que tiene la iniciativa en la transferencia y este es quien decide con quien se quiere conectar para enviar y recibir datos y también decide cuando finalizar la comunicación.

Los DOS hilos del BUS interfaz de comunicación I2C PIC son líneas de colector abierto donde una de las líneas lleva la señal de reloj y es conocida como (SCL), y la otra línea lleva los datos y es conocida como (SDA).

Para que la comunicación funcione se deben utilizar unas resistencias PULL UP (resistencias conectadas a positivo) para asegurar un nivel alto cuando NO hay dispositivos conectados al BUS I2C.

10.2.1 Configurar el módulo I^2C

Configurar el módulo I^2C del microcontrolador para trabajar en modo maestro.

para configurar el módulo será necesario ajustar correctamente los registros de configuración SSPSTAT, SSPCON, SSPCON2. Para ello se utilizará la función i2c_init(). además, se definirá diferentes funciones adicionales que permitirán interactuar de forma sencilla con el módulo tales como.

- i2c_start(), i2c_restart(), i2c_stpo(), i2c_idle(), i2c_read(), i2c_write() y i2c_sendnack().

Cuya definición y código sí introducirán en los archivos “i2c.h” e “i2c.c” respectivamente.

Código fuente:

```
// Configura el módulo I2C

void i2c_init(){
        TRISC3 = 1;             // SLC
        TRISC4 = 1;             // SDA
        SSPADD = 0x27;          // Reloj a 100KHz FOSC = 16MHZ SSPADD = (FOSC/ (4* Reloj))-1)
        SSPSTAT = 0x80;         // Habilita el control de velocidad del módulo
        SSPCON2 = 0x00;         // Baja los flags
        SSPCON = 0x28;          // Modo maestro, módulo I2C encendido
}
// Envía la secuencia de Start

void i2c_start(){
        i2c_idle();                     // Comprueba que está libre
        SSPCON2bits.SEN=1;              // Envía la condición de inicio
        while(SSPCON2bits.SEN==1);      // Espera a que se envíe
        while(SSPIF==0);                // Comprueba que se ha enviado
        SSPIF=0;                        // Baja el flag
}

// Envía la secuencia de (re)start

void i2c_restart(){
        SSPCON2bits.RSEN=1;             // Envía la secuencia restart
        while(SSPCON2bits.RSEN);        // Espera a que se envíe
        while(SSPIF==0);                // Comprueba que se ha enviado
        SSPIF=0;                        // Baja el flag
}
//      Envía la secuencia de Stop

void i2c_stop(){
        SSPCON2bits.PEN=1;              // Envía la secuencia de Stop
        while(SSPCON2bits.PEN);         // Espera a que se envíe
        while(SSPIF==0);                // Comprueba que se ha enviado
        SSPIF=0;                        // Baja el flag
}

// Comprueba que el modulo I2C está libre

void i2c_idle(){
while( ( SSPCON2 & 0x1F ) | SSPSTATbits.R_nW ); // Comprueba que el módulo está libre
}

// Lee un byte

unsigned char i2c_read(){
        i2c_idle();
        SSPCON2bits.RCEN=1;             // Activa el modo recepción
        while(SSPCON2bits.RCEN==1);     // Espera a recibir el byte
        while(SSPSTATbits.BF==0);       // Espera a que el buffer esté lleno
        return SSPBUF;                  // Devuelve el dato
}

// Envía un byte al esclavo y devuelve 0 en caso de error

unsigned char i2c_write(unsigned char byte){
        i2c_idle();
        SSPBUF=byte;
        while(SSPSTAT&0x05);            // Comprueba si hay transmisión en curso
        if(SSPCON2bits.ACKSTAT==1){     // No ha recibido confirmación
```

```
                SSPCON2bits.PEN=1;          // Envía secuencia de Stop
                return  0 ;                         // devuelve 0
        }
        else{                                       // Ha recibido confirmación
                while(SSPIF==0);            // Comprueba que ha transmitido
                SSPIF=0;                                    // Baja el flag
                return  1;                                  // Devuelve 1
        }
}

// Envía la secuencia Nack

void i2c_sendnack(){
        SSPCON2bits.ACKDT=1;
        SSPCON2bits.ACKEN=1;
        while(SSPCON2bits.ACKEN==1);
}
```

