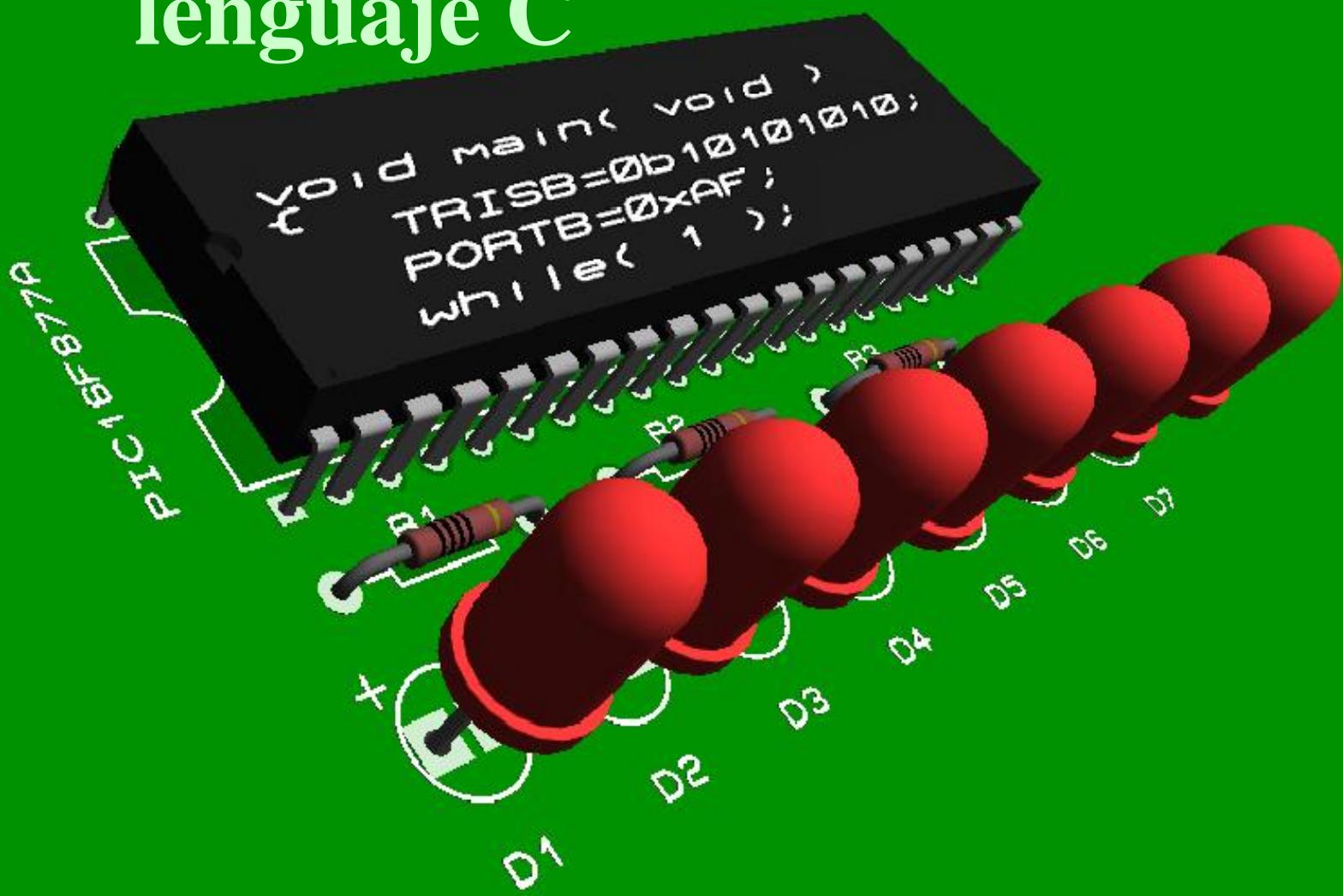


Diseño y simulación de sistemas microcontrolados en lenguaje C



Programación con MikroC PRO
Simulación en Proteus ISIS



Juan Ricardo Clavijo Mendoza

Dedicado a mí esposa:
Sandra, mi hijo Miguel, mis padres Jorge y Bibiana

Diseño y simulación de sistemas microcontrolados en lenguaje C

**Programación con el compilador MikroC PRO y
simulación en Proteus ISIS**

Juan Ricardo Clavijo Mendoza

Diseño y simulación de sistemas microcontrolados en lenguaje C.

Se prohíbe la reproducción total o parcial de este libro
sea cual fuere el medio, electrónico o mecánico,
sin el consentimiento escrito del autor.

Autor

Ing. Juan Ricardo Clavijo Mendoza

Corrección de estilo

Ing. Sandra Milena Bernate Bautista

Dr. Duilio Cruz Becerra

Hecho e impreso en Colombia

ISBN 978-958-44-8619-6

Primera edición mayo de 2011

Contenido

| | | |
|-------|---|-----|
| 1 | El Compilador MikroC y los PICMicro | 17 |
| 1.1 | El compilador MikroC PRO | 18 |
| 2 | Fundamentos de lenguaje C | 20 |
| 2.1 | Declaración de variables en lenguaje C | 20 |
| 2.2 | Formatos numéricos usados en el lenguaje C | 23 |
| 2.3 | Operadores en lenguaje C | 23 |
| 2.4 | Funciones en lenguaje C | 27 |
| 2.5 | Creación de un programa en lenguaje C | 29 |
| 2.6 | Condicionales e iteraciones en lenguaje C | 30 |
| 2.6.1 | Creación de condiciones lógicas en lenguaje C | 30 |
| 2.6.2 | La sentencia condicional if e if else | 31 |
| 2.6.3 | La sentencia switch case | 32 |
| 2.6.4 | El ciclo iterativo while y do while | 32 |
| 2.6.5 | El ciclo iterativo for | 33 |
| 2.6.6 | Uso anidado de ciclos iterativos | 33 |
| 3 | El simulador ISIS de Proteus | 34 |
| 3.1 | Características básicas del ISIS para simular | 35 |
| 4 | Creación del primer programa en MikroC PRO | 38 |
| 5 | Visualización de datos | 44 |
| 5.1 | Display de 7 segmentos | 44 |
| 5.1.1 | Control de display de 7 segmentos | 45 |
| 5.1.2 | Control de displays 7 segmentos dinámicos | 47 |
| 5.2 | Display LCD de caracteres | 50 |
| 5.2.1 | Funciones para imprimir caracteres | 54 |
| 5.2.2 | Funciones para imprimir cadenas de texto | 55 |
| 5.2.3 | Impresión de valores numéricos | 57 |
| 5.2.4 | Creación de caracteres propios | 59 |
| 5.3 | Display LCD gráficos | 64 |
| 6 | Teclados y sistemas de entrada de datos | 74 |
| 6.1 | Uso de pulsadores | 74 |
| 6.2 | Uso de Dip-Switch | 76 |
| 6.3 | Uso de teclados matriciales | 78 |
| 6.4 | Uso de teclados PS2 o Din | 84 |
| 7 | Comunicaciones seriales | 88 |
| 7.1 | Modulo serial I ² C | 88 |
| 7.2 | Módulo USART | 91 |
| 7.3 | Módulo USB | 95 |
| 8 | Conversión AD y DA | 109 |
| 8.1 | Conversión AD, o ADC | 109 |
| 8.2 | Conversión DA o DAC | 110 |
| 8.2.1 | Conversión DA con PWM | 110 |
| 8.2.2 | Conversión DA con arreglo R-2R | 114 |
| 9 | Memorias EEPROM y FLASH | 117 |
| 9.1 | Memoria EEPROM | 117 |
| 9.2 | Memoria FLASH | 118 |
| 10 | Módulos Timer | 121 |

| | | |
|---------|---|-----|
| 11 | Interrupciones..... | 124 |
| 12 | Sensores..... | 128 |
| 12.1 | Sensor de temperatura LM35 | 128 |
| 12.2 | Sensores de presión..... | 131 |
| 12.3 | Sensores de distancia | 134 |
| 12.4 | Sensores LDR | 138 |
| 12.5 | Sensores de humedad y temperatura..... | 141 |
| 13 | Comunicación con dispositivos..... | 147 |
| 13.1 | Módulos GPS | 147 |
| 13.2 | Módulos inalámbricos unidireccionales..... | 161 |
| 13.3 | Módulos inalámbricos bidireccionales..... | 167 |
| 13.4 | Comunicación RS 485 | 172 |
| 13.5 | Módulos inalámbricos infrarrojos..... | 174 |
| 13.6 | Comunicación con memorias SD..... | 177 |
| 13.7 | Reloj en tiempo real..... | 183 |
| 14 | Tratamiento digital de señales..... | 190 |
| 14.1 | Muestreo | 190 |
| 14.2 | Funciones de transferencia..... | 191 |
| 14.3 | Convolución | 192 |
| 14.4 | Filtros FIR | 192 |
| 14.4.1 | Filtro Pasa bajas | 193 |
| 14.4.2 | Filtros Pasa Altas..... | 194 |
| 14.4.3 | Filtro Pasa Banda | 194 |
| 14.4.4 | Filtro Rechaza Banda | 195 |
| 14.4.5 | Filtro Multi Band..... | 196 |
| 14.4.6 | Ventanas fijas | 196 |
| 14.4.7 | Ventanas Rectangulares | 197 |
| 14.4.8 | Ventanas Hamming | 198 |
| 14.4.9 | Ventanas Hanning | 199 |
| 14.4.10 | Ventanas Blackman | 201 |
| 14.5 | Ejemplos de diseño para filtros FIR..... | 202 |
| 14.5.1 | Ejemplo de diseño para filtro pasa bajas | 204 |
| 14.5.2 | Ejemplo de diseño para filtro pasa altas | 207 |
| 14.5.3 | Ejemplo de diseño para filtro pasa banda..... | 208 |
| 14.5.4 | Ejemplo de diseño para filtro rechaza banda | 209 |
| 14.5.5 | Ejemplo de diseño para filtro multi banda | 210 |
| 14.6 | Filtros IIR..... | 212 |
| 14.6.1 | Transformación bilineal | 213 |
| 14.6.2 | Filtro Pasa Bajas IIR | 213 |
| 14.6.3 | Ejemplo de diseño para filtro pasa bajas | 215 |
| 14.6.4 | Filtro Pasa Altas IIR | 216 |
| 14.6.5 | Ejemplo de diseño para filtro pasa altas | 217 |
| 14.6.6 | Filtro Pasa Banda IIR | 219 |
| 14.6.7 | Ejemplo de diseño para filtro pasa banda..... | 220 |
| 14.6.8 | Filtro Rechaza Banda IIR | 221 |
| 14.6.9 | Ejemplo de diseño filtro rechaza banda | 222 |
| 14.7 | Osciladores digitales | 224 |
| 14.7.1 | Ejemplo de diseño oscilador doble cuadrado | 225 |
| 14.7.2 | Ejemplo de diseño para oscilador de acople en cuadratura..... | 226 |

| | | |
|-------------------|---|-----|
| 14.8 | Transformada discreta de Fourier DFT..... | 228 |
| 14.8.1 | Transformada rápida de Fourier FFT | 232 |
| 14.9 | Control digital PID..... | 238 |
| 15 | Transmisión de datos..... | 243 |
| 15.1 | Control de flujo | 244 |
| 15.2 | Transparencia de datos..... | 247 |
| 15.3 | Control de errores CRC | 252 |
| 16 | Actuadores y potencia | 262 |
| 16.1 | Actuadores DC..... | 262 |
| 16.1.1 | Relevadores | 263 |
| 16.1.2 | Motores DC | 264 |
| 16.1.3 | Puente H | 265 |
| 16.1.4 | Motores Paso | 267 |
| 16.1.5 | Servomotores..... | 271 |
| 16.2 | Actuadores AC..... | 273 |
| 17 | Anexos..... | 279 |
| 17.1 | Tabla ASCII | 279 |
| Bibliografía..... | | 281 |
| Índice | | 283 |

El autor

Juan Ricardo Clavijo Mendoza, Ingeniero Electrónico egresado de la universidad Manuela Beltrán de Bogotá D.C, Colombia. Autor de capítulo de libro de: “Ciencia, tecnología e innovación”, Tomo 1, publicado por la Universidad Católica de Colombia. Docente de la Universidad Católica de Colombia, en el programa de Tecnología en Electrónica y Telecomunicaciones, asesor temático en proyectos de investigación de la Universidad Católica de Colombia, y la Escuela de Telemática de la Policía Nacional de Colombia, desarrollador de proyectos tecnológicos en diversas empresas de Bogotá Colombia.

Prologo

El desarrollo de la tecnología hace parte vital de la evolución y el progreso de una nación, está razón motiva la creación de documentos que formen a los desarrolladores e investigadores en las diversas áreas de la ciencia. Este libro busca hacer un aporte en este propósito como un curso de fácil entendimiento en el diseño de sistemas microcontrolados, para este fin este texto se enfoca en la simulación de sistemas digitales en el paquete de software PROTEUS, y la programación de microcontroladores PIC con la herramienta de programación en lenguaje C MikroC PRO. Este libro es ideal para estudiantes de ingeniería, tecnología, o carreras técnicas en electrónica, sistemas, mecatrónica, biomédica, y todas aquellas afines con la electrónica. De igual manera es un aporte importante al desarrollo de proyectos de investigación y tecnología.

Introducción

Este libro trata temáticas relacionadas con el diseño de sistemas con microcontroladores, de la familia microchip 12F, 16F, y 18F. Usando como herramienta de diseño, el compilador MikroC PRO, y el paquete de software para simulación Proteus. Para entender este libro de forma clara el estudiante o desarrollador debe tener bases sólidas de electrónica digital como circuitos combinacionales y secuenciales. También es de utilidad tener conocimientos básicos de programación en lenguaje C y algoritmos. De la misma forma, se debe conocer las teorías de circuitos eléctricos, y circuitos electrónicos análogos, teorías básicas de álgebra, y álgebra lineal así como la manipulación de números complejos en todas sus expresiones. Este libro se enfoca en la demostración de diseños simples que posteriormente pueden ser integrados para realizar diseños de mayor complejidad en función de la necesidad del desarrollador.

1 El Compilador MikroC y los PICMicro

Un microcontrolador es un dispositivo electrónico encapsulado en un circuito de alto nivel de integración. Los microcontroladores se pueden adquirir comercialmente de diferentes casas fabricantes como: Freescale, Motorola, Intel, Philips, y Microchip.

Microchip en particular es una empresa fabricante de dispositivos electrónicos, en sus líneas de producción se encuentran los microcontroladores PICMicro, los cuales se pueden adquirir en diferentes familias, algunas de ellas son: 12F, 16F, 18F, 24F, 30F, y 33F.

En función de la necesidad del proyecto el desarrollador debe escoger la familia y la referencia que más se acerque a su necesidad, por ejemplo el microcontrolador 12F675 es un PIC de 8 pines con módulos integrados básicos como: Timer y ADC. Un microcontrolador como el 16F877 cuenta con 40 pines y módulos como: Timer, ADC, USART, I2C, PWM, entre otros. Fácilmente se pueden apreciar diferencias que permiten crear aplicaciones diferentes entre estos dos ejemplos.



Figura 1-1

La información técnica, la masiva comercialización y la increíble información publicada acerca de los microcontroladores PIC, los hace ideales para aprender y estudiar su funcionamiento, la empresa Microchip cuenta con el portal WEB www.microchip.com en donde se puede descargar información y aplicativos de software que facilitan los desarrollos con sus microcontroladores.

Básicamente implementar un desarrollo con un microcontrolador PIC consiste en identificar la problemática del desarrollo, crear editar y depurar un programa de máquina y programar eléctricamente el microcontrolador con un programador específico para los PICMicro. Microchip suministra programadores especializados en diferentes escalas, tal vez el más popular es el PICSTART Plus, sin embargo existen otros como el PICkit2, PICkit3. A pesar de que existan programadores comerciales un desarrollador puede construir o adquirir un programador didáctico de bajo costo, de estos últimos existe amplia información publicada en Internet.



Figura 1-2

Un microcontrolador tiene una arquitectura básica que es similar a la de un computador de escritorio, cuenta con un bloque de memoria OTP o Flash en la cual se guardan las instrucciones del programa esta sección es similar al disco duro del computador, el PICMicro cuenta con una memoria RAM, que cumple las mismas funciones de la memoria RAM de un ordenador personal, el microcontrolador posee puertos de entrada y salida que son similares a los periféricos de entrada y salida del computador tales como puertos para el ratón, impresora, monitor, teclado y demás. Estas características hacen que un microcontrolador sea ideal para crear aplicaciones a pequeña escala que tengan interfaz de usuario, adecuando teclados, botones, lectura de memorias de almacenamiento masivo, sensores de diversas variables como: temperatura, humedad, presión, luminosidad, proximidad, entre otros. De igual manera, es posible crear ambientes de visualización con displays numéricos, alfanuméricos y gráficos. Los puertos seriales como la USART y USB permiten crear comunicaciones seriales y comunicaciones inalámbricas con otros dispositivos. En síntesis las posibilidades son infinitas.

1.1 El compilador MikroC PRO

La programación de microcontroladores se basa en un código de máquina que es conocido como código ensamblador, este código contiene una a una las instrucciones del programa, este código ensamblador o también conocido como código asembler es minucioso, y tedioso de editar. El asembler crea códigos de programa extensos y de difícil comprensión. La creación de compiladores de alto nivel facilitó la edición y creación de programas en todo modo de programación lógica, por supuesto los microcontroladores no fueron la excepción, comercialmente existen varios compiladores de diferentes fabricantes y diferentes lenguajes de alto nivel.

Es posible adquirir compiladores como el PICC, CCS, PIC Basic, entre otros. El estudio de este libro se centra en el compilador MikroC PRO, que es un compilador en lenguaje C para microcontroladores PICMicro de la familia 12F, 16F, y 18F.

MikroC PRO es un paquete de software con una amplia variedad de ayudas y herramientas que facilita la creación de proyectos y aplicativos para los microcontroladores PICMicro.

El estudio de este entorno de desarrollo es posible debido a que el estudiante puede descargar una versión demo o estudiantil, que tiene las mismas características de la versión completa, la única limitación es la dimensión del código de máquina que no puede exceder 2K bytes, sin embargo es una capacidad suficiente al tratarse de un primer aprendizaje. La versión demo se puede descargar de la pagina: www.mikroe.com.

En la siguiente figura se puede apreciar la apariencia visual del entorno de desarrollo.

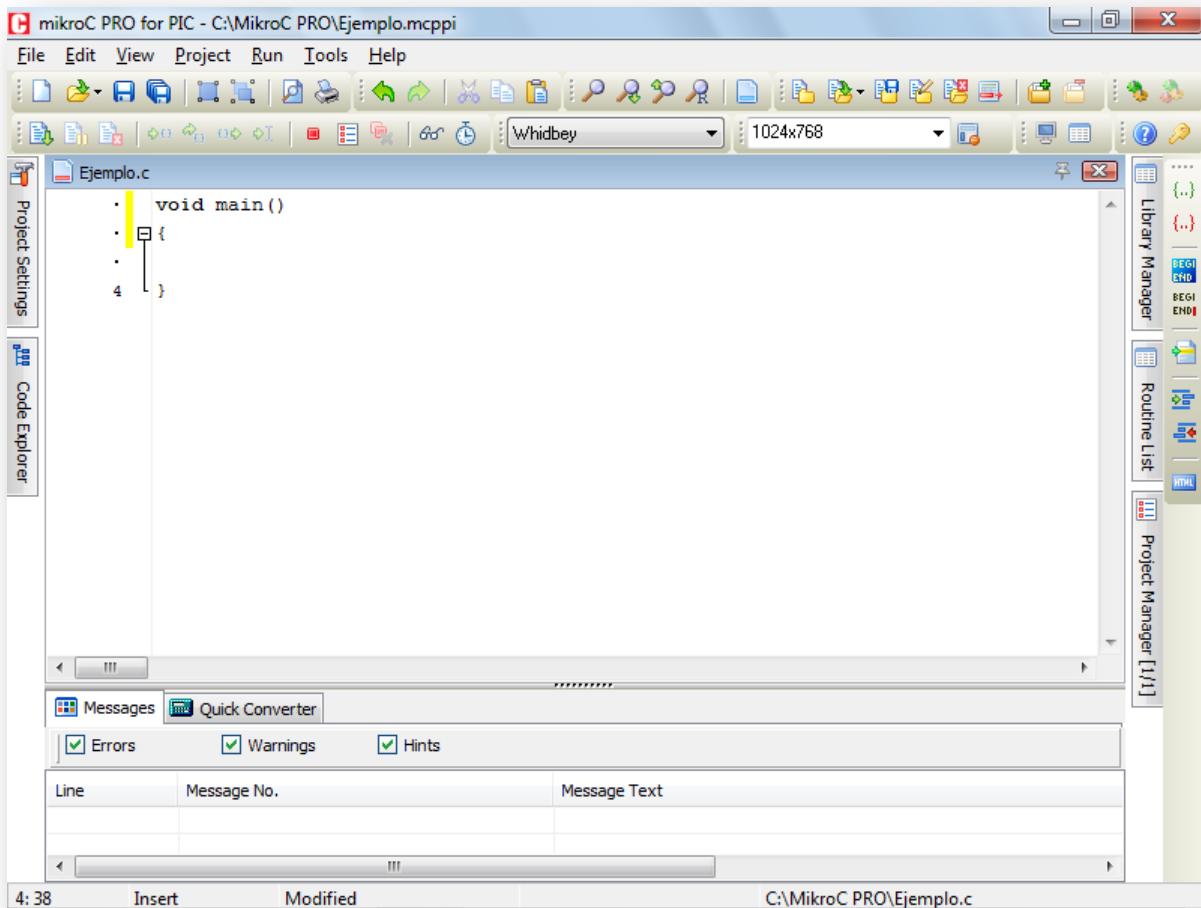


Figura 1-3

El compilador de alto nivel en lenguaje C utiliza estructuras que facilitan la programación, optimiza las operaciones matemáticas y los procesos, por medio del uso de funciones predefinidas y las no predefinidas que el desarrollador puede crear, así como el uso de un conjunto de variables, de tipo carácter, entero, y punto decimal. El compilador crea automáticamente el código ensamblador y a su vez un código similar consignado en un archivo con extensión *.hex, este archivo es el resultado primordial del compilador dado que con este se programa eléctricamente el microcontrolador o con el mismo se puede realizar una simulación computacional.

2 Fundamentos de lenguaje C

El lenguaje C data del año 1972; fue creado por los laboratorios Bell como resultado de la necesidad de reescribir los sistemas operativos UNIX con el fin de optimizar el conocido código ensamblador. De igual manera el lenguaje C fue la evolución de lenguajes previos llamados B, y BCPL. El nuevo lenguaje C, rápidamente tomó fuerza por su funcionalidad y facilidad en la implementación en diversos sistemas computacionales que requerían códigos de máquina.

La forma del lenguaje C, se fundamenta en un complejo estructural que requiere un amplio estudio de las mismas, sin embargo para la programación de los microcontroladores el estudiante requiere una porción fundamental que le permita iniciar y crear los primeros proyectos en MikroC PRO, para este fin el actual capítulo se centra en conocer las nociones necesarias para iniciar el estudio de los microcontroladores con este libro.

2.1 Declaración de variables en lenguaje C

Las variables básicas en este compilador específico son:

- *bit*
- *char*
- *short*
- *int*
- *long*
- *float*
- *double*

Las variables *bit* permiten almacenar un valor lógico es decir verdadero o falso, 0 ó 1.

Las variables *char* se utilizan para almacenar caracteres codificados con el código ASCII, son útiles para guardar letras o textos.

Una variable *short* almacena un número entero de 8 bits corto puede valer de: -127 a 127.

Las variables tipo *int* guardan números enteros de 16 bits, esta variable permite guardar números de: -32767 a 32767.

La variable tipo *long* almacena números enteros largos de 32 bits, su rango puede ser de: -2147483647 a 2147483647.

Las variables tipo *float* y *double* permiten guardar números con punto decimal.

Las anteriores variables pueden declararse incluyendo el signo positivo y negativo, o se pueden declarar por medio de la opción sin signo con la directriz *unsigned*.

En la siguiente tabla se pueden apreciar las características de las variables.

| Tipo de variable | Tamaño en Bytes | Valores que soporta |
|-------------------------|------------------------|----------------------------|
| <i>bit</i> | 1 | 0 ó 1 |
| <i>char</i> | 1 | -127 a 127 |
| <i>short</i> | 1 | -127 a 127 |
| <i>int</i> | 2 | -32767 a 32767 |
| <i>long</i> | 4 | -2147483647 a 2147483647 |
| <i>float</i> | 4 | -1.5x10^45 a 3.4x10^38 |
| <i>double</i> | 4 | -1.5x10^45 a 3.4x10^38 |
| <i>unsigned char</i> | 1 | 0 a 255 |
| <i>unsigned short</i> | 1 | 0 a 255 |
| <i>unsigned int</i> | 2 | 0 a 65535 |
| <i>unsigned long</i> | 4 | 0 a 4294967295 |

Tabla 2-1

La declaración de variables se realiza indicando el tipo de variable seguido de un nombre que el desarrollador asigna arbitrariamente a la variable. En el punto de la declaración de una variable es posible dar un valor inicial a cada una de las variables sin embargo este último no es estrictamente necesario. Por último la declaración debe culminar con el carácter punto y coma (;).

En los siguientes ejemplos se puede apreciar como se hacen las declaraciones:

```

bit VARIABLE1_BIT;           //Declaración de una variable tipo bit.
char CARACTER;             //Declaración de una variable tipo char.
char CARACTER2='J';          //Declaración de una variable tipo char inicializada con el
//valor ASCII del carácter J.
int ENTERO=1234;            //Declaración de una variable tipo entera inicializada con
//el valor 1234.
float DECIMAL=-12.45;        //Declaración de una variable con punto decimal
//inicializada con el valor -12,45.
double DECIMAL2=56.68;       //Declaración de una variable con punto decimal
//inicializada con el valor 56,68.
long ENTERO2=-954261;        //Declaración de una variable de tipo entero largo
//inicializada con el valor -954261.

```

Los siguientes ejemplos muestran como declarar variables sin signo:

```

unsigned char CARACTER; //Declaración de una variable tipo char sin signo.
unsigned int ENTERO;      //Declaración de una variable tipo entera sin signo.
unsigned long ENTERO2;    //Declaración de una variable de tipo entero largo sin signo.

```

Las variables también pueden ser declaradas en un formato que asocia varias variables a un mismo nombre, este formato se conoce como una cadena de variables, o un vector e incluso puede ser una matriz de variables, en conclusión este tipo de declaraciones pueden ser de una o más dimensiones.

El siguiente ejemplo muestra un vector de caracteres, o también conocido como una cadena de caracteres:

```
char Texto[20]; //Cadena de caracteres con 20 posiciones de memoria.
```

De igual manera las cadenas de caracteres o de variables pueden ser declaradas con un valor inicial, este tipo de declaración se puede ver en el siguiente ejemplo:

```
char Texto[20] = "Nuevo Texto"; //Declaración de una cadena de caracteres  
//inicializada con el texto: Nuevo Texto.  
int Enteros[5]={5,4,3,2,1}; //Declaración de una cadena de enteros con  
//valores iniciales.  
float Decimales[3]={0.8,1.5,5.8}; //Declaración de una cadena de números con  
//punto decimal inicializadas.
```

La declaración de las variables debe respetar algunas reglas básicas que evitan errores y contradicciones en la compilación del código, para este fin tenga presente las siguientes recomendaciones:

- Las variables no deben tener nombres repetidos.
- Las variables no deben empezar por un número.
- Una variable no puede utilizar caracteres especiales como: / * ` ; { } - \ ! . % &.

A continuación se muestran ejemplos de declaraciones de variables que no se pueden hacer:

```
bit 1_VARIABLE-;  
char -CARÁCTER!;  
int 3ENTERO*;
```

De igual manera es posible crear estructuras de información como un nuevo tipo de variable creada por el desarrollador. Estás estructuras se pueden realizar con las variables ya predefinidas y pueden ser declaraciones de variables o de arreglos de variables. Este tipo de estructuras se declarar, por medio de la directriz: *typedef struct*, y la forma de declarar una variable creada por el desarrollador es la siguiente:

```
typedef struct  
{  
    char Nombre[10];  
    int Edad;  
}Usuario;
```

La siguiente es la forma de usar una variable personalizada por el desarrollador:

```
Usuario U;  
U.Edad = 25;  
U.Nombre[0] = 'J';  
U.Nombre[1] = 'u';  
U.Nombre[2] = 'a';
```

```
U.Nombre[3]='n';
U.Nombre[4]=0;
```

2.2 Formatos numéricos usados en el lenguaje C

Los aplicativos en lenguaje C usan números en diferentes bases numéricas, a pesar de que para el trabajo de bajo nivel del microcontrolador todos sus números están procesados en base 2 es decir en números binarios. El ser humano no está acostumbrado a pensar y procesar operaciones en esta base numérica. Desde las primeras etapas de la formación académica las escuelas y colegios enseñan a pensar y procesar todos los cálculos en base 10, es decir con números decimales. Es por esto que los compiladores en lenguaje C trabajan los números decimales facilitando los diseños para el desarrollador. Además del sistema decimal el compilador en lenguaje C puede trabajar otras bases tales como el binario, y hexadecimal haciendo más simple realizar cálculos y tareas que en decimal serían más complejas. Los sistemas numéricos en base 2, 10, y 16, son los implementados en este compilador en lenguaje C. La escritura de números decimales, es la forma de mayor simplicidad ya que se escriben en lenguaje C de la misma manera convencional que se aprende desde los primeros cursos de matemáticas. Los números binarios se escriben con el encabezado 0b seguidos del número en binario, un ejemplo de esta escritura es: 0b10100001 que es equivalente al número decimal 161. Los números en base 16 o hexadecimales se denotan con el encabezado 0x precedidos de un número en hexadecimal, la expresión 0x2A, es un ejemplo de un número hexadecimal en lenguaje C, y equivale a 42 en decimal. Los números binarios solo pueden tener dos dígitos que son el 0 y el 1. Los números hexadecimales pueden tener 16 dígitos que son; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.

2.3 Operadores en lenguaje C

El lenguaje C permite hacer operaciones aritméticas o matemáticas básicas entre números contenidos en variables o constantes. Las operaciones aritméticas disponibles son las siguientes:

- Suma
- Resta
- Multiplicación
- División
- Modulo

La suma aritmética entre dos o más números, se puede hacer como se ve en el siguiente ejemplo:

```
int A;
```

```
int B;
```

```
int C;
```

C = A+B; //Está expresión guarda la suma de A y B en la variable C.

C = A+B+C; //Está expresión guarda la suma de A, B y C en la variable C.

La resta aritmética entre dos o más números, se puede hacer como se ve en el siguiente ejemplo:

```
int A;
```

```
int B;
```

```
int C;
```

C = A-B; //Está expresión guarda la diferencia entre A y B en la variable C.

C = A-B-C; //Está expresión guarda la diferencia entre A, B y C en la variable C.

La operación matemática de multiplicación se puede realizar entre dos o más números, la operación se relaciona con el carácter asterisco (*), esta expresión se puede apreciar con mayor claridad en los siguientes ejemplos:

int A;

int B;

int C;

*C = A*B; //Está expresión guarda la multiplicación entre A y B en la variable C.*

*C = A*B*C; //Está expresión guarda la multiplicación entre A, B y C en la variable C.*

La división aritmética en lenguaje C se especifica por medio de la barra inclinada (/), en el siguiente ejemplo se puede observar su implementación:

int A;

int B;

int C;

C = A/B; //Está expresión guarda la división A entre B en la variable C.

La operación módulo calcula el módulo de una división aritmética es decir calcula el residuo de una división, el cálculo del módulo se denota con el carácter de porcentaje, (%), la aplicación de esta operación se puede ver en el siguiente ejemplo:

int A;

int B;

int C;

C = A%B; //Está expresión guarda el residuo de la división de A entre B en la variable C.

Las operaciones aritméticas pueden ser usadas en forma combinada, es decir que se pueden mezclar varias operaciones en una misma expresión, para ver esto con mayor claridad observe los siguientes ejemplos:

int A;

int B;

int C;

C = (A+B)/C; //Está expresión es equivalente a C = (A+B)÷C.

*C = (A/B)*C; // Está expresión es equivalente a C = (A÷B) X C.*

Otros operadores matemáticos abreviados pueden ser utilizados cuando se requiere de conteos o cambios de una variable de forma constante, por ejemplo es posible incrementar o decrementar una variable en términos de un número, de igual manera es posible hacer esta operación con la multiplicación y la división. Para entender de forma más clara este concepto observe los siguientes ejemplos:

int A=100;

int B=10;

A++; //Este operador incrementa en una unidad el valor de A.

A--; //Este operador decrementa en una unidad el valor de A.

A+=4; //Este operador incrementa en 4 el valor de A.

A-=5; //Este operador decrementa en 5 el valor de A.

A/=4; //Este operador divide el valor de A en 4.

A=3; //Este operador multiplica el valor de A por 3.*

$A+=B;$ //Este operador incrementa el valor de A en el valor de B unidades.

$A*=B;$ //Este operador multiplica el valor de A por B veces.

Otras operaciones matemáticas de mayor complejidad se pueden realizar en lenguaje C, por medio de funciones predefinidas en la librería math, esta temática será tratada posteriormente cuando se estudie el uso y declaración de funciones.

Los operadores lógicos permiten realizar acciones u operaciones que respetan la lógica digital planteada por el matemático inglés George Boole, en el siglo XIX. Boole planteó las operaciones OR, AND, NOT, XOR, NOR, NAND, XNOR. Estas operaciones son ampliamente estudiadas en los cursos de sistemas digitales combinacionales y secuenciales.

Las operaciones lógicas se realizan en lenguaje C entre dos variables o constantes, estas se hacen bit a bit, del mismo peso en una variable o número. Para ver y entender los ejemplos recuerde primero las tablas de verdad de las operaciones lógicas que se muestran a continuación:

| Inversor NOT | |
|--------------|--------|
| Entrada | Salida |
| 0 | 1 |
| 1 | 0 |

| OR inclusiva | | |
|--------------|-----------|--------|
| Entrada 1 | Entrada 2 | Salida |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| NOR inclusiva | | |
|---------------|-----------|--------|
| Entrada 1 | Entrada 2 | Salida |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

| OR exclusiva o XOR | | |
|--------------------|-----------|--------|
| Entrada 1 | Entrada 2 | Salida |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| NOR exclusiva o XNOR | | |
|----------------------|-----------|--------|
| Entrada 1 | Entrada 2 | Salida |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| AND | | |
|-----------|-----------|--------|
| Entrada 1 | Entrada 2 | Salida |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| NAND | | |
|-----------|-----------|--------|
| Entrada 1 | Entrada 2 | Salida |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Las operaciones lógicas en lenguaje C, se realizan con caracteres específicos, que denotan cada una de ellas, en el siguiente ejemplo pueden verse las aplicaciones de los operadores lógicos:

Operación lógica NOT, negación, se denota con el carácter virgulilla (~);

unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.

unsigned short RESULTADO;

RESULTADO = ~VALOR1; //La variable RESULTADO guarda el complemento de //VALOR1, 175.

Operación lógica OR, o inclusiva, está se denota con el carácter barra vertical (|);

unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.

unsigned short VALOR2=0b01011111; //Variable inicializada en binario con el número 95.

unsigned short RESULTADO;

RESULTADO = VALOR1|VALOR2; //El valor de la operación lógica o, es guardado en //RESULTADO, 95.

Operación lógica XOR, o exclusiva, está se denota con el carácter acento circunflejo (^);

unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.

unsigned short VALOR2=0b01011111; //Variable inicializada en binario con el número 95.

unsigned short RESULTADO;

RESULTADO = VALOR1^VALOR2; //El valor de la operación lógica AND //es guardado en RESULTADO, 15.

Operación lógica AND, y, está se denota con el carácter ampersand (&);

unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.

unsigned short VALOR2=0b01011111; //Variable inicializada en binario con el número 95.

unsigned short RESULTADO;

RESULTADO = VALOR1&VALOR2; //El valor de la operación lógica o //exclusiva, es guardado en RESULTADO, 80.

La implementación de las operaciones lógicas NAND, NOR, XNOR, son similares a AND, OR, y XOR, a agregando el carácter de negación virgulilla, observe los siguientes ejemplos:

unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.

unsigned short VALOR2=0b01011111; //Variable inicializada en binario con el número 95.

unsigned short RESULTADO;

RESULTADO = ~(VALOR1|VALOR2); //El valor de la operación lógica o negada //exclusiva, es guardado en RESULTADO.

RESULTADO = ~(VALOR1&VALOR2); //El valor de la operación lógica y negada //es guardado en RESULTADO.

RESULTADO = ~(VALOR1^VALOR2); //El valor de la operación lógica o negada //exclusiva negada, es guardado en RESULTADO.

El desplazamiento de bits dentro de una variable es útil para realizar procesos y tareas que impliquen la manipulación de datos a nivel de los bits. El corrimiento a la derecha en una variable o valor constante, en lenguaje C se realiza por medio del doble carácter mayor que, (>>), de la misma manera el corrimiento a la izquierda se ejecuta con el doble carácter menor que, (<<). La operación de corrimiento hace perder los valores de los bits que salen, e ingresa ceros en

los nuevos bits. En los siguientes ejemplos se puede observar la implementación de estos operadores:

```
short Dato=0xFF; //Declaración de variables.
```

```
short Resultado;
```

```
Resultado = Dato>>4; //Está operación guarda en la variable Resultado el corrimiento de 4 bits  
//a la derecha de la variable Dato, valor final de Resultado es 0x0F.
```

```
Resultado = Dato<<4; //Está operación guarda en la variable Resultado el corrimiento de 4 bits  
//a la izquierda de la variable Dato, valor final de Resultado es 0xF0.
```

En la siguiente tabla se resumen las operaciones lógicas y aritméticas contempladas anteriormente:

| Descripción de la operación | Operador |
|--|----------|
| Suma | + |
| Restá | - |
| División | / |
| Módulo o residuo de la división entera | % |
| Incremento | ++ |
| Decremento | -- |
| Operación OR | / |
| Operación AND | & |
| Operación XOR | ^ |
| Complemento a 1 | ~ |
| Corrimiento a la derecha | >> |
| Corrimiento a la izquierda | << |

Tabla 2-2

2.4 Funciones en lenguaje C

Una función es una fracción de código que realiza una tarea específica cada vez que está es invocada por el flujo del programa principal. Las funciones cuentan con dos características fundamentales, uno o más parámetros de entrada, y un parámetro de salida. Cualquiera de estas dos características puede ser una variable o un arreglo de ellas, de igual manera estos parámetros de entrada y salida pueden ser vacíos.

La estructura de las funciones se puede apreciar en los siguientes ejemplos:

```
void Funcion ( void ) //Función con parámetros de entrada y salida vacíos.  
{ //Apertura de la función con corchete.  
    //Porción de código donde se ejecutan la rutina de la función.  
} //Cierre de la función con corchete.
```

```
void Funcion ( int A ) //Función con un parámetro de entrada y salida vacía.  
{ //Apertura de la función con corchete.  
    //Porción de código donde se ejecutan la rutina de la función.  
} //Cierre de la función con corchete.
```

```

int Funcion ( void ) //Función con parámetro de entrada vacío y salida entera.
{
    //Apertura de la función con corchete.
    //Porción de código donde se ejecutan la rutina de la función.
}
//Cierre de la función con corchete.

int Funcion ( int A ) //Función con un parámetro de entrada y salida enteras.
{
    //Apertura de la función con corchete.
    //Porción de código donde se ejecutan la rutina de la función.
}
//Cierre de la función con corchete.

```

Los nombres que se designan a las funciones cumplen con las mismas reglas para nombrar las variables. El siguiente ejemplo muestra una función que es capaz de calcular el producto de dos números con punto decimal:

```

float Producto ( float A, float B ) //Función para calcular el producto de A y B.
{
    //Apertura de la función con corchete.
    float RESULTADO;
    RESULTADO = A*B; //Producto de A y B.
    return RESULTADO; //La función retorna el resultado guardado en RESULTADO.
}
//Cierre de la función con corchete.

```

Una función puede recurrir a otra función para realizar funciones más complejas, para demostrar esta situación se puede ver el siguiente ejemplo, que realiza el cálculo del área de una circunferencia en función de su radio:

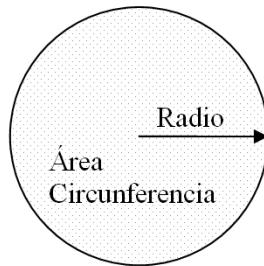


Figura 2-1

El área de la circunferencia se calcula por medio de la ecuación $A=\pi r^2$, donde el valor de π es aproximadamente 3,1416.

```

float Valor_PI ( void ) //Función que retorna el valor de  $\pi$ .
{
    //Apertura de la función con corchete.
    float PI=3.1416;
    return PI;
}
//Cierre de la función con corchete.

```

En el caso en que una función requiera variables internas está declaración se debe hacer al principio de la misma, antes de realizar cualquier otra acción o instrucción.

```

float Area_Circunferencia ( float Radio ) //Función para calcular el área del círculo.
{
    //Apertura de la función con corchete.
    float Area;
}

```

```

Area = Valor_PI()*Radio*Radio; //Cálculo del área del círculo.
return Area;
} //Cierre de la función con corchete.

```

Las funciones son creadas por el desarrollador, sin embargo existen algunas que están predefinidas en el compilador y pueden ser consultadas e implementadas respetando los parámetros de entrada y salida, la más importante de ellas es la función *main* o principal. La importancia de esta función radica en el hecho de que es sobre ella que corre todo el programa del microcontrolador y está se ejecuta automáticamente cuando el microcontrolador se energiza. La función *main* en el caso particular de los microcontroladores no contiene parámetros de entrada ni de salida, su declaración se puede apreciar en el siguiente ejemplo:

```

void main( void ) //Declaración de la función main.
{
    //Apertura de la función main.
    //Código del programa principal del microcontrolador.
} //Cierre de la función main.

```

Las declaraciones de funciones hechas por el desarrollador deben ser declaradas antes de la función *main*, y en el caso de una función que invoque a otra función debe ser declarada antes de la función que hace el llamado.

Las funciones matemáticas trigonométricas y otras como logaritmos, exponenciales, y potenciación pueden ser usadas con la librería predefinida por el compilador. Esta librería se denomina *C_Math*.

2.5 Creación de un programa en lenguaje C

La estructura de un programa en lenguaje C es relativamente simple, primero es indispensable declarar las variables globales que el desarrollador considere necesarias para el funcionamiento del programa, estas variables globales son reconocidas por todos los puntos de código del programa incluidas las funciones propias del desarrollador y la función *main*. El paso a seguir es hacer las declaraciones de funciones diseñadas por el desarrollador para las tareas específicas en su programa. Posteriormente se declara la función *main* y al comienzo de ésta se deben declarar las variables que se requieran dentro de la misma. El código que sigue debe configurar e inicializar los puertos y módulos del microcontrolador que sean indispensables en la aplicación. Por último se edita el código que contiene el aplicativo concreto del programa. En el siguiente ejemplo se puede ver cómo hacer una estructura para un programa:

```

int Variable1; //Declaración de variables globales.
char Variable2;
float Variable3;

//Declaración de funciones propias del desarrollador.
float Valor_PI ( void )
{
    float PI=3.1416;
    return PI;
}

```

```

float Calculo_Area_Circulo( float Radio )
{
    float Area;
    Area = Valor_PI()*Radio*Radio; //Cálculo del área.
    return Area;
}

void main( void ) //Declaración de la función main o principal
{
    float Valor_Radio; //Declaración de variables de la función main.
    float Valor_Area;
    TRISB=0; //Configuración de puertos y módulos.
    TRISC=0;
    PORTB=0;
    PORTC=0;
    while( 1 ) //Código concreto del aplicativo.
    {
        Valor_Area=Calculo_Area_Circulo( Valor_Radio );
    }
}

```

2.6 Condicionales e iteraciones en lenguaje C

Las formas condicionales e iterativas hacen parte vital de las estructuras de cualquier código en lenguaje C. Estás permiten hacer ciclos en función de conteos y condiciones de variables que hacen efectivo el flujo del programa. Las sentencias condicionales *if*, *if else*, y *switch case*, permiten realizar menús, tomar decisiones, y controlar el flujo del programa o aplicativo. Las estructuras iterativas *while*, *do while* y *for*, permiten crear bucles iterativos, que cuentan o gobiernan tareas y procesos desarrollados en el programa.