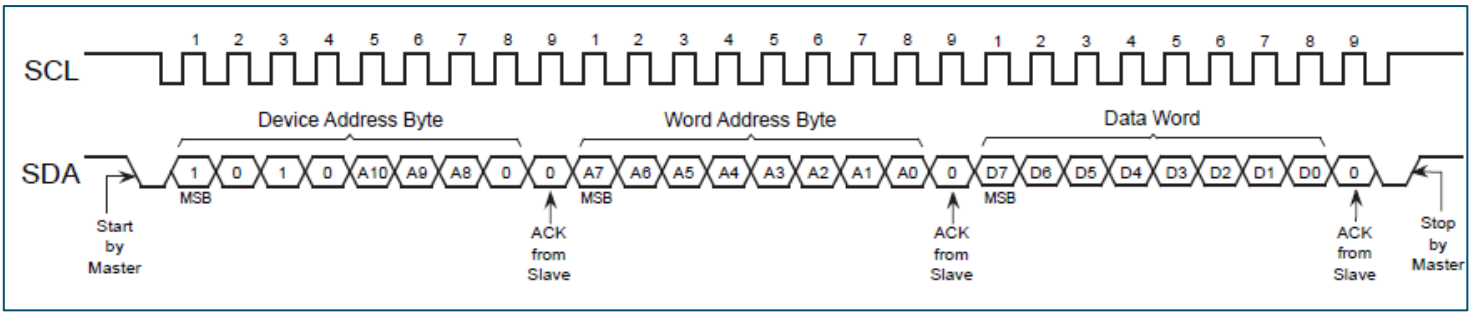
10.2.2 Lectura de memoria

Realizar un programa basado en un microcontrolador que almacene en la posición de memoria 0x0000 una memoria pro externa, el número de veces que se ha arrancado el programa almacenando en el pic. La memoria para usar es la siguiente AT24C16D.

Memoria nuestra memoria externa cuenta con las resistencias de pull-up conectados a la línea de comunicación I^2C que mantendrán estas líneas en alto cuando estén sin utilizar. normalmente las memorias externas cuentan con terminales para determinar la dirección estas son A0-A2, que en total son 3 terminales, pero nuestra memoria no cuenta con estas direcciones por lo que se consideran como un valor bajo '0V', estas direcciones ayudan a identificar cada uno de los dispositivos que se conectan en el hilo del I^2C. La terminal WP (write-protect) se conectará también a tierra '0V'.
Internamente en diferentes memorias cuenta con una dirección interna divididas en dos grupos, una dirección alta y una dirección baja, pero en esta memoria cuenta con una sola dirección por lo que no es necesario configurar las dos direcciones. Si desea enviar las dos direcciones solo se debe de descomentar las líneas que viene en la función EEPROM_i2c_readbyte.

Para la comunicación de la memoria EEPROM es necesario configurar el módulo de comunicaciones I^2C de microcontrolador en modo maestro cómo se mostró en el ejemplo anterior, una vez configurado el módulo se deberá de establecer la comunicación con el dispositivo siguiendo las indicaciones de la hoja de especificaciones para las operaciones de lectura y escritura que se detallan en la figura siguiente.

Escribir Byte



Leer Byte

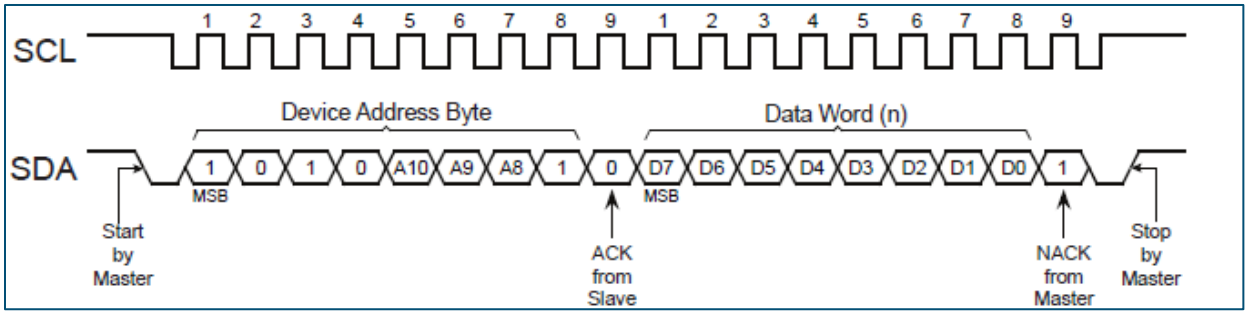


Imagen 10-2 Esquema para escritura y lectura de datos de la memoria.