2.6.1 Creación de condiciones lógicas en lenguaje C

Las condiciones lógicas permiten tomar decisiones en función de una evaluación. Las condiciones lógicas solo retornan dos posibles valores: falso o verdadero. Cuando la condición lógica se hace directamente sobre una variable o número, la condición es falsa cuando su valor es 0 y retorna verdadero para cualquier otro valor diferente de 0. Las formas condicionales usadas en lenguaje C son: NOT, o negación, OR o o inclusiva, AND o y. También se pueden usar comparaciones de magnitud entre dos valores tales como: mayor que, menor que, mayor o igual que, menor o igual que, diferente que, e igual que. El uso de la negación se realiza con el carácter admiración (!), la condición OR, se realiza con los caracteres doble barra vertical (||), la condición AND o y, se usa con los caracteres doble ampersand (&&), la condición mayor que se usa con el carácter que tiene su mismo nombre (>), la condición menor que, se hace con el carácter menor que (<), la condición mayor o igual que, se realiza con los caracteres mayor que, e igual (>=), la condición menor que, se realiza con los caracteres menor que, e igual (<=), la condición de igualdad se hace con los caracteres doble igual (==), y la condición diferente que, se usa con los caracteres de igualdad y admiración (!=). Para entender con claridad estos operadores observe y analice los siguientes ejemplos:

short A;

```

short B;
//La condición es verdadera si A vale 10 y B vale 20.
(A==10)&&(B==20)
//La condición es verdadera si A es diferente de 5 y B es mayor que 2.
(A!=5)&&(B>2)
//La condición es verdadera si A es menor o igual que 4 o si B es mayor o igual que 6.
(A<=4)&&(B>=6)
//La condición es verdadera si A no es mayor que 3 y si B es diferente de 4.
!(A>3)&&(B!=4)

```

2.6.2 La sentencia condicional if e if else

La estructura de la sentencia *if* evalúa una condición lógica y ejecuta una porción de código si y solo si el resultado de la evaluación es verdadera. Observe la estructura del la sentencia *if* en el siguiente ejemplo:

```

short Valor;
if( Valor>100 ) //Este if evalúa si la variable Valor es mayor que 100.
{
    // Porción de código que se ejecuta si el if es verdadero.
}

```

La estructura *if else* es igual que la sentencia *if*, su única diferencia es que en el dado caso de que la evaluación de la condición sea falsa se ejecuta la porción de código asociada al *else*. Para entender esta situación observe el siguiente ejemplo:

```

short Valor;
if( Valor>100 ) //Este if evalúa si la variable Valor es mayor que 100.
{
    // Porción de código que se ejecuta si la condición del if es verdadero.
}
else
{
    // Porción de código que se ejecuta si la condición del if es falsa.
}

```

Las sentencias *if* e *if else* se pueden anidar para crear estructuras más completas y complejas, está característica se puede ver con claridad en el siguiente ejemplo:

```

short Valor1; //Declaración de variables.
short Valor2;
if( Valor1>30 ) //Sentencia if.
{
    // Código que se ejecuta cuando el primer if tiene una condición verdadera.
    if( (Valor2==4)&&(Valor1<50) ) //Segunda sentencia if anidada.
    {
        //Código que se ejecuta cuando el segundo if tiene una condición verdadera.
    }
}

```

2.6.3 La sentencia switch case

La sentencia *switch case*, funciona de manera similar a la sentencia *if*, difiere en que no se evalúa una condición si no el valor de una variable, y ejecuta una porción de código para cada valor posible de la variable. Los valores contemplados en los casos o *case* posibles de la variable los escoge el desarrollador arbitrariamente en función de su criterio y necesidad. Los casos o valores que no son de interés para el desarrollador se ejecutan en un fragmento de código por defecto, por medio de la directriz *default*. Cada uno de los fragmentos de código editados deben terminar con la directriz *break* para romper el flujo de la estructura *switch case*, esta acción es necesaria dado que si dos casos están seguidos los fragmentos de código seguidos se ejecutarán hasta encontrar la directriz *break* o hasta finalizar la estructura *switch case*. Para comprender de forma clara el funcionamiento de la sentencia *switch case* observe el siguiente ejemplo:

```
short Valor;
switch( Valor ) //Evaluación de la variable Valor.
{
    case 0: //Fragmento de código correspondiente al valor 0 de la variable Valor.
        break; //Ruptura del caso 0.
    case 1: //Fragmento de código correspondiente al valor 1 de la variable Valor.
        break; //Ruptura del caso 1.
    case 10: //Fragmento de código correspondiente al valor 10 de la variable Valor.
        break; //Ruptura del caso 10.
    case 20: //Fragmento de código correspondiente al valor 20 de la variable Valor.
        break; //Ruptura del caso 20.
    case 50: //Fragmento de código correspondiente al valor 50 de la variable Valor.
        break; //Ruptura del caso 50.
    default: //Código correspondiente para cualquier otro valor por defecto.
        break; //Ruptura del default.
}
```

2.6.4 El ciclo iterativo while y do while

El ciclo iterativo *while* repite o ejecuta un fragmento de programa siempre y cuando la condición contenida en el *while* sea verdadera. Para conocer la forma de declarar este tipo de ciclo observe el siguiente ejemplo:

```
short CONT=0; //Declaración de variable entera que cuenta hasta 100.
while( CONT<100 )//Declaración del ciclo while.
{
    CONT++; //Incremento de la variable CONT.
}
```

La implementación de un ciclo *do while*, es similar al ciclo *while*, con la diferencia puntual de que en este ciclo el fragmento de código se ejecuta y después evalúa la condición del *while*. En conclusión el código se ejecuta por lo menos una vez antes de evaluar la condición. En el siguiente ejemplo se puede ver la forma de usar este tipo de ciclo:

```
short CONT=0; //Declaración de variable entera que cuenta hasta 100.
do //Declaración del ciclo do while.
{
```

```

    CONT++; //Incremento de la variable CONT.
} while( CONT<100 );

```

2.6.5 El ciclo iterativo for

El ciclo iterativo *for* es una estructura parecida al ciclo *while*, la diferencia es una estructura más compleja en los paréntesis de la condición. La condición cuenta con tres parámetros, el primero es un espacio de código para dar valores iniciales a una o más variables, el segundo campo permite crear una condición lógica que hace repetir el fragmento de código del *for* cuando está es verdadera, el último espacio realiza cambio sobre una o más variables, tal como: incrementos, decrementos, etc. Las variables que se cambian en el último campo son generalmente las mismas que se inician en el primer campo, pero está no es una regla. Observe la estructura del *for* en el siguiente modelo:

```

for( _INICIO_ ; _CONDICION_ ; _CAMBIO_ )
{
}

```

Para conocer el funcionamiento del ciclo *for* observe el siguiente ejemplo:

```

short A; //Declaración de variables.
short B;
for( A=0, B=100; (A<100)//(B>20); A++, B-- ) //Estructura del ciclo for.
{
    //Fragmento de código que se ejecuta cuando la condición del for es verdadera.
}

```

2.6.6 Uso anidado de ciclos iterativos

Muchas de las aplicaciones incrementan su funcionalidad y optimizan el tamaño del código de máquina final al usar los ciclos iterativos de forma anidada. La anidación de ciclos consiste en ejecutar un ciclo dentro de otro. Un ciclo puede contener uno o más ciclos anidados. Para comprender este concepto de mejor manera observe el siguiente ejemplo:

```

short COL; //Declaración de variables.
short FIL;
short MATRIZ[10][10]; //Declaración de un vector bidimensional.
for( COL=0; COL<10; COL++ ) //Declaración de ciclo for.
{
    //Fragmento del primer ciclo.
    for( FIL=0; FIL<10; FIL++ ) //Declaración de ciclo for anidado.
    {
        //Fragmento del ciclo anidado.
        MATRIZ[COL][FIL] = COL*FIL;
    }
}

```

El ejemplo anterior, guarda valores en una matriz de 10 por 10, recorriendo una a una las posiciones de la misma.

3 El simulador ISIS de Proteus

El simulador ISIS de Proteus es un poderoso paquete de software, desarrollado por la compañía labcenter electronics, que se ha posicionado desde hace mas de 10 años, como una de las herramientas más útiles para la simulación de los microcontroladores PICMicro. El ISIS permite la simulación de las familias de los PICMicro más populares tales como la: 12F, 16F, 18F. Además de los PIC, el ISIS puede simular una gran variedad de dispositivos digitales y analógicos, entre los dispositivos digitales es posible simular displays de siete segmentos, de caracteres, y gráficos. ISIS puede simular sensores de temperatura, humedad, presión, y luminosidad, entre otros. El simulador permite, simular actuadores como: motores dc, servo motores, luces incandescentes entre otros. Es posible simular periféricos de entrada y salida como teclados, y puertos físicos del ordenador como: RS232, y USB. Este simulador cuenta con una amplia variedad de instrumentos de medición como voltímetros, amperímetros, osciloscopios, y analizadores de señal. En conclusión estas y otras características hacen del ISIS de Proteus, una herramienta ideal para el diseño y estudio de los PICMicro. Una versión demostrativa del paquete de software se puede descargar de la página: www.labcenter.com. En la siguiente imagen se puede apreciar la apariencia visual del entorno de desarrollo del ISIS:

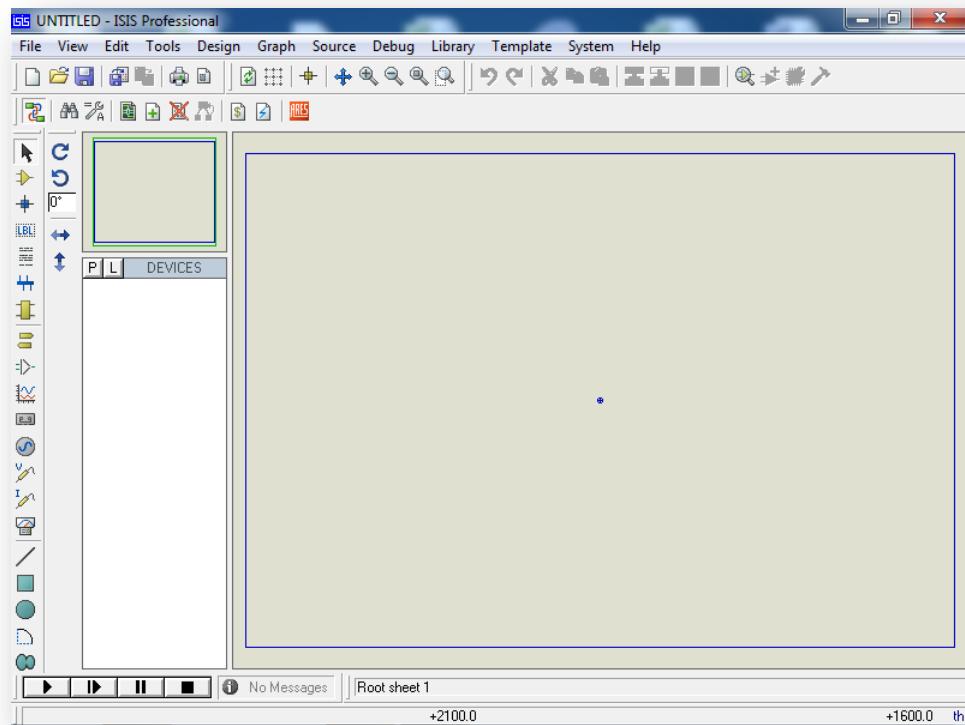


Figura 3-1

El uso y trabajo bajo en el torno de ISIS demanda una amplia cantidad de herramientas y opciones que se deben conocer pero se conocerán paulatinamente en el transcurso de los ejemplos.

3.1 Características básicas del ISIS para simular

En la siguiente sección se mostrarán las características básicas para hacer la primera simulación en ISIS. Como primera medida se debe identificar la paleta de dispositivos, que está a la izquierda de la pantalla, en esta paleta el desarrollador debe identificar el botón P, para mayor claridad observe la siguiente imagen:

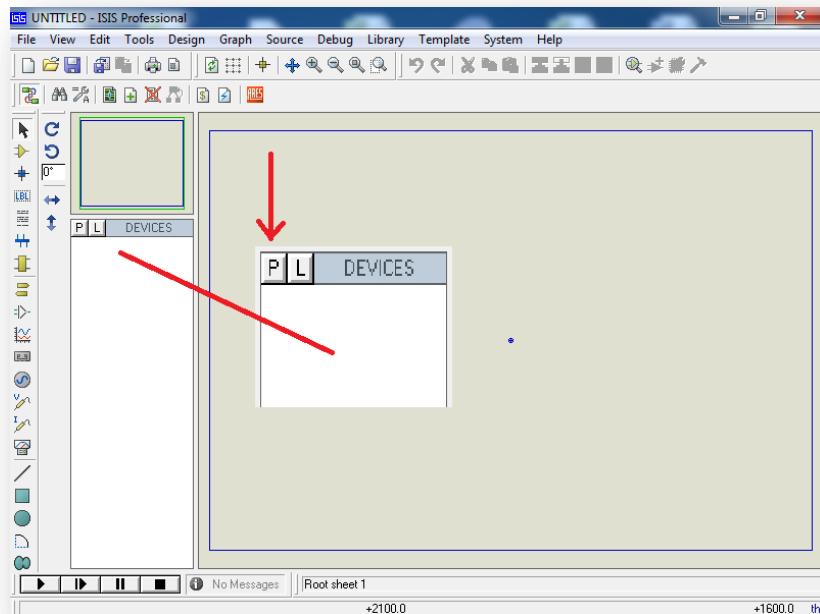


Figura 3-2

Al picar el botón P, el programa abre una nueva ventana que permite buscar los dispositivos electrónicos por medio de la referencia comercial, o bajo la clasificación que el mismo ISIS tiene. Para buscar un dispositivo por medio de la referencia se digita la referencia en la casilla: Keywords, el programa genera una lista en la parte derecha de la ventana con los dispositivos relacionados a la solicitud del usuario. Para seleccionar un dispositivo de esta lista se da doble clic sobre cada uno de los numerales de la lista. Para iniciar busque los siguientes dispositivos: BUTTON, LED-RED, RES, que corresponden a un pulsador, un LED de color rojo, y una resistencia. Después de este proceso, en la paleta de dispositivos debe verse lo siguiente:

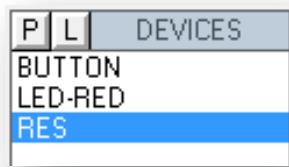


Figura 3-3

El paso siguiente es buscar las terminales de referencia y poder, para este fin se debe pulsar el ícono siguiente: este se encuentra en la paleta de herramientas que está en la parte izquierda de la ventana del programa. Cuando este botón se pica permite ver una lista de terminales, en las

cuales se encuentra GROUND, y POWER, las cuales corresponden respectivamente a la referencia eléctrica y al poder o Vcc. La terminal de poder tiene una diferencia de potencial por defecto de 5 voltios. Para colocar estas terminales en el área de trabajo se pica los ítems en la paleta de terminales y posteriormente en el área de trabajo, después de esta acción en el área de trabajo se debe ver lo siguiente:

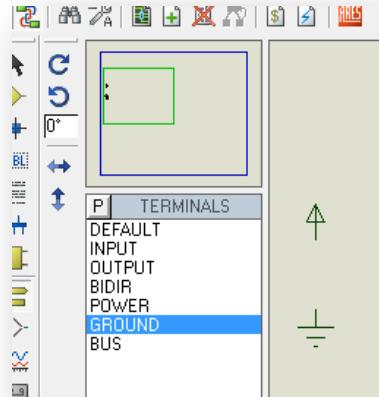


Figura 3-4

El paso siguiente es pegar los dispositivos en el área de trabajo para esto pique el botón: que se encuentra en la paleta de herramientas de la izquierda. Para pegar los dispositivos en el área de trabajo se sigue el mismo procedimiento de las terminales, al finalizar se debe tener la siguiente vista:

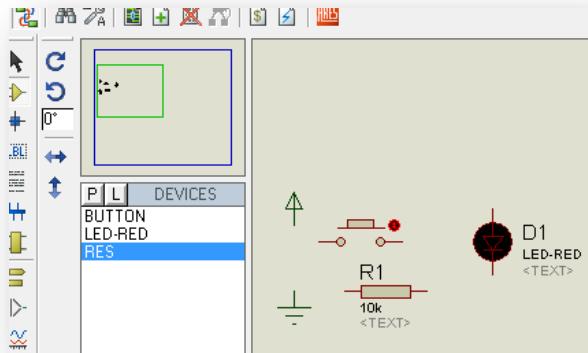


Figura 3-5

El paso siguiente es realizar las conexiones eléctricas, para este ejemplo se conectan todos los elementos en serie, para este fin el cursor del ratón adopta la forma de lápiz, para hacer la conexión entre dos terminales se pica una terminal y después la siguiente. El programa termina rutiando la conexión. Al terminar las conexiones se ve la siguiente vista:

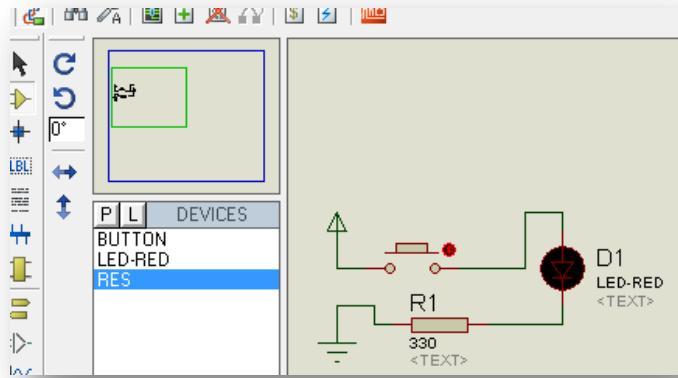


Figura 3-6

Para finalizar se puede cambiar el valor de la resistencia dando clic izquierdo sobre la resistencia, con esta acción sale una ventana que permite editar el valor de la resistencia, que por defecto es de 10k, lo que se debe hacer es cambiarlo por 330.

El paso siguiente es ejecutar o correr la simulación para esto se utiliza la paleta de reproducción que está en la parte inferior izquierda de la pantalla. La apariencia de esta paleta es la siguiente:



Figura 3-7

Por último se pica el botón de play y la simulación se ejecuta, cuando la simulación este corriendo se puede pulsar el BUTTON, y ver el efecto de la simulación. Este simple ejemplo permite mecanizar el proceso de creación de un circuito digital, para su posterior simulación.

La creación de circuitos en ISIS, implica hacer otra serie de acciones, que serán tratadas en el transcurso de los ejemplos.

4 Creación del primer programa en MikroC PRO

El proceso siguiente debe ser mecanizado para ser implementado cada vez que se haga un proyecto o programa nuevo para un PICMicro. Al correr el MikroC PRO, se identifica en el menú superior el ítem *Project*, se pica y dentro de este se pica *New Project...*, con esta acción el programa despliega un asistente fácil de usar para crear el nuevo proyecto. La apariencia visual de este asistente es la siguiente:



Figura 4-1

La siguiente acción es pulsar el botón *Next*, con este paso el asistente muestra una casilla para seleccionar la referencia del PICMicro, que se desea usar. En esta opción seleccione el PIC P16F84A. El siguiente paso es definir la frecuencia de oscilación con la cual trabajará el PIC, en este ejemplo se selecciona 4.000000 MHz. La siguiente opción permite definir el directorio en donde el desarrollador guardará el proyecto, en este directorio el programa guardará todos los archivos requeridos, en los cuales la fuente del código será el archivo con extensión .c, y el ejecutable del PIC es el archivo .hex. El siguiente paso solicita adicionar archivos que serán anexados al proyecto. Cuando se realiza un proyecto nuevo este paso se puede omitir y se pulsa *Next*. El último ítem del asistente pregunta si el desarrollador quiere seleccionar las librerías que usará en este trabajo, por defecto este ítem selecciona todas las librerías habilitadas para este PIC,

lo mejor es dejar todas las librerías activas. Por último se termina la configuración y se crea el proyecto, al terminar debe aparecer una vista como la siguiente:

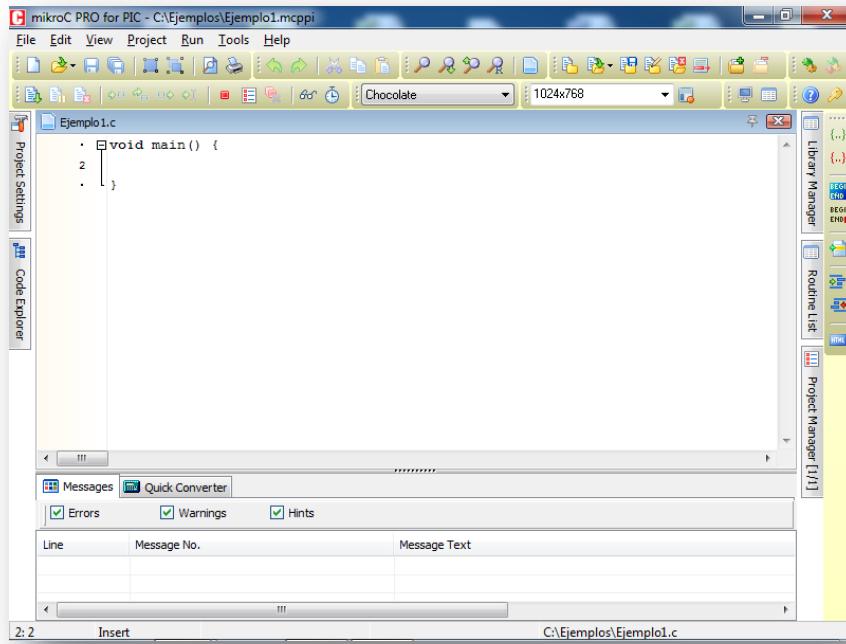


Figura 4-2

Cuando un cambio se realice sobre el código del programa se debe compilar el código picando el siguiente botón: que está ubicado en la parte superior del programa dentro de la paleta de herramientas. Esta acción genera los resultados de la compilación que se encuentran en la parte inferior de la ventana del programa. Los mensajes se deben terminar con un texto de finalización exitosa.

Para iniciar la edición del proyecto, se configuran los puertos del PIC, y posteriormente se encierra el programa en un bucle infinito. El PIC 16F84A, tiene dos puertos él A y el B, para configurar los puertos como salida o como entrada se manipula el registro TRIS. Cada puerto cuenta con su respectivo registro TRIS, que hace alusión a los tres estados posibles, alto, bajo, y alta impedancia. Los registros TRIS cuentan con el mismo número de bits del puerto, por ejemplo el puerto B o PORTB de este PIC tiene 8 bits, por lo tanto el TRISB también tiene 8 bits. Los bits de los registros TRIS, son correspondientes al puerto, y definen bit a bit el estado del puerto. Si un bit en el TRIS es 0 el mismo bit en el puerto es de salida, y si el bit del TRIS es 1 el mismo bit del puerto es de entrada o está en alta impedancia. Para entender este concepto con mayor claridad observe y analice el siguiente ejemplo:

TRISB = 0b11110000; // Configura los cuatro bits de menor peso como salida, y //los cuatro bits de mayor peso como entrada.

PORTB=0b00000000; //Los bits de salida asumen un 0 lógico.

Este ejemplo usa un pulsador y un par de LEDs para ver el comportamiento del programa. Observe y analice el programa a continuación:

void main (void)

```

{
    unsigned int CONTADOR=0;
    TRISB = 0b11110000; // Configura los cuatro bits de menor peso como salida, y
    //los cuatro bits de mayor peso como entrada.
    PORTB=0b00000000; //Los bits de salida asumen un 0 lógico.
    while( 1 )//Bucle infinito
    {
        if( PORTB.F7==0 )//Evalúa si bit RB7 es 0
        {
            if( PORTB.F0==1 )//Evalúa el valor del bit RB0 y conmuta su valor.
                PORTB.F0=0;
            else
                PORTB.F0=1;
            while( PORTB.F7==0 ); //Espera a que el RB7 cambie a 1.
        }
        CONTADOR++; //Incrementa el valor del CONTADOR.
        //La siguiente condición if cambia automáticamente el estado del bit RB1
        if( CONTADOR&0x0100 )//Evalúa si el bit 8 del CONTADOR es 1
            PORTB.F1=1;
        else
            PORTB.F1=0;
    }
}

```

El paso siguiente es hacer la simulación en ISIS, las resistencias de los LEDs deben ser cambiadas a 330Ω , la terminal Master Clear, MCLR debe ser conecta a Vcc para que el PIC, no se reinicie al terminar se debe ver de la siguiente forma:

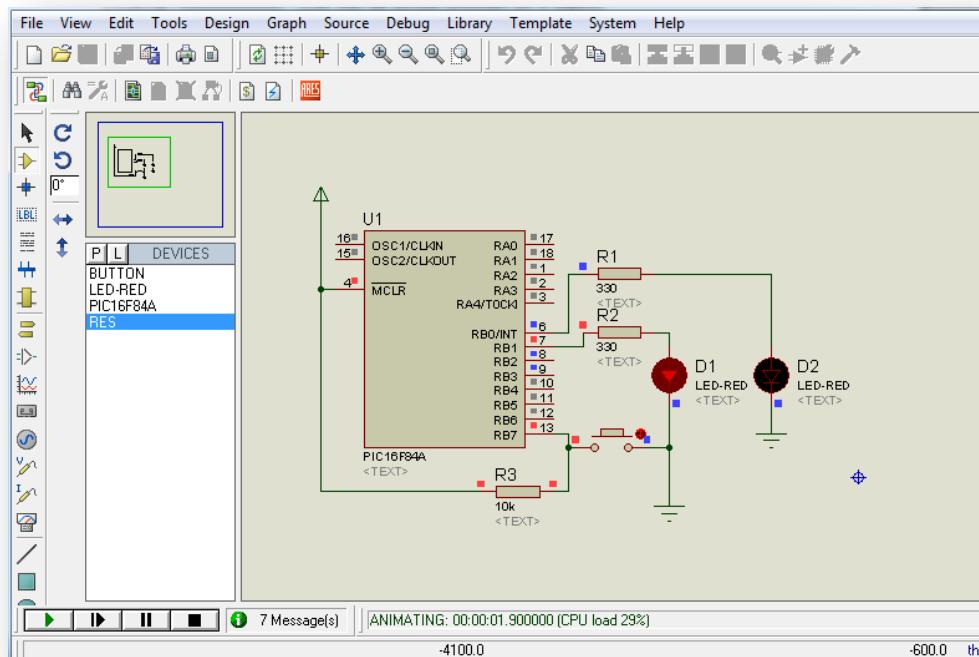


Figura 4-3

Antes de correr la simulación se debe cargar el archivo .hex, para este proceso se hace clic izquierdo sobre el PIC, y aparece una ventana que permite buscar el archivo .hex, en esta ventana también se ajusta la frecuencia de oscilación. Por defecto este valor es 1MHz, para efectos de simulación en este caso se debe seleccionar 4MHz, la vista de esta ventana es la siguiente:

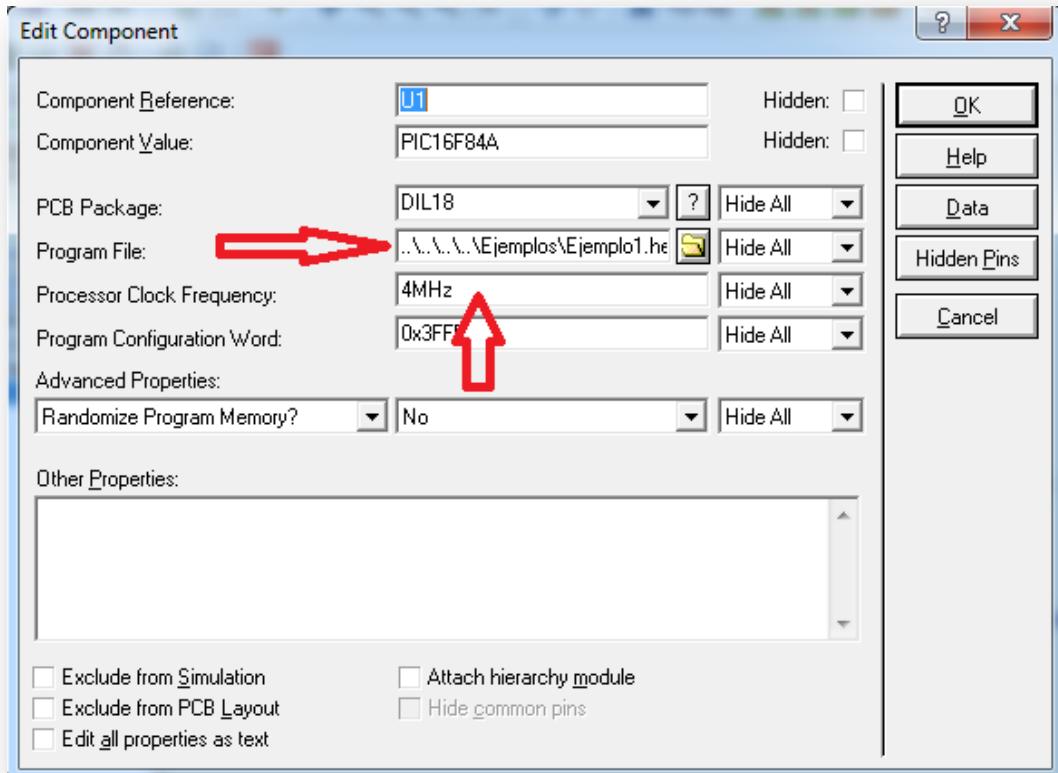


Figura 4-4

Para fines de la programación física del microcontrolador es importante tener presente las condiciones de configuración del PIC, cada uno de los microcontroladores de Microchip, poseen un campo de memoria que caracteriza el comportamiento de real del PIC, entre estos se pueden destacar; el tipo de oscilador o reloj de procesamiento, por ejemplo cuando se usa un cristal de 4M Hz, el oscilador del PIC se debe configurar como XT, si el oscilador es de 20M Hz se usa la configuración HS, si se trata de un cristal de baja velocidad como 400K Hz se usa la configuración LP, y si se implementa un oscilador por medio de circuito RC, o circuito de resistencia en serie con capacitor se usa la opción RC. De la misma forma es posible configurar otras opciones como el uso del perro guardián, la protección de código en memoria FLASH y EEPROM, esta última opción es de suma importancia para proteger la información del microcontrolador y evitar que el programa o los datos de la memoria sean leídos. Para modificar la configuración del PIC, se debe buscar el ítem *Edit Project...* que se encuentra en la pestáña *Project*, del menú principal del programa MikroC PRO.

El acceso a este ítem se puede apreciar en la siguiente figura:

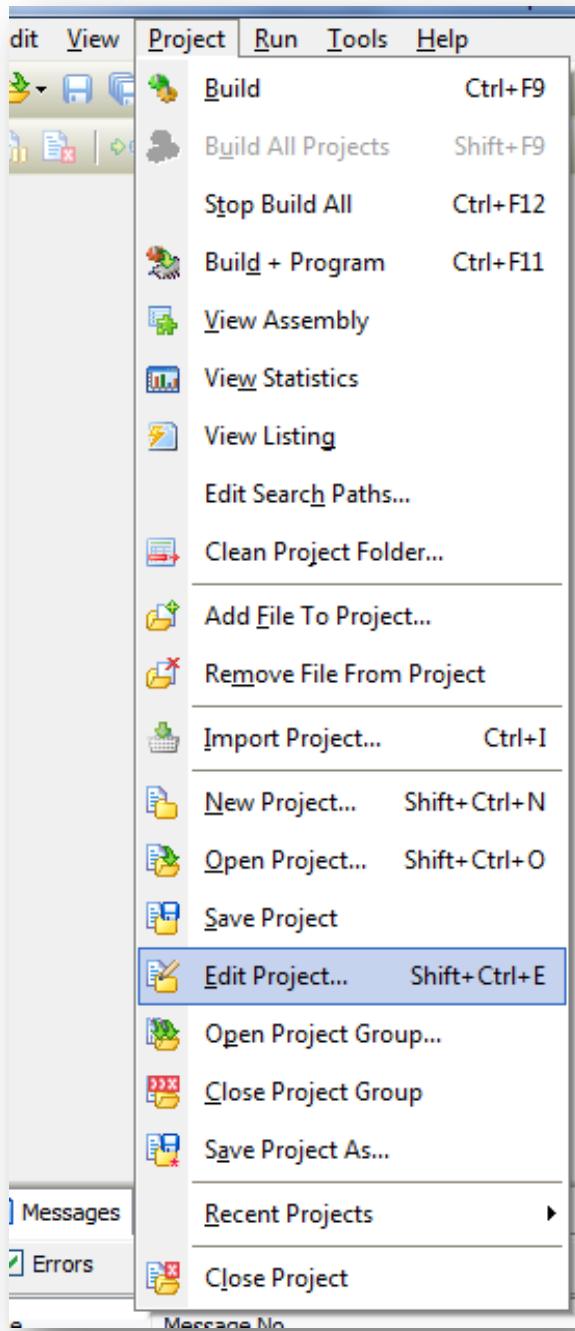


Figura 4-5

Después de picar esta opción el compilador MikroC PRO, despliega una ventana con casillas de selección para editar las opciones de configuración del PIC, en función del microcontrolador está ventana puede cambiar de apariencia dado que no todos los PIC, cuentan con las mismas configuraciones, sin embargo en la siguiente figura se puede apreciar la apariencia de esta ventana para un PIC 16F877A:

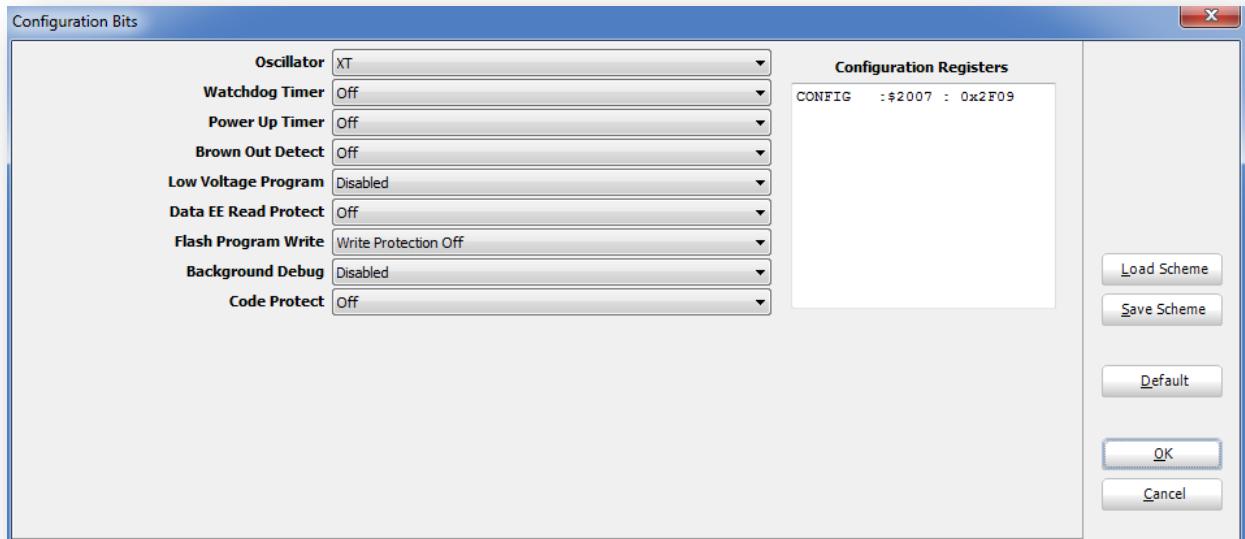


Figura 4-6

5 Visualización de datos

El diseño de sistemas microcontrolados implican en algunos casos la visualización de datos al usuario, para este fin se pueden usar display 7 segmentos, display LCD de caracteres y display gráficos LCD. En este capítulo se estudia y se ejemplarizan estos dispositivos.

5.1 Display de 7 segmentos

Un display 7 segmentos es un dispositivo que permite visualizar un número limitado de caracteres esencialmente numéricos, sin embargo es posible visualizar unos pocos caracteres mas como: b, d, E, A, o, F, C, -. Los display 7 segmentos son un arreglo de diodos LED, organizados de tal forma que permiten visualizar los caracteres según los segmentos que estén activos. Los displays de 7 segmentos tienen una designación estándar de cada segmento que es consecutiva de la 'a' hasta la 'g'. Esta designación y la apariencia física de estos displays se pueden apreciar en las siguientes figuras:

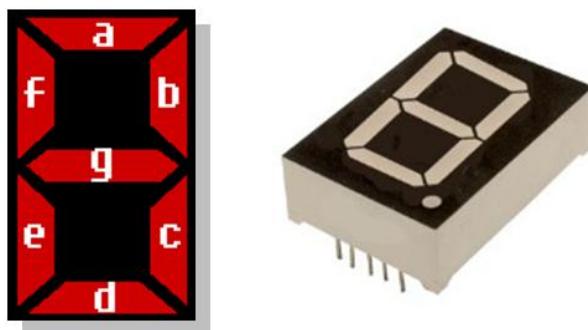


Figura 5-1

Los display de 7 segmentos son fabricados en dos formatos; de ánodo común y de cátodo común, los display de 7 segmentos también existen en un formato dinámico, estos últimos usan dos o más dígitos en un solo encapsulado conectando todos los segmentos en paralelo pero con los terminales comunes por separado. Las siguientes figuras muestran displays de 7 segmentos en su formato dinámico:

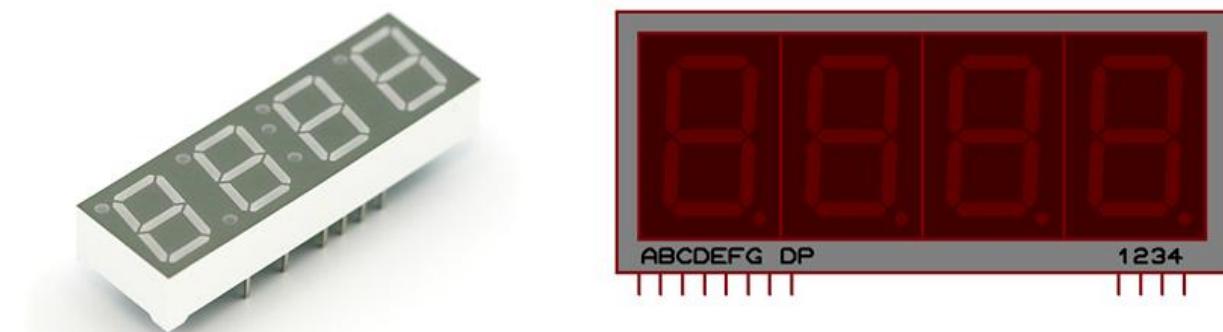


Figura 5-2

La terminal DP, que se puede apreciar en la figura anterior es un octavo segmento que en algunos displays es implementado y corresponde al punto decimal, este se usa si la aplicación así lo requiere.

5.1.1 Control de display de 7 segmentos

Para realizar el primer ejemplo con este display se debe conocer como realizar la asignación de pines del PIC, a cada uno de los segmentos del display se debe conectar un pin del PIC, para hacer un diseño optimo se puede asignar los pines en orden consecutivo de un puerto por ejemplo el segmento ‘a’ se conecta al pin RB0, el segmento ‘b’ al pin RB1, y así sucesivamente hasta el segmento ‘g’ al pin RB6. Sin embargo la designación de pines la puede hacer el desarrollador de manera arbitraria. Es importante conocer una herramienta con la que cuenta el paquete de software MikroC PRO, que permite editar los dígitos del display, para este fin se pica el ítem *Tools* en el menú del MikroC PRO, dentro de este nuevo menú se pica el ítem *Seven Segment Editor*. Con esta acción emerge una nueva ventana que permite editar de manera simple los segmentos del display de 7 segmentos. A medida que se edita el display aparece el valor constante que debe ser usado en las salidas del puerto que se designe para su control. El editor permite implementar las constantes para un display de ánodo o de cátodo común, de la misma forma es posible usar las constantes en formato decimal o hexadecimal. La apariencia visual del editor es la siguiente:

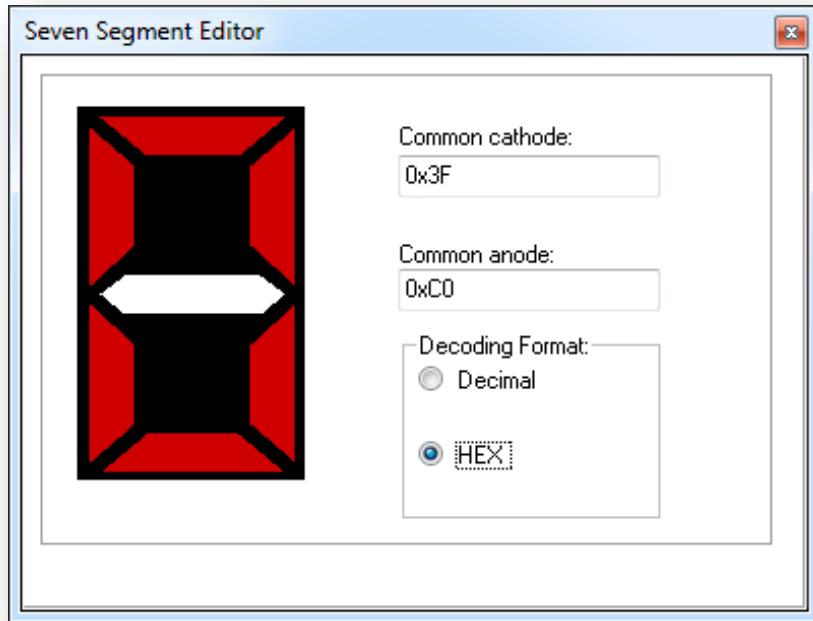


Figura 5-3

El uso de esta herramienta, implica que todos los pines del display estén asignados a un mismo puerto, y en orden consecutivo como se comentó anteriormente. Para la visualización de los dígitos en el display se debe organizar la información de los dígitos en orden consecutivo, en el caso de este ejemplo del 0 al 9. Para este fin la forma de mayor simplicidad es con el uso de una cadena de datos que contenga los 10 códigos de los dígitos. En el siguiente ejemplo se declarará un arreglo constante con los códigos de cada dígito, para este ejemplo se usará un display de

cátodo común. Observe la forma de declarar las constantes que deben ser insertadas antes de la función *main*:

```
const unsigned short DIGITOS[] =  
{  
    0x3F, //Código del dígito 0  
    0x06, //Código del dígito 1  
    0x5B, //Código del dígito 2  
    0x4F, //Código del dígito 3  
    0x66, //Código del dígito 4  
    0x6D, //Código del dígito 5  
    0x7D, //Código del dígito 6  
    0x07, //Código del dígito 7  
    0x7F, //Código del dígito 8  
    0x6F, //Código del dígito 9  
};
```

Para visualizar los dígitos en el display de 7 segmentos, este ejemplo usará el puerto B del microcontrolador PIC 16F84A, la visualización de los dígitos estará controlada por una rutina que cambia temporizadamente por medio de la función *delay_ms* que está predefinida por las librerías del compilador. Esta función tiene como parámetro de entrada un número entero que representa el tiempo en milisegundos, durante los cuales el PIC ejecutará un retardo, de la misma forma se puede usar la función *delay_us*, que es idéntica a *delay_ms* pero en micro segundos.

A continuación se muestra el código fuente del PIC, para este ejemplo:

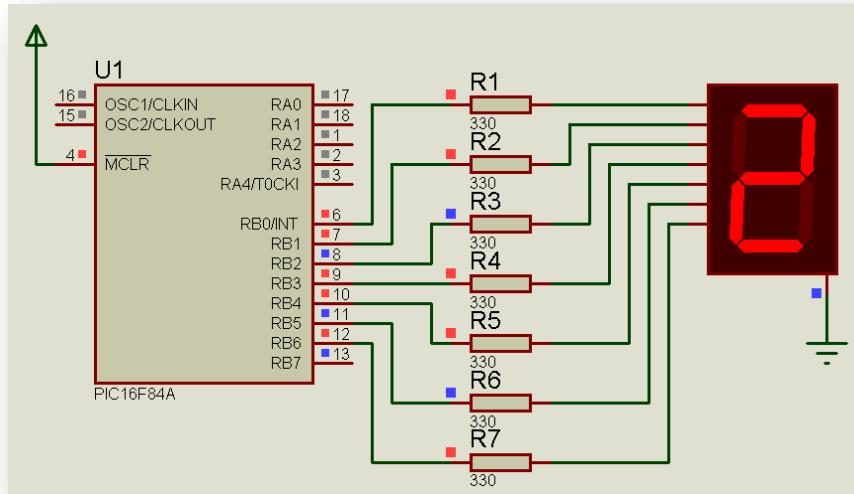
```
const unsigned short DIGITOS[] =  
{  
    0x3F, //Código del dígito 0  
    0x06, //Código del dígito 1  
    0x5B, //Código del dígito 2  
    0x4F, //Código del dígito 3  
    0x66, //Código del dígito 4  
    0x6D, //Código del dígito 5  
    0x7D, //Código del dígito 6  
    0x07, //Código del dígito 7  
    0x7F, //Código del dígito 8  
    0x6F, //Código del dígito 9  
};  
  
void main ( void )  
{  
    unsigned short CONTADOR=0;  
    TRISB = 0; // Configura el puerto B como salida  
    while( 1 )//Bucle infinito  
    {  
        PORTB = DIGITOS[CONTADOR]; //Se visualiza el dígito correspondiente al  
        //número guardado en la variable CONTADOR.  
        CONTADOR++; //Se incrementa el valor del conteo.  
    }  
}
```

```

        delay_ms(1000); //Se hace un retardo de 1 segundo.
    }
}

```

El paso siguiente es realizar la simulación en ISIS, para este fin se buscan los siguientes dispositivos: 16F84A, RES, y 7SEG-COM-CATHODE, para la conexión entre el PIC, y el display se deben usar resistencias de 330Ω . Posteriormente se arma el siguiente circuito:



Circuito 5-1

Al correr la simulación se debe ver un conteo en el display del 0 al 9, con una cadencia de un segundo entre dígito y dígito.