Configuración

Para realizar la comunicación con la memoria de EEPROM se crearán dos funciones EEPROM_i2c_readbyte() y EEPROM_i2c_writebyte(), que se encargarán de leer y escribir en la memoria EEPROM. estas funciones harán uso de la librería “i2c.h” donde se encuentran definidas todas las funciones de comunicación para trabajar con el módulo. Crear los archivos de cabecera necesarios.

```
#include <xc.h>
#include "i2c.h"
#include "i2c_eeprom.h"

// Gets a byte from the I2C slave ADDRESS=dat and REG=reg

unsigned char      EEPROM_i2c_readbyte(unsigned char add, unsigned int reg){
            unsigned char temp=0;
            i2c_start();
            if(i2c_write(add+WRITE)==0){
                        return 0;
            }
//          if(i2c_write((unsigned char)(reg>>8))==0){      //Descomentar para usar doble dirección
//          return 0;
//          }
            if(i2c_write((unsigned char)(reg))==0){
            return 0;
            }
            i2c_restart();
            if(i2c_write(add+READ)==0){
            return 0;
            }
            temp=i2c_read();
            i2c_sendnack();
            i2c_stop();
            return temp;
}

// Writes a byte=dato from the I2C slave ADDRESS=dat at REG=reg

unsigned char EEPROM_i2c_writebyte(unsigned char add, unsigned int reg, unsigned char dato){
            i2c_start();
            if(i2c_write(add+WRITE)==0){
                        return 0;
            }
//          if(i2c_write((unsigned char)(reg>>8))==0){   //Descomentar para usar doble dirección
//          return 0;
//          }
            if(i2c_write((unsigned char)(reg))==0){
            return 0;
            }
            if(i2c_write(dato)==0){
            return 0;
```

```
        }
        i2c_stop();
        return 1;
}
```

Código fuente:

```
#include <xc.h>
#include "configuracion_bits.h"
#include "usart.h"                          // Librería para el módulo de comunicaciones USART
#include "i2c.h"
#include "i2c_eeprom.h"
#include <stdio.h>                                  // Librería con funciones para el control de entradas y salidas

#define EEPROM_ADDRESS 0xA0 // Dirección del módulo EEPROM (ver especificaciones)
#define EEPROM_REG 0x00                     // Dirección del registro de memoria (ver especificaciones)

char ch;
void main(void){

    /*Configuración de los puertos */
    ADCON1 = 0x07;          // Deshabilitar PORTA Y PORTE de ADC
    TRISA = 0X03;          //Se configura 0-1 como 1, 2-5 como 0
    TRISB = 0X00;
//    TRISC = 0x10;          //Solo como entrada RC4, los demás salidas
    TRISD = 0x0E;          //Entradas RD1-RD3, las demás salidas
    TRISE = 0X00;          //Todo como salidas

    PORTA = 0xFF;
    PORTE = 0xFF;
    PORTC = 0xFF;
    PORTD = 0xFF;
    __delay_ms(500);
    PORTA = 0x00;
    PORTE = 0x00;
    PORTC = 0x00;
    PORTD = 0x00;
    PORTDbits.RD4 = 1;      // Apagar LED Rojo
    PORTDbits.RD5 = 1;      // Apagar LED Verde
    PORTDbits.RD6 = 1;      // Apagar LED Azul
    PORTCbits.RC5 = 0;      // Apagar ánodo
    __delay_ms(500);       // Esperamos 500ms

        INIT_USART(9600,HIGH_SPEED,EIGHT_BITS);       // Configura el módulo USART
    printf("\n\n\r Leer puerto serial");
    unsigned char dato,i,lectura;       // Definición de variables de programa
        di();                                   // Se deshabilitan las interrupciones
        i2c_init();
    printf("\n\r Leer memoria EEPROM ");
        dato = EEPROM_i2c_readbyte(EEPROM_ADDRESS, EEPROM_REG);// Lectura del valor almacenado
    printf("\n\r dato = %d",dato);
        dato ++; // Se incrementa el valor en 1
    printf("\n\r Escribir memoria EEPROM");
        i = EEPROM_i2c_writebyte(EEPROM_ADDRESS, EEPROM_REG, dato);          // Se escribe el Nuevo valor
    printf("\n\r dato = %d",dato);
    dato = EEPROM_i2c_readbyte(EEPROM_ADDRESS, EEPROM_REG);          // Lectura del valor almacenado
    printf("\n\r dato = %d",dato);

        while(1);
}
```

11 Anexos

Código fuente:

```
#include <xc.h>
#
```

12 Control de versiones

Versión	Fecha	Responsable	Cambios
V0.0	3/12/2020	Francisco A.	Inicio del Documento
V0.1	19/01/2020	Francisco A.	Cambios de imagen. División de temas
V0.2			

Imagen 0-1 Esquemático