5.1.2 Control de displays 7 segmentos dinámicos

La implementación de displays dinámicos usa la misma teoría de un solo display, la visualización dinámica consiste en mostrar un solo dígito al mismo tiempo. Por ejemplo si se muestran cuatro dígitos se activa el display de las unidades, después se apagan y se activa el dígito de las decenas, posteriormente se apaga y se activa el dígito de las centenas, y por último se hace lo mismo con las unidades de mil. Este proceso se debe hacer con una velocidad de tal manera que engañe al ojo humano y así se verá como si todos los dígitos estuvieran activos. Este arreglo minimiza las conexiones eléctricas y el consumo de energía, dado que en realidad solo un dígito está activo para todo tiempo. A la vista del ojo humano los cambios deben ser de 25Hz o más, por lo tanto todos los dígitos deben verse durante un periodo igual al inverso de 25Hz, en este caso es 40m segundos. Para este ejemplo se usarán 4 displays por lo tanto el tiempo visible de cada dígito debe ser la cuarta parte del periodo es decir 10m segundos. La forma más eficiente de mostrar los números en el display es por medio de una función. Con 4 dígitos es posible visualizar un número de 0 a 9999, por esto el número puede ser consignado en una variable de tipo entera. Para ver cada dígito por separado se hace necesario calcular cada uno de los dígitos, por ejemplo para deducir las unidades de mil vasta con dividir el número en mil, y luego restar las unidades de mil del número completo, este proceso se repite hasta llegar a las unidades. La activación de los displays se debe hacer por medio de otro puerto para este ejemplo se hará por el puerto A. Para comprender este proceso observe y analice la siguiente función:

```

// Declaración de las constantes para el display.
const unsigned short DIGITOS[] =
{
    0x3F, //Código del dígito 0
    0x06, //Código del dígito 1
    0x5B, //Código del dígito 2
    0x4F, //Código del dígito 3
    0x66, //Código del dígito 4
    0x6D, //Código del dígito 5
    0x7D, //Código del dígito 6
    0x07, //Código del dígito 7
    0x7F, //Código del dígito 8
    0x6F, //Código del dígito 9
};

//Función para visualizar el display dinámico.
void VerDisplay( int Numero )
{
    unsigned short U; //Variable para guardar las unidades.
    unsigned short D; //Variable para guardar las decenas.
    unsigned short C; //Variable para guardar las centenas.
    unsigned short UM; //Variable para guardar las unidades de mil.
    UM = Numero/1000; //Cálculo de las unidades de mil.
    C = (Numero-UM*1000)/100; //Cálculo de las centenas.
    D = (Numero-UM*1000-C*100)/10; //Cálculo de las decenas.
    U = (Numero-UM*1000-C*100-D*10); //Cálculo de las unidades.

    PORTB = DIGITOS[U]; //Visualiza las unidades.
    PORTA.F0=1; //Activa en alto el primer display
    delay_ms(10); //Retardo de 10m segundos
    PORTA=0; //Desactiva todos los displays.

    PORTB = DIGITOS[D]; //Visualiza las decenas.
    PORTA.F1=1; //Activa en alto el segundo display
    delay_ms(10); //Retardo de 10m segundos
    PORTA=0; //Desactiva todos los displays.

    PORTB = DIGITOS[C]; //Visualiza las centenas.
    PORTA.F2=1; //Activa en alto el tercer display
    delay_ms(10); //Retardo de 10m segundos
    PORTA=0; //Desactiva todos los displays.

    PORTB = DIGITOS[UM]; //Visualiza las unidades de mil.
    PORTA.F3=1; //Activa en alto el cuarto display
    delay_ms(10); //Retardo de 10m segundos
    PORTA=0; //Desactiva todos los displays.
}

```

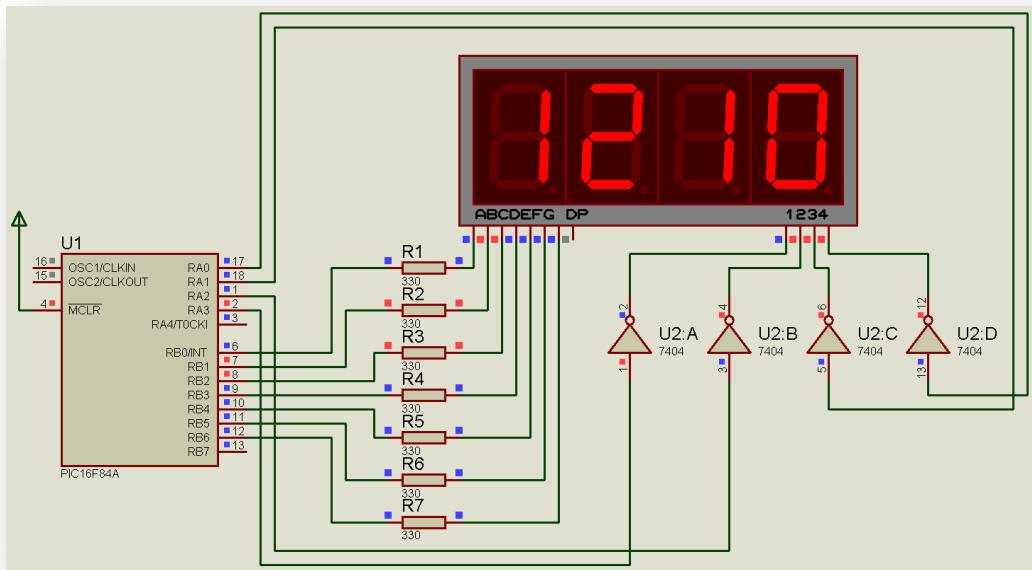
Por último se completa el código fuente con la función *main* de la siguiente forma:

```

void main ( void )
{
    unsigned short N=0; //Variable de conteo.
    int Numero=0;
    TRISB = 0; //Configura el puerto B como salida
    TRISA = 0; //Configura el puerto A como salida
    PORTA = 0; //Se desactiva todos los displays
    while( 1 )//Bucle infinito
    {
        //Se visualiza el valor de Número.
        VerDisplay( Numero ); //Esta función dura aproximadamente 40m segundos.
        //Se cuentan 12 incrementos en N para hacer un incremento
        //en Número aproximadamente cada 500m segundos.
        N++;
        if( N==12 )
        {
            N=0; //Se reinicia el conteo de N.
            Numero++; //Se incrementa el valor de Número.
            if( Numero==10000 ) //Se evalúa si Número vale 10000
                Numero=0;      //y se reinicia en 0 si es 10000.
        }
    }
}

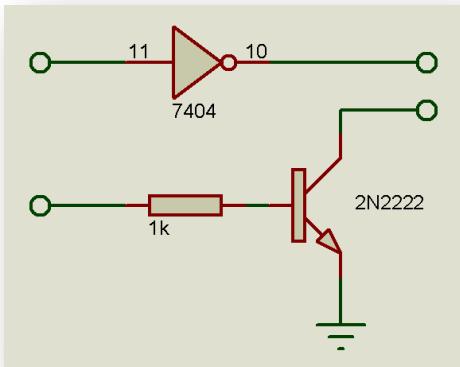
```

Al terminar y compilar el programa se realiza la simulación en ISIS, con los siguientes dispositivos: 16F84A, RES, 7SEG-MPX4-CC, 7404. Finalmente se construye el siguiente circuito:



Circuito 5-2

Para fines prácticos los inversores 7404 se pueden remplazar por un arreglo de transistores como se ven en la siguiente imagen:



Circuito 5-3

Cuando la simulación este en marcha el circuito debe mostrar un conteo de 0 a 9999 con una cadencia de 500m segundos entre cada incremento.

5.2 Display LCD de caracteres

Los display de caracteres LCD, son módulos prefabricados que contienen controladores incluidos. Estos displays cuentan con un bus de datos y un bus de control, para el manejo de estos dispositivos el compilador MikroC PRO, tiene una librería predefinida para el control de estos LCD. La apariencia física y la vista en ISIS de estos LCD es la que se puede apreciar en las siguientes gráficas:

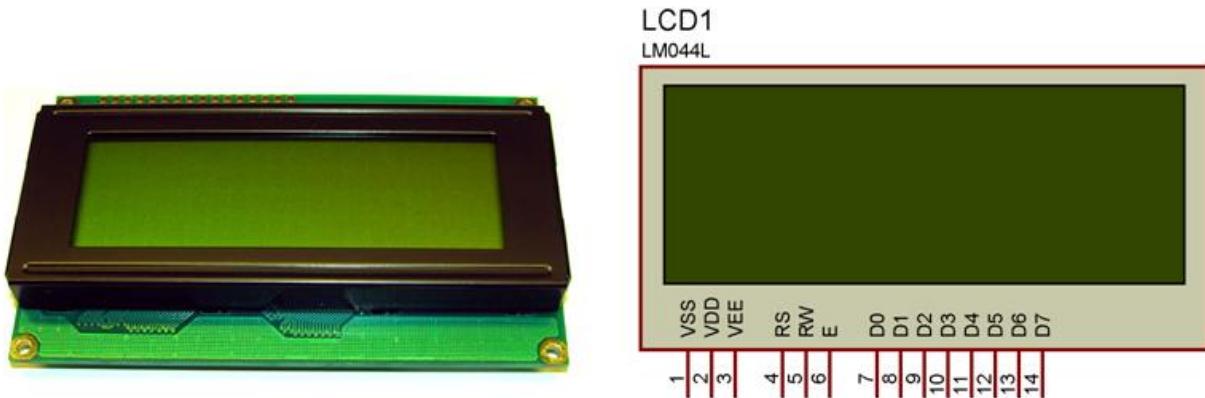


Figura 5-4

Los displays LCD, permiten graficar los caracteres contemplados en el código ASCII. Además del código ASCII, los displays LCD admiten graficar hasta 8 caracteres diseñados por el desarrollador, otra característica fundamental de los LCD, es la conexión del bus de datos, físicamente tienen 8 bits, pero es posible configurar las conexiones con solo 4 bits. La conexión de 8 bits implica una mayor cantidad de cables para su uso, pero la velocidad de trabajo es mayor, por consiguiente la conexión de 4 bits minimiza las conexiones pero disminuye la velocidad de trabajo. La librería predefinida en MikroC PRO, funciona con la configuración de 4 bits.

Para ver la librería predefinida para este dispositivo y otros que tiene MikroC PRO, se debe picar la paleta de herramientas ubicada en la parte derecha del programa, esta paleta se identifica con una pestáña que tiene el título: *Library Manager*. Al picar esta pestáña se despliega un menú que muestra las diferentes librerías habilitadas para el microcontrolador que se está trabajando. En este nuevo menú se identifica el ítem *Lcd*, posteriormente se puede picar cualquiera de las funciones que contiene la librería para ver la ayuda correspondiente. La apariencia visual de esta paleta se puede apreciar en la siguiente imagen:

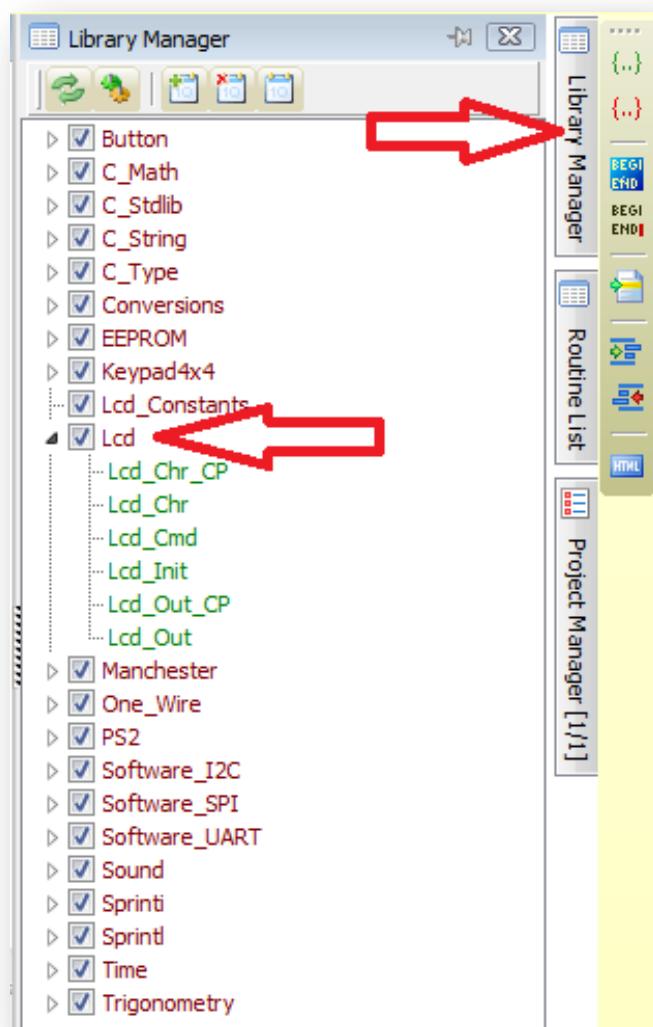


Figura 5-5

La implementación del display LCD, requiere del uso de instrucciones y comandos secuenciales para la configuración y desempeño del display, sin embargo la librería de MikroC PRO, minimiza este trabajo dado que está se encarga de hacer todas estas configuraciones, haciendo mucho más simple el trabajo del desarrollador. Como primer paso para el uso del LCD, se requiere definir los pines de conexión y la ejecución de la función predefinida de inicio del LCD; *Lcd_Init()*. La definición de los pines de conexión la asume el desarrollador de forma arbitraria según su criterio. Para cumplir este objetivo se usa la siguiente declaración de constantes:

```

//Pines de salida para el LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;

//Bits de configuración TRIS
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;

```

Para cambiar los pines solo se requiere cambiar el nombre de las declaraciones del ejemplo anterior. Como se puede ver en la declaración inicial solo se necesitan 6 pines para gobernar el LCD, con 4 bits de datos y 2 bits de control, el display cuenta con un tercer pin de control denotado como WR, este pin permite leer si el display está ocupado realizando alguna operación o permite escribir comandos y caracteres. Para fines de la librería este pin no es requerido, y simplemente se conecta a referencia. Por último se debe invocar la función de inicio del display dentro de la función *main* después de la configuración de puertos. Los pines de la LCD, no se deben configurar, está tarea ya la realiza la función: *Lcd_Init()*. La declaración de la función *main* debe quedar de la siguiente forma:

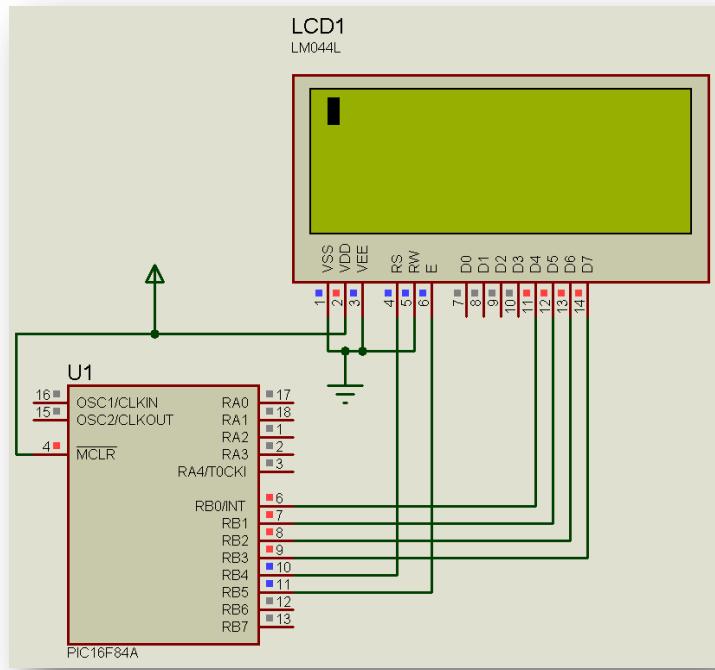
```

void main( void )
{
    Lcd_Init(); //Inicio del LCD.
    while(1) //Bucle infinito.
    {
    }
}

```

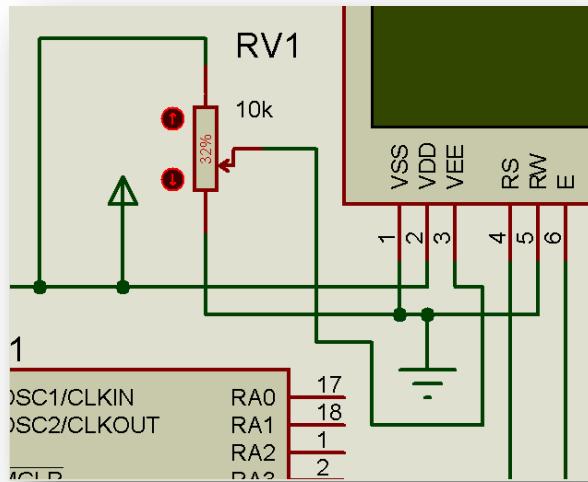
Terminada la edición del código anterior el LCD, se inicializa y debe quedar listo para empezar a graficar caracteres, posicionándose en la primera fila y primera columna, mostrando el cursor parpadeante.

Los displays de caracteres se fabrican en diversas formas y colores se pueden conseguir con pantallas de color verde, azul, y amarillo. Se fabrican con distribuciones de caracteres de forma matricial como 2 filas, 16 columnas estos se conocen como 2x16, y de la misma forma se pueden encontrar de 1x16, 2x16, 2x8, 2x20, 4x20, entre otros. Para los ejemplos empleados en este capítulo se usará el display de 4x20. Para iniciar la simulación del display LCD, se debe buscar el dispositivo LM044L, y el PIC 16F84A en el simulador ISIS. La referencia LM044L en ISIS corresponde a un LCD de 4x20. Finalmente se realizan las conexiones como se muestra en el siguiente circuito:



Circuito 5-4

El display LCD cuenta con un pin denominado vee, este pin funciona como controlador de contraste de la pantalla, pero para fines de la simulación no tiene efecto, este pin puede ser conectado a referencia para generar el mayor contraste, algunas pantallas de gran tamaño requieren de un voltaje negativo externo para el control de este contraste, en síntesis, cuanto más negativo sea el voltaje en Vee más oscuro será el contraste en la pantalla. Para fines prácticos el contraste se puede ajustar por medio de un potenciómetro como se ve en la siguiente circuito:



Circuito 5-5

El paso siguiente es imprimir en la pantalla información, para este objetivo se pueden usar cuatro funciones. Dos de estas funciones permiten imprimir caracteres, y las otras dos imprimen cadenas

de texto. Para imprimir los caracteres se puede hacer de dos maneras, la primera simplemente imprime los caracteres en el orden consecutivo que asume el display, y la siguiente función imprime los caracteres en la fila y columna que se designe por el desarrollador.

5.2.1 Funciones para imprimir caracteres

La primera función de impresión de caracteres es: *Lcd_Chr_Cp(char out_char)*; cuando está función es invocada imprime en la pantalla el carácter correspondiente al código ASCII, que está en el parámetro de entrada *out_char*. Con la impresión de un nuevo carácter la pantalla LCD, incrementa automáticamente el cursor en una posición. Para contextualizar el funcionamiento de esta función observe y analice el siguiente ejemplo:

```
void main( void )
{
    Lcd_Init(); //Inicio del LCD.
    Lcd_Chr_Cp('H'); //Estás funciones imprimen letra a letra la palabra "Hola".
    Lcd_Chr_Cp('o');
    Lcd_Chr_Cp('l');
    Lcd_Chr_Cp('a');
    while(1) //Bucle infinito.
    {
    }
}
```

Después de correr la simulación se debe observar lo siguiente en la pantalla del LCD:

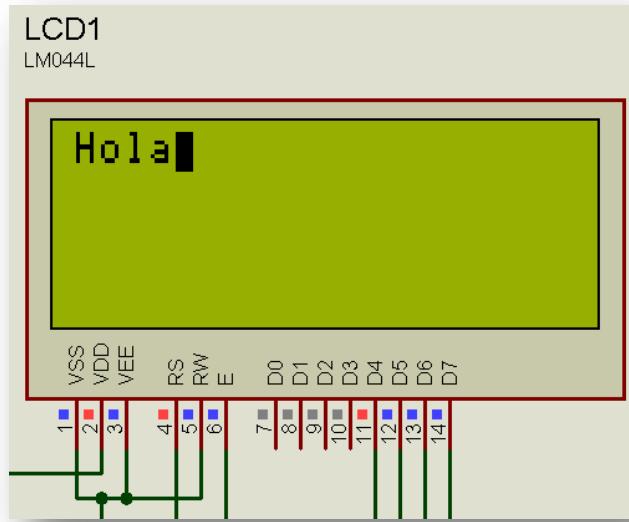


Figura 5-6

Para realizar la impresión de caracteres por medio de una coordenada fila, columna, se implementa la función: *Lcd_Chr(char row, char column, char out_char)*; está función imprime el carácter *out_char*, en la columna *column* y en la fila *row*. En el siguiente ejemplo se puede ver cómo usar esta función:

```

void main( void )
{
    Lcd_Init(); //Inicio del LCD.
    Lcd_Ch(chr('H'); //Estás funciones imprimen letra a letra la palabra "Hola".
    Lcd_Ch(chr('o');
    Lcd_Ch(chr('l');
    Lcd_Ch(chr('a');
    Lcd_Ch( 1, 6, '1'); //Imprime el carácter 1, en la fila 1, columna 6
    Lcd_Ch( 2, 7, '2'); //Imprime el carácter 2, en la fila 2, columna 7
    Lcd_Ch( 3, 8, '3'); //Imprime el carácter 3, en la fila 3, columna 8
    Lcd_Ch( 4, 9, '4'); //Imprime el carácter 4, en la fila 4, columna 9
    while(1) //Bucle infinito.
    {
    }
}

```

Después de ejecutar la simulación se debe tener la siguiente vista en la pantalla del LCD:

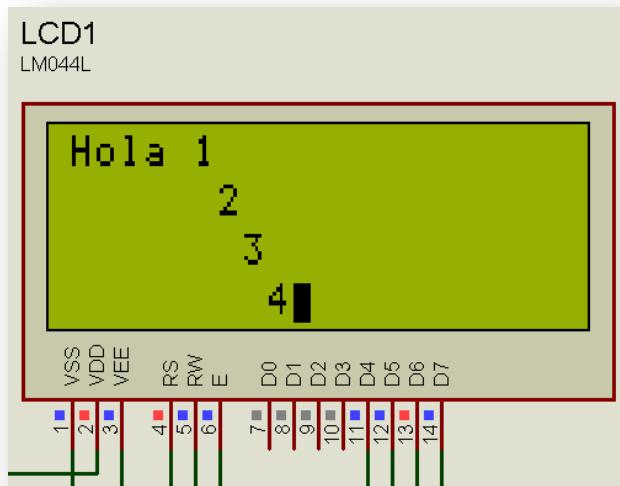


Figura 5-7

5.2.2 Funciones para imprimir cadenas de texto

El uso de cadenas de texto es similar a las dos funciones anteriores, para esto se pueden imprimir cadenas de texto en el punto donde está el flujo de la impresión del LCD, o en un punto arbitrario por medio de las coordenadas de filas y columnas. Para hacer la impresión de una cadena de texto en el punto de impresión se utiliza la siguiente función: *Lcd_Out_Cp(char *text);* está cuenta con un único parámetro de entrada que es un apuntador a la cadena de caracteres en este caso la función la denota como *text*. Para el uso de esta función observe y analice el siguiente ejemplo:

```

void main( void )
{
    Lcd_Init(); //Inicio del LCD.
    Lcd_Out_Cp("Hola Mundo...");
    while(1) //Bucle infinito.
}

```

```

{
}
}
```

La implementación de esta función puede hacerse con cadenas de caracteres constantes o variables, las cadenas constantes se denotan por medio de las dobles comillas al inicio y al final del texto por ejemplo: “*Hola Mundo...*”, el formato variable se declara con la forma: **char Text[20]=“Hola Mundo...”;**. Después de correr la simulación se debe ver una vista como la que aparece en la siguiente figura:



Figura 5-8

Para imprimir una cadena de caracteres con una coordenada como punto de inicio se implementa la función: **Lcd_Out(char row, char column, char *text);**. Esta función trabaja de forma similar a la anterior, con la diferencia que se incluyen los datos de *row*, y *column*, que hacen referencia a las filas y las columnas respectivamente. Para comprender de forma clara el funcionamiento de esta función observe y analice el siguiente ejemplo:

```

void main( void )
{
    Lcd_Init(); //Inicio del LCD.
    Lcd_Out( 1, 1, "Fila 1, Columna 1" );
    Lcd_Out( 2, 2, "Fila 2, Columna 2" );
    while(1) //Bucle infinito.
    {
    }
}
```

Después de compilar y simular este ejemplo se debe observar una vista en la pantalla del LCD como la que se puede apreciar en la siguiente figura:

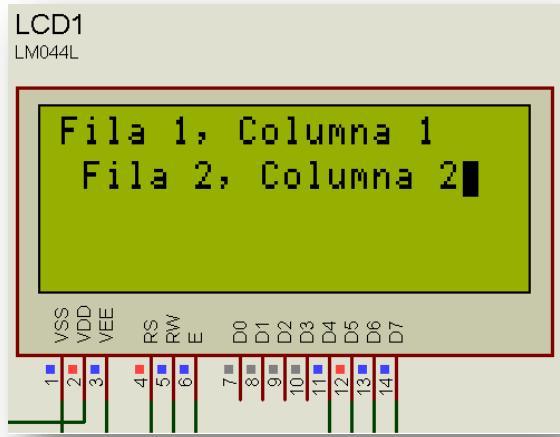


Figura 5-9

5.2.3 Impresión de valores numéricos

La impresión de valores numéricos es de vital importancia en muchos de los desarrollos. Por ejemplo cuando se desea visualizar, el estado de una variable, un sensor como: temperatura, humedad, presión, o cualquier otro que se esté manipulando. Para lograr este objetivo se puede recurrir a funciones de conversión predefinidas por el compilador. Para esto se puede utilizar la librería: *Conversions*, está librería cuenta con funciones que hacen la conversión de un valor numérico a una cadena de caracteres. Si se requiere visualizar un valor entero se usa la función: *IntToStr(int input, char *output)*; esta función tiene dos parámetros de entrada que son: *input*, que es un valor entero a visualizar, y *output*, que es el apuntador a una cadena de caracteres donde se quiere escribir el forma de texto el valor que contiene el valor *input*. Para entender este tipo de conversión observe y analice el siguiente ejemplo:

```
void main( void )
{
    int ENTERO=123; //Declaración de una variable entera con valor inicial 123.
    char Text[20]; //Cadena de caracteres para impresión de datos.
    Lcd_Init(); //Inicio del LCD.
    IntToStr( ENTERO,Text ); //Función que hace la conversión.
    Lcd_Out_Cp(Text); //Impresión del texto en la pantalla LCD.
    while(1) //Bucle infinito.
    {
    }
}
```

La impresión de los números enteros en la cadena de texto con esta función siempre asume un campo fijo de 7 caracteres, es decir que si el número tiene menos de 7 dígitos el resto del texto se completa con espacios vacíos. Después de compilar y simular el programa se tendrá una vista en ISIS, como la que se puede ver en la siguiente figura:

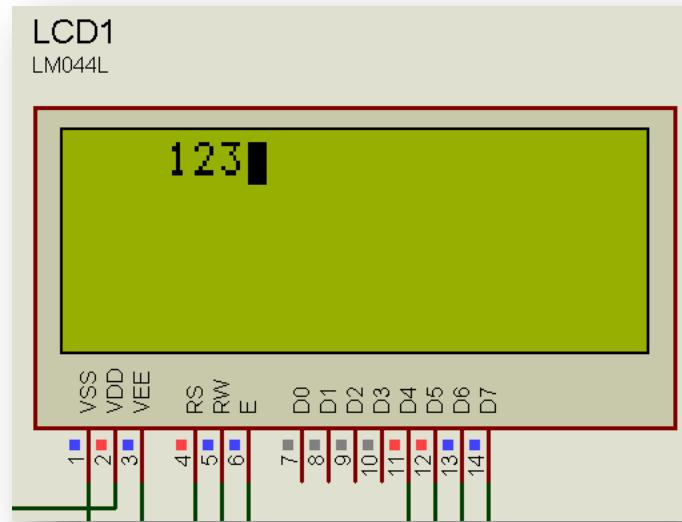


Figura 5-10

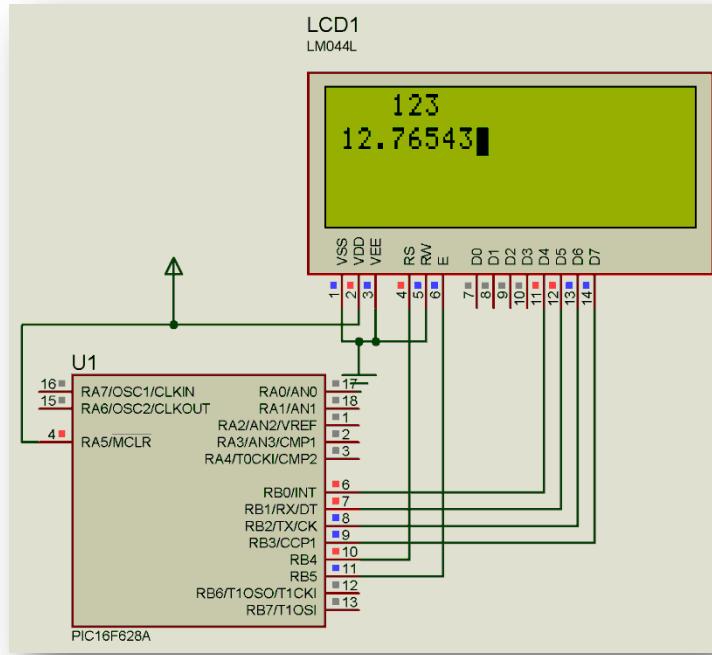
La impresión de números con punto decimal se puede hacer con la función: *FloatToStr(float fn, char *str)*; La filosofía de funcionamiento de esta función es idéntica a la anterior que hace las conversiones de números enteros. Para realizar el siguiente ejemplo se debe cambiar la referencia del microcontrolador, esto se debe a que la capacidad de memoria del PIC 16F84A, no es suficiente para los siguientes ejercicios. A partir de este punto se usara el PIC 16F628A, que tiene la misma distribución de pines del 16F84A, pero cuenta con una capacidad mayor de memoria y con módulos integrados superiores como la USART. Para contextualizar su funcionamiento observe el siguiente ejemplo:

```

void main( void )
{
    int ENTERO=123; //Declaración de una variable entera con valor inicial 123.
    float DECIMAL=12.76543; //Declaración de una variable con punto decimal
    //inicializada en 12,76543.
    char Text[20]; //Cadena de caracteres para impresión de datos.
    Lcd_Init(); //Inicio del LCD.
    IntToStr( ENTERO,Text ); //Función que hace la conversión entera.
    Lcd_Out(1,1,Text); //Impresión del texto en la pantalla LCD.
    FloatToStr( DECIMAL,Text ); //Función que hace la conversión decimal.
    Lcd_Out(2,1,Text); //Impresión del texto en la pantalla LCD.
    while(1) //Bucle infinito.
    {
    }
}

```

Terminada la simulación se de tener una vista como la que se aprecia en la figura siguiente:



Circuito 5-6

El mismo proceso se puede seguir para otros tipos de variables como son: *short* con la función: `ShortToStr(short input, char *output);`

Las variables *long* con la función:

`LongToStr(long input, char *output);`

Las variables *unsigned short* con la función:

`ByteToStr(unsigned short input, char *output);`

Las variables *unsigned long* con la función:

`LongWordToStr(unsigned long input, char *output);`

Las variables *unsigned int* con la función:

`WordToStr(unsigned int input, char *output);`

5.2.4 Creación de caracteres propios

Las pantallas de caracteres LCD, cuentan con la capacidad de almacenar hasta 8 caracteres diseñados por el desarrollador. Los valores ASCII, para los caracteres diseñados por el usuario corresponde a los valores numéricos del 0 al 7. Por ejemplo, si se desea imprimir el primer carácter diseñado por el usuario se hace con las funciones de impresión de caracteres: `Lcd_Chр_Cp(0);` o `Lcd_Chр(1,1,0);`. La creación de caracteres propios creados por el usuario se guardan en un campo de memoria RAM, de la pantalla LCD, por esta razón deben ser reprogramados cada vez que se inicializa la pantalla LCD. Los caracteres son un conjunto de valores binarios que forman un mapa de bits, y dan como resultado la imagen en la pantalla con una resolución de 5 x 7 píxeles. La edición y programación de estos datos resulta dispendioso y cuidadoso con las órdenes de bajo nivel que se deben dar a la pantalla LCD. Sin embargo el compilador MikroC PRO, cuenta dentro de sus herramientas con un editor para los caracteres diseñados por el desarrollador. En este editor de forma simple se edita el mapa de bits del carácter y este genera la fracción de código en lenguaje C, requerida para programar el carácter en la pantalla. En conclusión se edita el carácter y posteriormente se pega el código en la fuente

del proyecto que se está desarrollando. Para usar este editor se busca el ítem *Tools* dentro del menú superior del programa MikroC PRO, y dentro de este se pica el submenú: *LCD Custom character*, esta acción despliega la ventana del editor que tiene la apariencia que se puede ver en la siguiente figura:

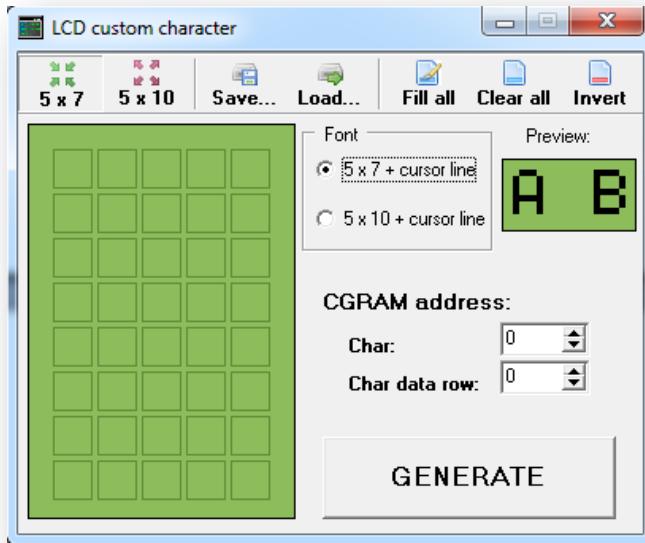


Figura 5-11

En este editor el usuario pica los cuadros de la cuadricula de color verde y al terminar la edición del carácter, se selecciona el valor ASCII que se asignará al carácter por medio de la casilla *Char*: este valor debe ser un número entre 0 y 7. Terminado este proceso se pulsa el botón *GENERATE*, y con esta acción el editor mostrará la fracción de código para usar en el programa que se está desarrollando. Las siguientes gráficas muestran la vista del editor cuando se hace un carácter en la dirección ASCII 0:

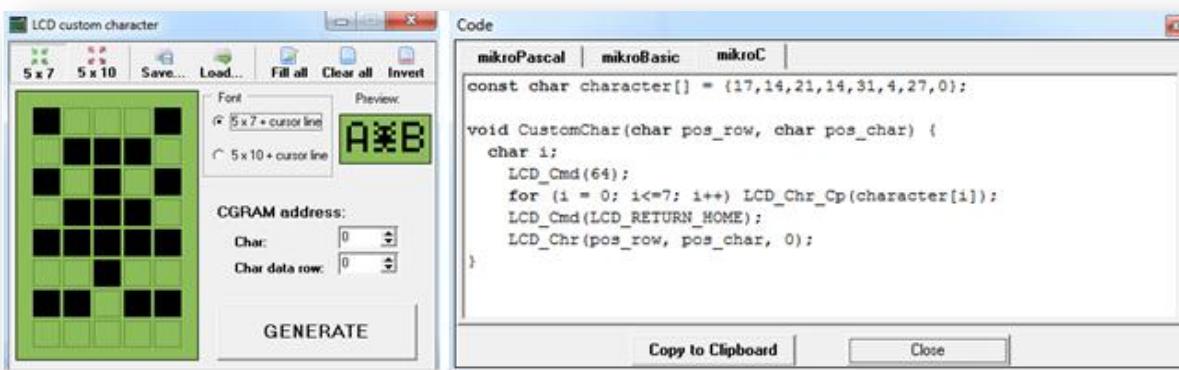


Figura 5-12

Como último paso se debe integrar la información entregada por el editor y el programa del desarrollador, para este fin observe el siguiente ejemplo:

```
//Constantes creadas por el editor, está contiene el mapa de bits del carácter.  
const char character[] = {17,14,21,14,31,4,27,0};
```

```
//Pines de salida para el LCD  
sbit LCD_RS at RB4_bit;  
sbit LCD_EN at RB5_bit;  
sbit LCD_D7 at RB3_bit;  
sbit LCD_D6 at RB2_bit;  
sbit LCD_D5 at RB1_bit;  
sbit LCD_D4 at RB0_bit;
```

```
//Bits de configuración TRIS  
sbit LCD_RS_Direction at TRISB4_bit;  
sbit LCD_EN_Direction at TRISB5_bit;  
sbit LCD_D7_Direction at TRISB3_bit;  
sbit LCD_D6_Direction at TRISB2_bit;  
sbit LCD_D5_Direction at TRISB1_bit;  
sbit LCD_D4_Direction at TRISB0_bit;
```

```
//Función para guardar en la memoria RAM del LCD  
//el carácter del editor.  
void CustomChar(char pos_row, char pos_char)  
{  
    char i;  
    LCD_Cmd(64); //Dirección del carácter 0.  
    //Bucle for para guardar el mapa de bits.  
    for (i = 0; i<=7; i++) LCD_Ch(chr(character[i]);  
    //Para fines prácticos esta función puede ser ignorada.  
    //LCD_Cmd(LCD_RETURN_HOME);  
    //Impresión del código ASCII 0.  
    LCD_Ch(pos_row, pos_char, 0);  
}
```

```
void main( void )  
{  
    Lcd_Init(); //Inicio del LCD.  
    CustomChar(1, 1); //Impresión del Carácter 0.  
    while(1) //Bucle infinito.  
    {  
    }  
}
```

Transcurrida la edición compilación y simulación se debe tener la siguiente vista en la simulación de ISIS:

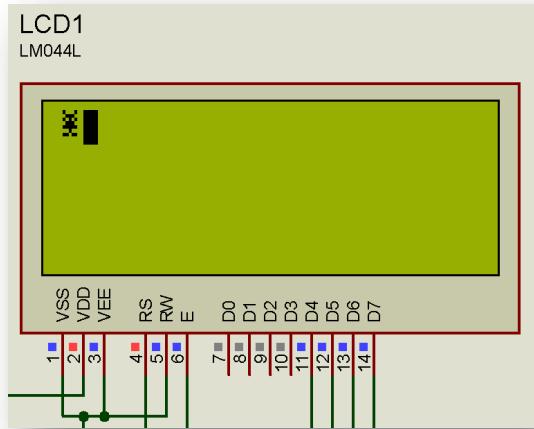


Figura 5-13

Para optimizar el uso de la generación de caracteres arbitrarios el desarrollador puede simplemente grabar la memoria al principio de las configuraciones y después hacer la impresión de los caracteres, editados del 0, al 7. Para esto se debe tener presente el arreglo constante de datos que corresponden a los mapas de bits, la dirección inicial que muestra el editor, e invocar las funciones de impresión de caracteres cuando sea necesario. Para formalizar esta idea observe y analice el siguiente ejemplo que imprime en la pantalla la imagen de 4 caracteres con forma de extraterrestres:

```
//Declaración de constantes, representan los mapas de bits,
//de un marcianito animado. Estos código son el resultado del editor de caracteres de
//MikroC PRO.
const char Marciano1[] = {17,14,21,14,31,4,27,0}; //Dirección inicial 64.
const char Marciano2[] = {17,14,21,14,31,4,10,0}; //Dirección inicial 72.
const char Marciano3[] = {17,14,21,14,31,21,10,0}; //Dirección inicial 80.
const char Nave[] = {24,28,30,31,30,28,24,0}; //Dirección inicial 88.

//Pines de salida para el LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;

//Bits de configuración TRIS
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;
```

```

//Declaración de función para guardar en RAM los caracteres.
//Está función es diseñada por el desarrollador.
void Lcd_RAM_Car( char IntDir, const char *Mapa )
{
    char i;
    LCD_Cmd(IntDir); //Dirección inicial.
    //Bucle para guardar el mapa en la RAM del LCD.
    for( i=0; i<8; i++ ) LCD_ChR_Cp( Mapa[i] );
}

void main( void )
{
    Lcd_Init(); //Inicio del LCD.
    Lcd_RAM_Car( 64, Marciano1 );//Se guarda el character 1
    Lcd_RAM_Car( 72, Marciano2 );//Se guarda el character 2
    Lcd_RAM_Car( 80, Marciano3 );//Se guarda el character 3
    Lcd_RAM_Car( 88, Nave );//Se guarda el character 4
    LCD_ChR(1, 1, 0);//Se imprime el carácter Marciano 1.
    LCD_ChR(1, 2, 1); //Se imprime el carácter Marciano 2.
    LCD_ChR(1, 3, 2); //Se imprime el carácter Marciano 3.
    LCD_ChR(1, 4, 3); //Se imprime el carácter Nave.

    while(1) //Bucle infinito.
    {
    }
}

```

Terminada la compilación y corrida la simulación se debe tener una vista como la siguiente en el simulador ISIS:

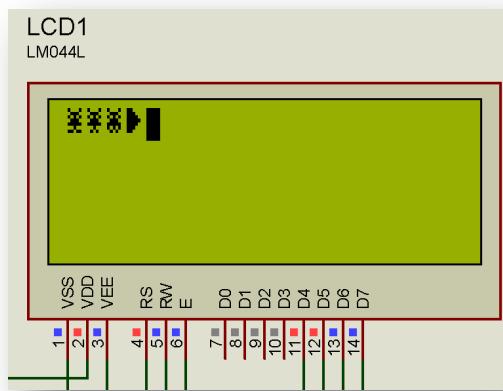


Figura 5-14

La impresión de información en la pantalla LCD, implica en algunos casos la necesidad de cambiar las características del cursor. Para esto, es posible sustituir el cursor por uno que no es sólido o desactivar el efecto parpadeante del mismo, incluso se puede simplemente desactivar el cursor. Con el fin de realizar cambios de características en el LCD, se usa la función:

Lcd_Cmd(char out_char);, en la cual se ingresan los comandos de configuración, por medio del parámetro: *out_char*. Para colocar un cursor no sólido, o raya al piso (_), se usa el siguiente comando: *Lcd_Cmd(_LCD_UNDERLINE_ON);*, para activar el cursor sólido parpadeante se implementa el comando: *Lcd_Cmd(_LCD_BLINK_CURSOR_ON);*; Para desactivar el cursor y hacerlo no visible se puede invocar el comando: *Lcd_Cmd(_LCD_CURSOR_OFF);*.

5.3 Display LCD gráficos

Los display gráficos, son módulos similares a los LCD de caracteres, su particularidad principal es que permiten graficar un mapa de bits de mayor resolución. Estos se pueden adquirir comercialmente en tamaños como: 128x64, 128x128, 240x320, píxeles entre otros. Estos dispositivos de manera similar a los displays de caracteres requieren un protocolo de configuración, y para ello cuentan con un bus de datos y un bus de control. Los LCD gráficos, permiten imprimir fragmentos de líneas, horizontales o verticales armando en conjunto una imagen total. Para fines de estudio y análisis en este capítulo se demostrará el control sobre un LCD gráfico de 128x64 píxeles. La apariencia física y la vista desde el simulador ISIS, se puede apreciar en las siguientes imágenes:

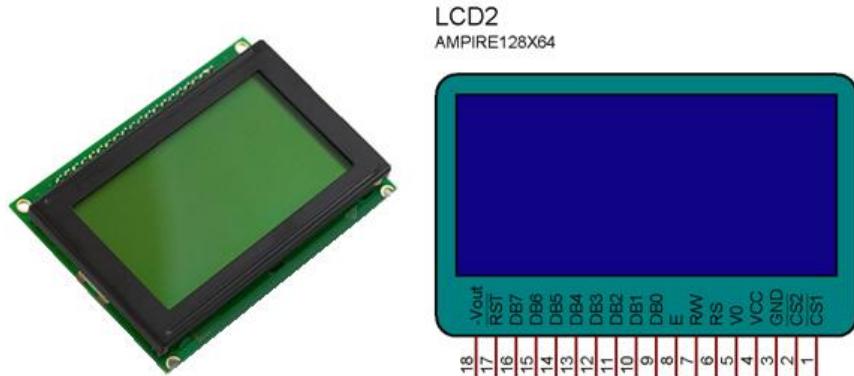


Figura 5-15

Estos dispositivos cuentan con 20 pines de los cuales 8 son el bus de datos, 6 son de control, 2 son de polarización, y 2 son para ajustar el contraste. De la misma forma que el LCD de caracteres para fines de simulación los pines de contraste son ignorados. Sin embargo para fines prácticos los pines de contraste se deben instalar con un potenciómetro como se muestra en la siguiente figura:

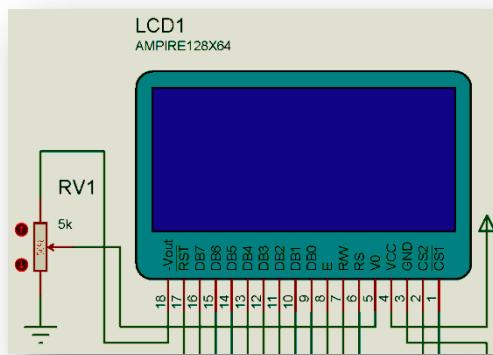


Figura 5-16

Para los ejemplos tratados en este capítulo, se creará un proyecto nuevo en MikroC PRO, con el microcontrolador 16F877A. Este dispositivo es un PIC de 40 pines y 8K bytes de memoria de programa, la cual es suficiente para realizar las primeras prácticas con el LCD gráfico. Nuevamente las bondades del compilador MikroC PRO, brindan una librería especializada en el uso y control de este display gráfico. Para esto se cuenta con la librería *Glcd*, ubicada en la paleta de librerías. Como primera medida para configurar el módulo gráfico, se definen de manera arbitraria los pines por los cuales se conectarán el display. Para esto se hace una declaración de bits constantes similar a la que se hace con el LCD de caracteres. En el siguiente ejemplo se puede apreciar esta declaración:

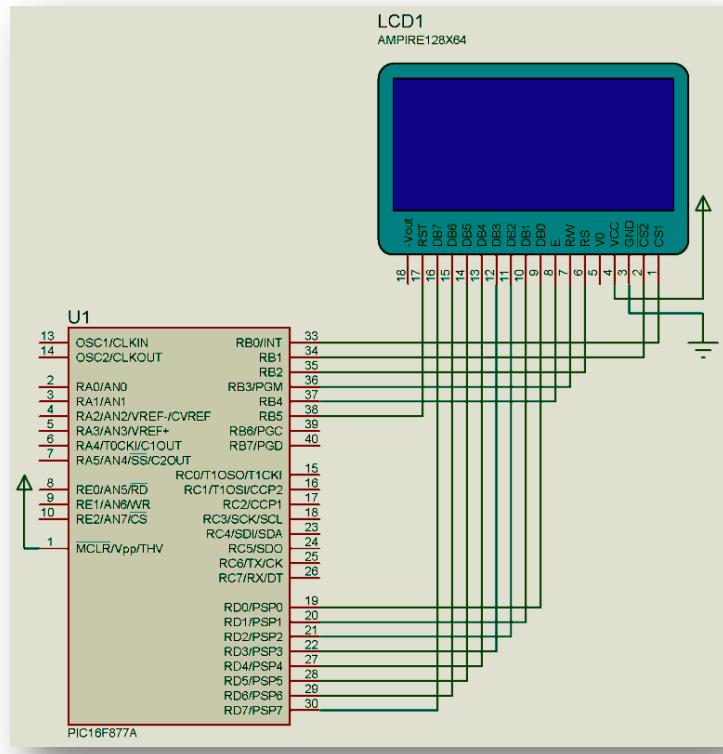
```
// Declaración del puerto con el bus de datos.
char GLCD_DataPort at PORTD;
//Declaración de los pines de control.
sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS at RB2_bit;
sbit GLCD_RW at RB3_bit;
sbit GLCD_EN at RB4_bit;
sbit GLCD_RST at RB5_bit;
//Declaración de los registros de TRIS de control.
sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction at TRISB2_bit;
sbit GLCD_RW_Direction at TRISB3_bit;
sbit GLCD_EN_Direction at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
```

Para inicializar el display se usa la función: *Glcd_Init()*; después de las configuraciones del PIC dentro de la función *main*. El paso siguiente es cargar alguna gráfica, para este ejemplo se dibujará en el display la siguiente imagen:



Figura 5-17

La configuración antes citada se debe respetar en las conexiones físicas con el display gráfico. En la creación del circuito electrónico en el simulador ISIS, se implementan los dispositivos: PIC 16F877A, y el módulo gráfico: AMPIRE 128x64. Posteriormente se realizan las conexiones respectivas como se puede apreciar en la siguiente gráfica:



Circuito 5-7

Las imágenes que se desean cargar deben ser editadas con anterioridad en algún editor de imágenes como el *Paint* de *Windows*. Estos archivos deben respetar las dimensiones de la pantalla gráfica en este caso de 128x64 píxeles. Por último el archivo debe ser guardado en formato bmp, y color monocromático. Para el ingreso del mapa de bits, se puede usar un herramienta incluida en el MikroC PRO, denominada: *GLCD Bitmap Editor*, este editor se ejecuta buscando en el menú principal dentro del ítem *Tools*, y su submenuí *GLCD Bitmap Editor*. Al correr esta aplicación se debe tener una vista como la siguiente:

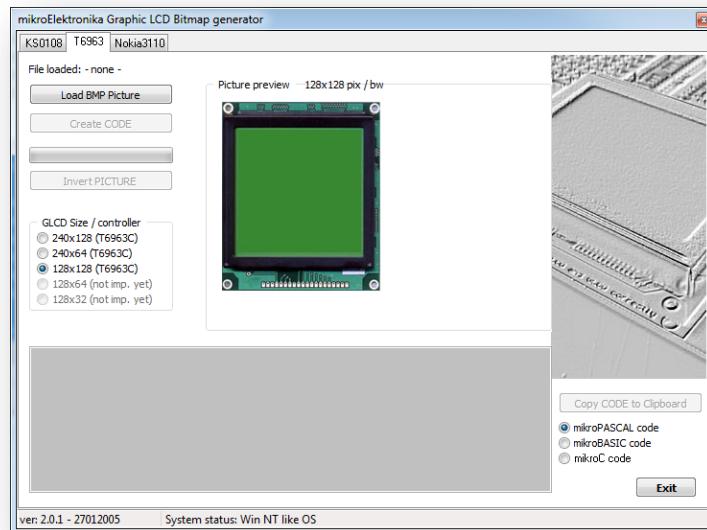


Figura 5-18

En esta aplicación se pica la pestaña: *KS0108*, y se asegura la selección circular de *128x64* (*KS0108*). Posteriormente se pulsa el botón *Load BMP Picture*, y se carga el archivo de imagen previamente diseñado en el editor de imagen. Seguidamente se selecciona el círculo con la opción *mikroC Code*, y se pulsa el botón: *Create CODE*. Esta acción genera el código en lenguaje C, para agregar al código de programa en el inicio en la parte de la declaración de constantes. Esta declaración es un arreglo de tipo: *const unsigned char*. Para el caso práctico de este ejemplo el resultado es el siguiente:

```
// -----
// GLCD Picture name: Imagen.bmp
// GLCD Model: KS0108 128x64
// -----
unsigned char const Imagen_bmp[1024] = {
255, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
193, 241, 193, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 65, 65, 65,
65, 65, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
33, 33, 33, 33, 33, 65, 65, 65, 65, 65, 65, 129, 129, 129,
129, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
255, 0, 0, 0, 0, 16, 48, 112, 112, 240, 240, 240, 240, 248, 254,
255, 255, 255, 254, 248, 240, 240, 240, 240, 112, 112, 48, 16, 0, 0,
0, 0, 0, 0, 128, 64, 64, 32, 32, 16, 16, 8, 8, 4,
4, 2, 2, 2, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 2, 2, 2, 4, 4, 8, 8, 8, 16, 16, 32,
64, 64, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255,
255, 0, 0, 0, 0, 0, 0, 0, 0, 129, 241, 255, 255, 127, 127,
63, 31, 63, 127, 127, 255, 255, 241, 129, 0, 0, 0, 0, 0, 192,
32, 16, 12, 4, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 192, 192, 192, 192, 192, 192, 128, 128, 0, 0, 0, 192, 192,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 192, 64, 192, 128, 192, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 2, 2, 12, 16, 32, 192, 0, 0, 0, 0, 0, 255,
255, 0, 0, 0, 0, 0, 0, 0, 3, 1, 0, 240, 240, 240,
240, 240, 240, 240, 0, 0, 1, 3, 0, 0, 0, 0, 0, 252, 3, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 255, 255, 0, 0, 0, 0, 0, 129, 255, 126, 0, 0, 254, 254,
0, 0, 156, 190, 54, 118, 102, 238, 204, 0, 0, 248, 252, 182, 54, 54,
188, 184, 0, 0, 254, 254, 12, 6, 6, 6, 254, 252, 0, 0, 248, 252,
142, 6, 6, 142, 252, 248, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 7, 248, 0, 0, 0, 255,
255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
255, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 1, 14, 16,
96, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 3, 3, 3, 3, 3, 3, 3, 1, 1, 0, 0, 0, 3, 3,
```

```
0, 0, 1, 3, 3, 3, 3, 1, 0, 0, 0, 1, 3, 3, 3,
1, 0, 0, 0, 3, 3, 0, 0, 0, 0, 3, 3, 0, 0, 0, 1,
3, 3, 3, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,128, 64, 48, 14, 1, 0, 0, 0,255,
255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 48, 240, 240, 255, 255, 255,
255, 255, 255, 255, 255, 240, 240, 48, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 2, 4, 8, 8, 16, 32, 32, 64, 64, 128, 128, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 32,
32, 16, 8, 8, 4, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255,
255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 31, 255, 255,
255, 255, 255, 255, 31, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
30, 226, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 128, 64,
64, 64, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 16, 16, 16, 16, 16, 16, 16, 8,
8, 8, 8, 4, 4, 4, 2, 2, 2, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255,
255, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 131,
159, 159, 131, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
128, 135, 152, 136, 136, 136, 132, 132, 130, 130, 129, 129, 129, 129, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 255
};
```

La declaración anterior contiene codificada la información del mapa de bits cargado en el editor. La vista final del generador debe ser la siguiente:

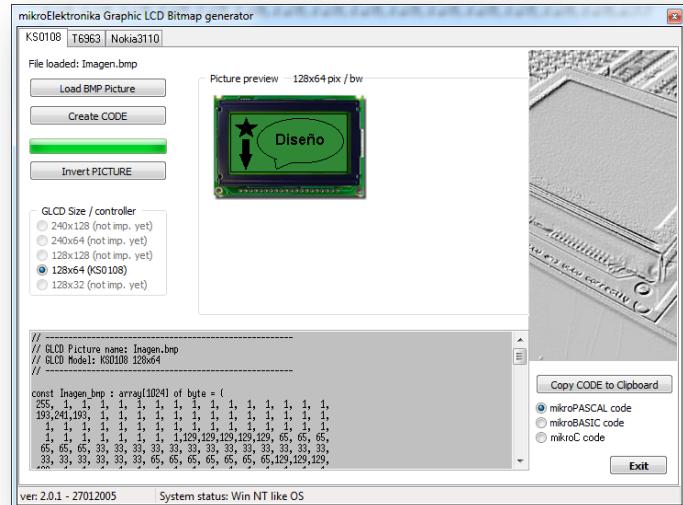


Figura 5-19

Después de editar, inicializar, y anexar el mapa de bits en el código fuente se tiene la siguiente fuente total:

```
// Declaración del puerto con el bus de datos.
char GLCD_DataPort at PORTD;

//Declaración de los pines de control.
sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS at RB2_bit;
sbit GLCD_RW at RB3_bit;
sbit GLCD_EN at RB4_bit;
sbit GLCD_RST at RB5_bit;

//Declaración de los registros de TRIS de control.
sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction at TRISB2_bit;
sbit GLCD_RW_Direction at TRISB3_bit;
sbit GLCD_EN_Direction at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;

unsigned char const Imagen_bmp[1024] = {
    255, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    193, 241, 193, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    65, 65, 65, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
    33, 33, 33, 33, 33, 33, 33, 65, 65, 65, 65, 65, 65, 65, 129, 129, 129,
    129, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    255, 0, 0, 0, 0, 16, 48, 112, 112, 240, 240, 240, 240, 240, 248, 254,
    255, 255, 255, 254, 248, 240, 240, 240, 240, 240, 112, 112, 48, 16, 0, 0,
    0, 0, 0, 0, 0, 128, 64, 64, 32, 32, 16, 16, 8, 8, 4,
    4, 2, 2, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 2, 2, 2, 4, 4, 8, 8, 8, 16, 16, 32,
    64, 64, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255,
    255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 129, 241, 255, 255, 127, 127,
    63, 31, 63, 127, 127, 255, 255, 241, 129, 0, 0, 0, 0, 0, 0, 192,
    32, 16, 12, 4, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 192, 192, 192, 192, 192, 192, 192, 128, 128, 0, 0, 0, 192, 192,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 192, 64, 192, 128, 192, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, 2, 2, 12, 16, 32, 192, 0, 0, 0, 0, 0, 255,
    255, 0, 0, 0, 0, 0, 0, 0, 0, 3, 1, 0, 240, 240, 240,
    240, 240, 240, 240, 0, 0, 1, 3, 0, 0, 0, 0, 252, 3, 0,
```


La función: `Glcd_Fill(0);`, rellena el contenido de la pantalla con datos binarios iguales a 0, este proceso equivale a borrar toda la pantalla del LCD. Finalizada la creación del código, y compilado el proyecto la simulación en ISIS debe mostrar una vista como la siguiente:

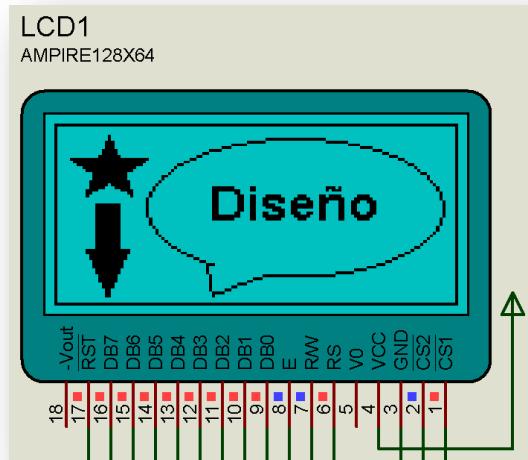


Figura 5-20

La carga de mapas de bits, solo está limitada por el tamaño de las mismas, es decir que se pueden anexar tantas imágenes como la memoria de programa del PIC, soporte. Si se requiere anexar más imágenes y la memoria ya no es suficiente, se debe cambiar el microcontrolador por uno que cuente con mas campo de memoria de programa, por ejemplo el PIC 18F452, este tiene la misma distribución de pines del PIC 16F877A, pero cuenta con una memoria superior y puede procesar su unidad hasta 40MHz, el PIC 16F877A se puede desempeñar hasta 20MHz.

Además de cargar mapas de bits, la librería del MikroC PRO, permite dibujar líneas, círculos, y rectángulos. Por otra parte permite cargar una o más fuentes de texto previamente creadas para la impresión de datos. Lamentablemente, el compilador MikroC PRO, no cuenta con un editor de fuentes para este fin, pero es posible diseñarlas o incluso cargarlas de las fuentes convencionales del sistema operativo Windows. Para lograr este cometido se requiere un editor de fuentes para GLCD, como el software de la siguiente imagen:

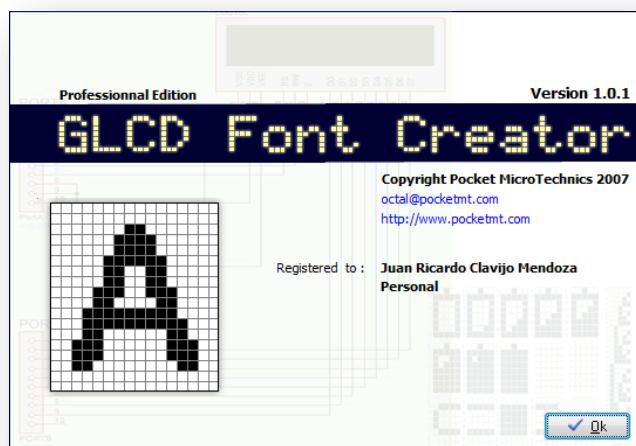


Figura 5-21

Sin embargo la librería del MikroC PRO, contiene las siguientes fuentes de texto predeterminadas:

- *System3x5, definida como: Font_Glcd_System3x5*
- *FontSystem5x7_v2, definida como: Font_Glcd_System5x7*
- *font5x7, definida como: Font_Glcd_5x7*
- *Character8x7, definida como: Font_Glcd_Character8x7*

Estás fuentes, en conjunto con las capacidades de graficación y la importación de mapas de bits hacen de los módulos LCD gráficos, una poderosa herramienta para dar una presentación profesional a los proyectos requeridos por el desarrollador.

Para el siguiente ejemplo se usarán las opciones de graficación por medio de puntos, líneas, círculos, rectángulos, e impresión de textos.

Para hacer la impresión de puntos se implementa la función:

Glcd_Dot(unsigned short x_pos, unsigned short y_pos, unsigned short color);, está función imprime píxeles, con el color especificado en los parámetros de la función. El parámetro color puede valer: 0 para hacer un píxel de color blanco, 1 para hacer un píxel de color negro, o 2 para invertir el color sobre el píxel dibujado.

Para dibujar líneas se utiliza la función:

Glcd_Line(int x_start, int y_start, int x_end, int y_end, unsigned short color);, está función contempla como parámetros de entrada las coordenadas de inicio y final así como el parámetro color este puede valer: 0 para hacer la línea de color blanco, 1 para hacer la línea de color negro, o 2 para invertir el color sobre el cual se dibuja la línea.

La impresión de rectángulos se logra con la función:

Glcd_Rectangle(unsigned short x_upper_left, unsigned short y_upper_left, unsigned short x_bottom_right, unsigned short y_bottom_right, unsigned short color);, Esta función trabaja de forma similar a la función de graficación de líneas, la diferencia radica en que las coordenadas ingresadas son los puntos diagonales del rectángulo.

La impresión de rectángulos con relleno es posible por medio de la siguiente función:

Glcd_Box(unsigned short x_upper_left, unsigned short y_upper_left, unsigned short x_bottom_right, unsigned short y_bottom_right, unsigned short color); Su funcionamiento es equivalente a la función Glcd_Rectangle, con la diferencia en el parámetro del color que es correspondiente a color de relleno.

La graficación de círculos se realiza por medio de la función:

Glcd_Circle(int x_center, int y_center, int radius, unsigned short color);, está función respeta las mismas condiciones del parámetro del color de las anteriores funciones, tiene las coordenadas del centro del circulo y el radio del mismo.

Para demostrar el funcionamiento de estás funciones observe y analice el siguiente ejemplo:

```
void main( void )  
{
```

```

Glcd_Init(); //Inicialización del display gráfico.
Glcd_Fill(0); //Borra el display
Glcd_Set_Font( Font_Glcd_5x7 , 5, 7, 32); // Carga fuente de texto 5x7
delay_ms(500); //Retardo de 500ms
Glcd_Dot(4,4,1); //Se dibuja un píxel en la coordenada (4,4)
delay_ms(500); //Retardo de 500ms
Glcd_Line(6,6,30,30,1); //Dibuja una línea de (6,6) a (30,30)
delay_ms(500); //Retardo de 500ms
Glcd_Box(40,5,70,35,2); //Rectángulo relleno entre (40,5) y (70,35)
delay_ms(500); //Retardo de 500ms
Glcd_Circle(64, 32, 20, 2); //Círculo con centro en (64,32) y radio de 20
Glcd_Write_Text( "Hola mundo!", 3, 7, 1); //Impresión del texto en las coordenadas (3,7)

while(1)//Bucle infinito.
{
}
}

```

Lamentablemente los dispositivos LCD gráficos de Proteus tienen una falla de diseño que hace que la simulación no funcione correctamente, este es un problema que se espera que sea corregido en versiones futuras de ISIS. Sin embargo los resultados de la simulación son muy cercanos al comportamiento real del dispositivo de graficación. El resultado en ISIS, sobre la simulación y una vista del comportamiento real sobre el LCD gráfico, se pueden apreciar en las siguientes figuras:

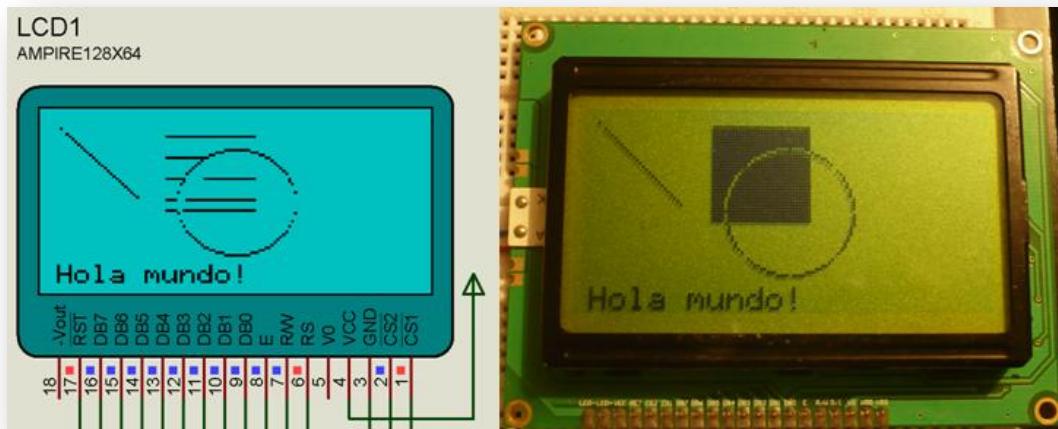


Figura 5-22

6 Teclados y sistemas de entrada de datos

La interacción de los microcontrolados requiere de sistemas de entrada para los datos con el usuario. Para este propósito se pueden usar dispositivos como pulsadores, teclados matriciales, o paralelos, e incluso teclados PS2 como los que usan los ordenadores de escritorio. Este capítulo se centra en el estudio de estos dispositivos.

6.1 Uso de pulsadores

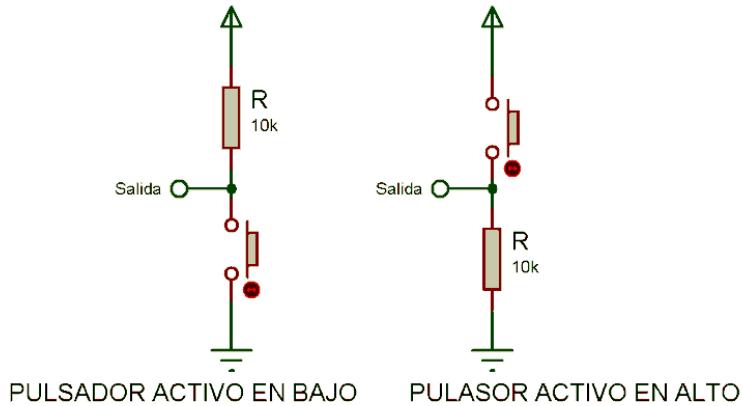
La implementación de pulsadores es una de las alternativas de mayor popularidad en las interfaces de acción con los usuarios. Los pulsadores son de uso simple y de costo económico en la implementación. Los pulsadores pueden ser normalmente abiertos o normalmente cerrados al efecto eléctrico ante el flujo de corriente. La implementación de estos dispositivos es propensa a los efectos de los rebotes o ruidos de tensión eléctrica cuando cambian de estado. Dado el veloz procesamiento de los microcontroladores estos efectos de ruido, hacen que el PIC pueda detectar cambios o estados lógicos no definidos. Para comprender este concepto observe la siguiente gráfica que muestra el comportamiento de este ruido:



Figura 6-1

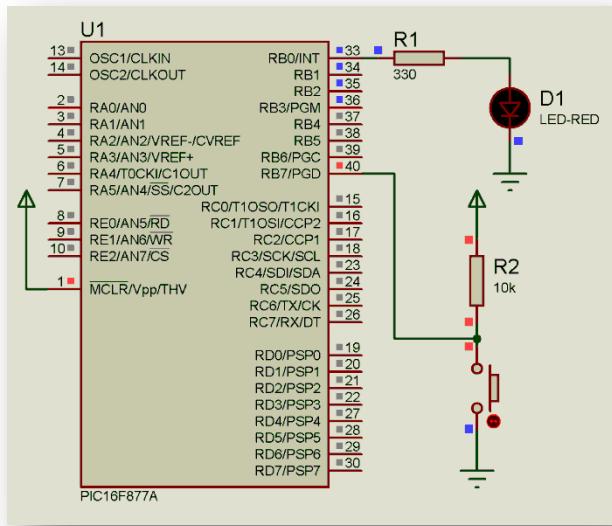
Cuando los cambios de tensión cruzan la zona indefinida, generan cambios de alto a bajo y viceversa que el microcontrolador detecta como un pulso. Para evitar el efecto del ruido o rebotes se debe realizar una pausa en espera de la estabilidad del estado lógico. La duración promedio de un rebote es de 1 a 5 milisegundos, lo que indica que se debe hacer un retardo superior a este tiempo para esperar la estabilidad. Un retardo adecuado para este efecto es un tiempo mayor o igual a 10 milisegundos. Este retardo se debe aplicar después de detectar el primer cambio sobre el pulsador.

Para el tratamiento de los pulsadores y la eliminación de los rebotes el compilador MikroC PRO cuenta con la librería Button, que se puede encontrar en la paleta o pestáña de librerías. Esta librería tiene la única función: ***unsigned short Button(unsigned short *port, unsigned short pin, unsigned short time, unsigned short active_state);***, Donde *port* es el puerto donde se conecta el pulsador, *pin* es el bit del puerto donde se conecta el pulsador, *time* es el tiempo en milisegundo de espera del ruido, y *active_state*, es el estado lógico para el cual se desea hacer la activación del pulsador. La función retorna 0 si el pulsador no está activo y 255 si el mismo está activo. La instalación de los pulsadores se puede hacer de dos formas, activo en alto o activo en bajo, en la siguiente figura se puede apreciar la forma de configurar estas dos posibilidades:



Circuito 6-1

La decisión de usar la activación en alto o en bajo, depende del desarrollador quien debe analizar la forma de mayor simplicidad en el momento de hacer el diseño. Cabe recordar que de todos modos la activación final puede ser invertida con la función de la librería. Para analizar y estudiar el uso de esta librería se creará un nuevo proyecto con los siguientes dispositivos: PIC 16F877A, BUTTON, RES, y LED-RED. El circuito correspondiente en ISIS, es el siguiente:



Circuito 6-2

El programa del microcontrolador respectivo es el siguiente:

```
void main( void )
{
    //Configuración de puertos.
    TRISB=0xF0;
    PORTB=0;
    while(1)//Bucle infinito.
```

```

{
    if( Button(&PORTB, 7, 100, 0 )//Evalúa el estado del pulsador por RB7,
    //activado en bajo.
        PORTB.F0=1; //Prende el LED si el pulsador está activo.
    else
        PORTB.F0=0; //Apaga el pulsador si el pulsador está no activo.
}
}

```

Al correr esta simulación el LED debe prender cuando se pulsa el botón, y apagarse cuando se suelta. El siguiente ejemplo conmuta el estado del LED, para este se evalúa el estado del pulsador y se espera que se suelte el mismo:

```

void main( void )
{
    //Configuración de puertos.
    TRISB=0xF0;
    PORTB=0;
    while(1)//Bucle infinito.
    {
        if( Button(&PORTB, 7, 100, 0 )//Evalúa el estado activo del pulsador.
        {
            if( PORTB.F0==1 )// Se conmuta el estado del LED.
                PORTB.F0=0;
            else
                PORTB.F0=1;
            //Se espera a que el pulsador este no activo.
            while( Button(&PORTB, 7, 100, 0 ) );
        }
    }
}

```

6.2 Uso de Dip-Switch

Los Dip-Switch, son dispositivos mecánicos que contienen múltiples interruptores en un solo encapsulado. Estos dispositivos permiten configurar de forma simple las características binarias de los sistemas microcontrolados. Los Dip-Switch, se consiguen comercialmente en tamaños, colores, y número de interruptores diferentes. La apariencia física y la vista en el simulador ISIS, de estos dispositivos es la siguiente:

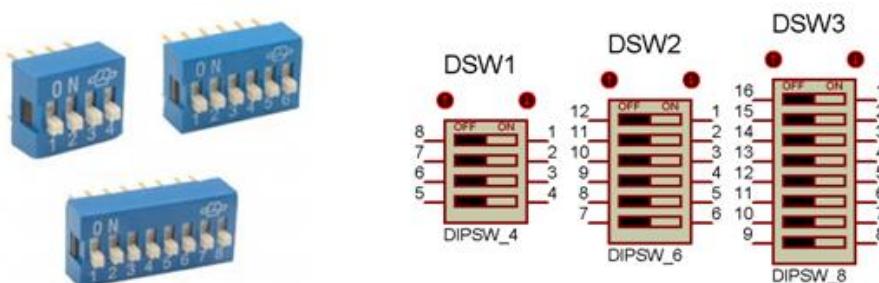
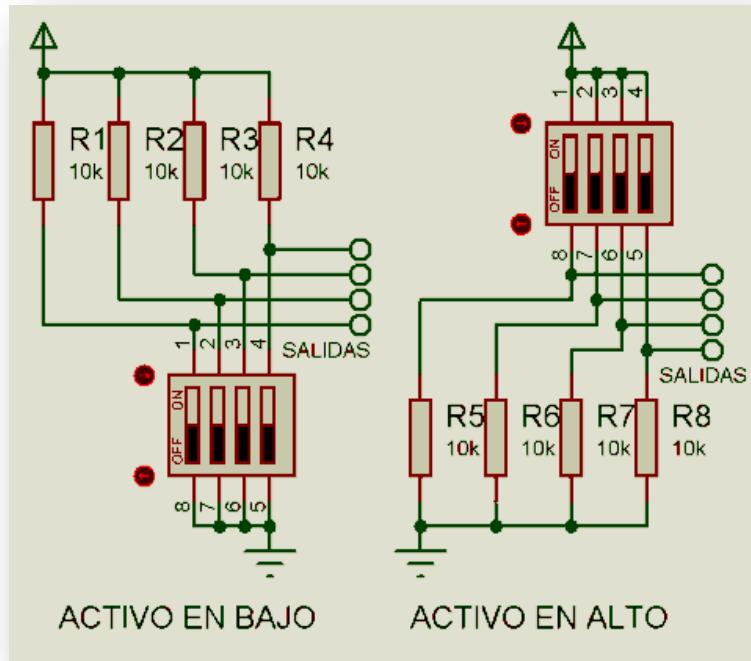


Figura 6-2

El uso de los Dip-Switch es similar a los pulsadores y se puede configurar de la misma forma que un pulsador con activación en alto o en bajo. Las configuraciones antes citadas se pueden apreciar en el siguiente circuito:

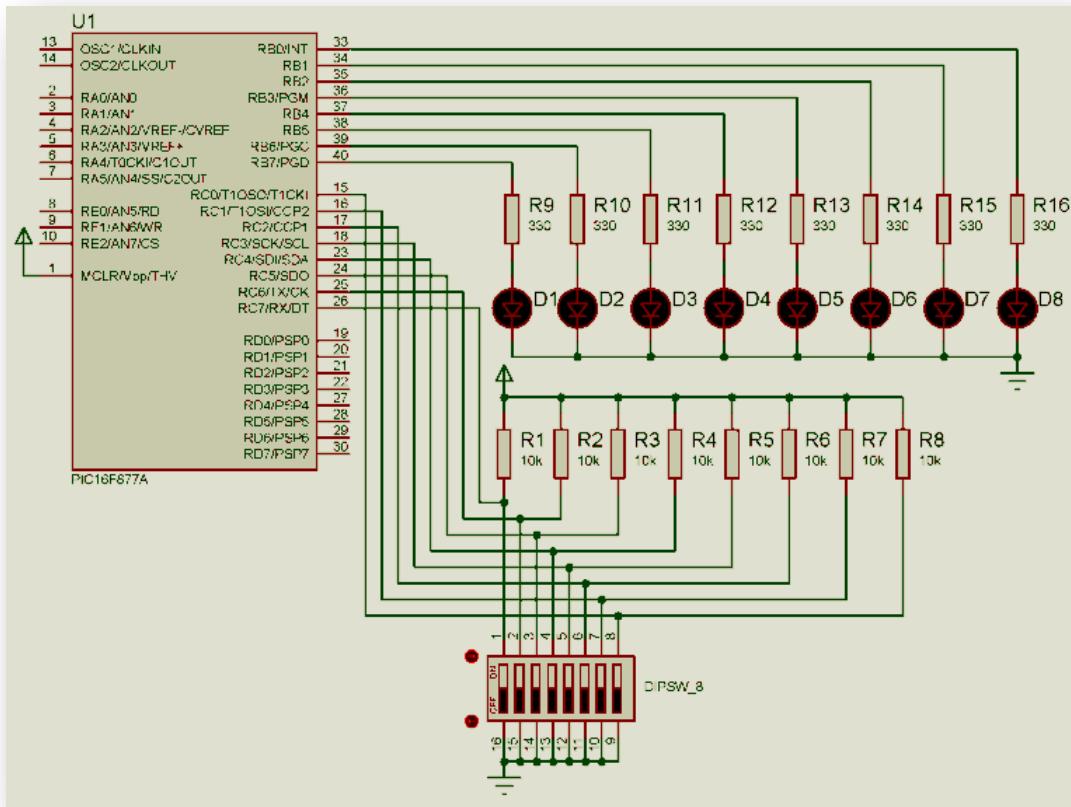


Circuito 6-3

Para comprender y demostrar el uso de los Dip-Switch con los microcontroladores PIC, observe y analice el siguiente ejemplo:

```
void main( void )
{
    //Configuración de puertos.
    TRISB = 0;
    PORTB = 0;
    TRISC = 255;
    while(1) //Bucle infinito.
    {
        //Se guarda el valor complementado del puerto C en el puerto B.
        PORTB = ~PORTC;
    }
}
```

Para ejecutar la simulación en ISIS, se usan los siguientes dispositivos: 16F877A, RES, LED-RED, DIPSW_8. Seguidamente se implementa el siguiente circuito en ISIS:



Circuito 6-4

Al correr la simulación los LEDs muestran el estado del Dip-Switch.

6.3 Uso de teclados matriciales

Las aplicaciones con microcontroladores, requieren en algunos casos el uso de teclas de entrada de datos, para datos numéricos, funciones e incluso caracteres de texto. La opción de mayor practicidad es el uso de teclados matriciales, estos consisten en un arreglo de pulsadores alineados en filas y columnas, minimizando el número de conexiones eléctricas. En las siguientes imágenes se puede ver la apariencia física de un teclado matricial de 4x4 y su equivalente esquemático:

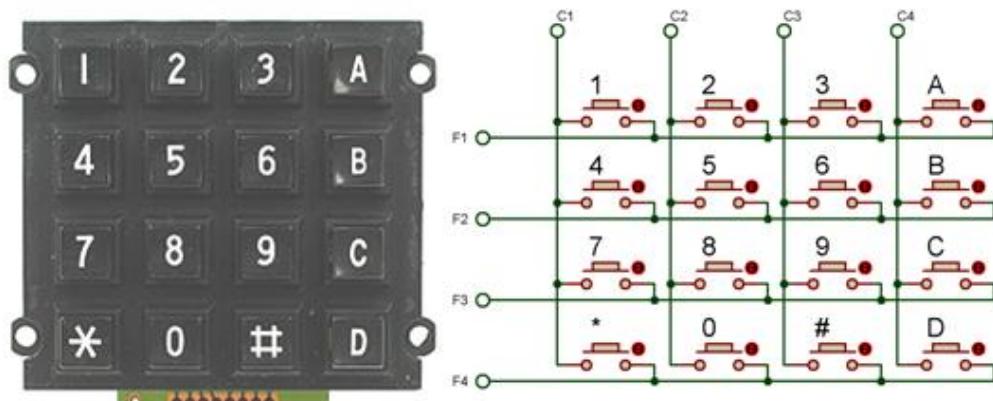
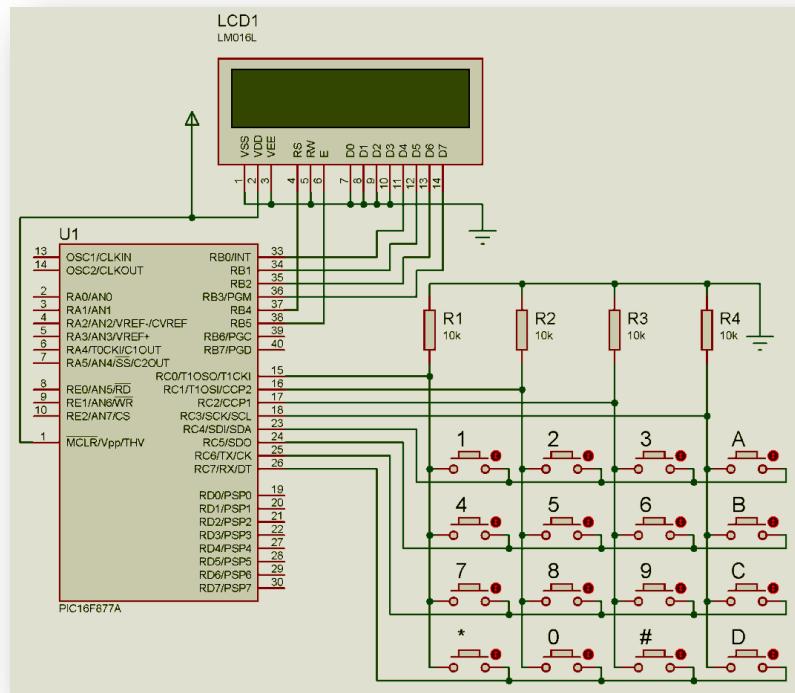


Figura 6-3

Los teclados matriciales pueden tener dimensiones mayores en función de la necesidad del desarrollador. Los teclados especiales pueden ser fabricados con membranas plásticas con la cantidad de teclas y la distribución de las mismas, de tal manera que suplan las necesidades del usuario. Sin embargo los teclados 4x4 permiten hacer una interfaz lo suficientemente completa para muchas de las aplicaciones. Los teclados matriciales funcionan activando una de 4 columnas y revisando cual de las filas se activan, este proceso determina cual es la tecla pulsada, de igual manera el análisis se puede hacer invirtiendo las columnas con las filas. El compilador MikroC PRO, contiene una librería que controla y lee un teclado de 4x4, está librería cuenta con tres funciones para este propósito. La librería que permite el uso de teclados 4x4 es: *Keypad4x4*, y las funciones que usa son: *Keypad_Init(void)*; está inicializa el uso del teclado en función del puerto designado para este fin. La función: *char Keypad_Key_Press(void)*; Esta función, retorna un valor de 0 a 16, donde 0 representa el teclado totalmente inactivo, los resultados del 1 al 16 representan las 16 posibles teclas. Esta función retorna inmediatamente el estado del teclado cada vez que es invocada. La última función de la librería para el teclado matricial es: *char Keypad_Key_Click(void)*; Su comportamiento es idéntico a la función anterior, su diferencia vital es que si la función detecta una tecla pulsada, solo retorna su valor hasta que esta tecla es soltada.

Para estudiar las características de este tipo de teclados, se mostrará un ejemplo con un nuevo proyecto en ISIS, que requiere de los siguientes dispositivos: PIC 16F877A, RES, BUTTON, LM016L, este último es un display de caracteres de 2x16. De la misma forma se debe crear el proyecto nuevo en el compilador MikroC PRO.

Para iniciar el proyecto se debe construir un circuito en ISIS, como el visualizado en la siguiente figura:



Circuito 6-5

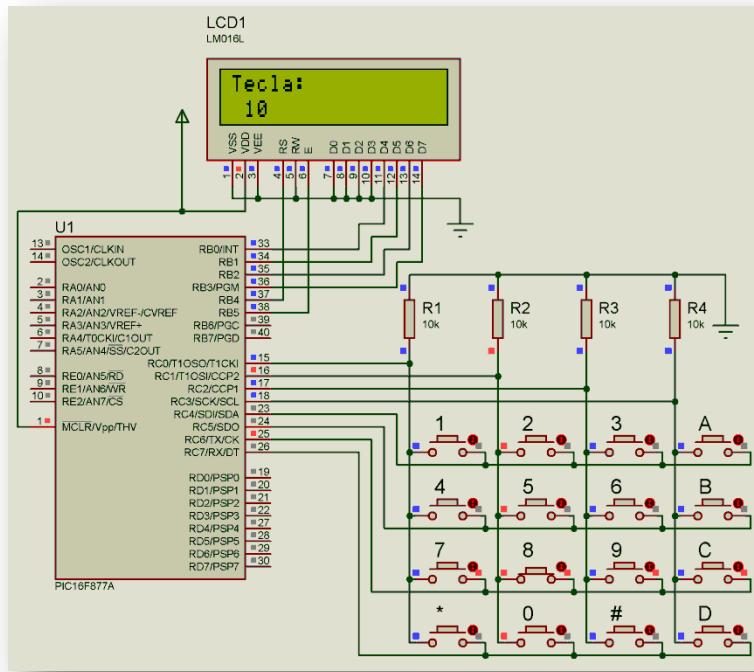
El programa diseñado en MikroC PRO, leerá el teclado y mostrará en el LCD, el valor retornado por el mismo. Para este fin observe y analice el siguiente ejemplo:

```
//Declaración del puerto para el teclado 4x4
char keypadPort at PORTC;
//Definición pines para el LCD.
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;

//Definición de registros TRIS para el LCD.
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;

void main( void )
{
    //Declaración de variables.
    unsigned short Tecla;
    char Text[20];
    //Configuración e inicialización del PIC.
    Lcd_Init(); //Inicialización del LCD.
    Lcd_Cmd(_LCD_CURSOR_OFF); //Se apaga el cursor.
    Lcd_Out(1, 1,"Tecla:"); //Se imprime texto.
    while(1)//Bucle infinito.
    {
        Tecla=Keypad_Key_Press(); //Se lee el teclado y se guarda el resultado en Tecla.
        ByteToStr(Tecla, Text); //Conversión de entero a texto.
        Lcd_Out(2,1,Text); //Visualización del valor retornado por el teclado.
    }
}
```

Al correr la simulación de este programa en la pantalla se pueden ver los valores retornados por el teclado. En la siguiente figura se puede apreciar un a vista de la simulación cuando se pulsa la tecla rotulada con el número 8:



Circuito 6-6

Para el caso puntual de este ejemplo el teclado hace un barrido de izquierda a derecha y de arriba hacia abajo, empezando con la tecla 1, y terminando con la tecla D. Esto significa que los retornos de las funciones de lectura para el teclado son los siguientes datos con respecto a cada una de las teclas:

| <i>Retorno de las funciones</i> | <i>Equivalecia</i> |
|---------------------------------|-----------------------|
| 0 | Ninguna tecla pulsada |
| 1 | Tecla 1 |
| 2 | Tecla 2 |
| 3 | Tecla 3 |
| 4 | Tecla A |
| 5 | Tecla 4 |
| 6 | Tecla 5 |
| 7 | Tecla 6 |
| 8 | Tecla B |
| 9 | Tecla 7 |
| 10 | Tecla 8 |
| 11 | Tecla 9 |
| 12 | Tecla C |
| 13 | Tecla * |
| 14 | Tecla 0 |
| 15 | Tecla # |
| 16 | Tecla D |

Tabla 6-1

Hasta este punto se pueden leer del teclado una serie de números pero estos no son equivalentes a la tecla pulsada, para corregir este aspecto se debe implementar una función que decodifique los valores entregados por el teclado y los convierta en el verdadero carácter que cada una de las teclas representa. Para este fin observe la siguiente función que se hace por medio de una estructura *switch case* para decodificar el teclado.

```
//Función para decodificar el teclado.
char LeerTeclado( void )
{
    //Estructura switch case para evaluar los valores retornados
    //por la librería del teclado.
    switch(Keypad_Key_Press() )
    {
        case 1: return '1';
        case 2: return '2';
        case 3: return '3';
        case 4: return 'A';
        case 5: return '4';
        case 6: return '5';
        case 7: return '6';
        case 8: return 'B';
        case 9: return '7';
        case 10: return '8';
        case 11: return '9';
        case 12: return 'C';
        case 13: return '*';
        case 14: return '0';
        case 15: return '#';
        case 16: return 'D';
        default: return 0; //Tecla no pulsada.
    }
}
```

El programa final de esta aplicación es el siguiente:

```
//Declaración del puerto para el teclado 4x4
char keypadPort at PORTC;

//Definicion pines para el LCD.
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;

//Definicion de registros TRIS para el LCD.
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
```

```

sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;

//Función para decodificar el teclado.
char LeerTeclado( void )
{
    //Estructura switch case para evaluar los valores retornados
    //por la librería del teclado.
    switch( Keypad_Key_Press() )
    {
        case 1: return '1';
        case 2: return '2';
        case 3: return '3';
        case 4: return 'A';
        case 5: return '4';
        case 6: return '5';
        case 7: return '6';
        case 8: return 'B';
        case 9: return '7';
        case 10: return '8';
        case 11: return '9';
        case 12: return 'C';
        case 13: return '*';
        case 14: return '0';
        case 15: return '#';
        case 16: return 'D';
        default: return 0; //Tecla no pulsada.
    }
}

void main( void )
{
    //Declaración de variables.
    char Tecla;
    //Configuración e inicialización del PIC.
    Lcd_Init(); //Inicializa el LCD.
    Lcd_Cmd(_LCD_CURSOR_OFF); //Se apaga el cursor.
    Lcd_Out(1, 1,"Tecla:");
    while(1)//Bucle infinito.
    {
        Tecla=LeerTeclado(); //Se lee el teclado y su resultado se guarda en Tecla.
        Lcd_Ch(2,1,Tecla); //Visualización el valor retornado por el teclado.
    }
}

```

Una vez editado, compilado y simulado este programa, se debe ver en la simulación del display LCD, el carácter correspondiente a la tecla que se está pulsando sobre el teclado.

6.4 Uso de teclados PS2 o Din

Los teclados PS2 o Din son de uso común en los ordenadores de escritorio, e incluso existen algunos que son flexibles, y de fácil transporte. El compilador MikroC PRO, posee una librería especializada en el uso de estos teclados, lamentablemente estos teclados no son simulables con el programa ISIS. Por esta razón el ejemplo que se expone en este capítulo debe ser probado con los elementos de forma real. La apariencia física de estos teclados es la siguiente:



Figura 6-4

La lectura de estos teclados es hace por medio de un protocolo serial que la librería ya tiene implementada. Los teclados cuentan con un terminal PS2 o DIN, en los cuales se pueden identificar 4 pines fundamentales, el Vcc o poder, el Gnd o referencia, un pin de datos seriales, y un pin de reloj. La comunicación de estos dispositivos es de forma síncrona, lo que hace necesario la señal de reloj.

La distribución de pines de estos terminales es la que se puede apreciar en la siguiente gráfica:

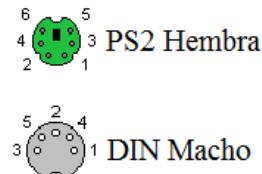


Figura 6-5

En la siguiente tabla se puede apreciar la distribución de pines de la figura anterior:

| Pin PS2 | Finalidad | Pin DIN |
|---------|----------------|---------|
| 1 | Datos | 2 |
| 3 | Referencia GND | 4 |
| 4 | Vcc, +5V | 5 |
| 5 | Reloj | 1 |

Tabla 6-2

La implementación de este tipo de teclados se hace por medio de dos funciones contenidas en la librería *PS2*, de MikroC PRO. La primera función es: *Ps2_Config()*; Esta función debe ser invocada en la configuración del PIC, dentro de la función *main*, la función configura e inicializa el teclado para poder ser leído en función de los pines que sean definidos con anterioridad como pin de reloj y pin de datos. La otra función usada es: *unsigned short Ps2_Key_Read(unsigned short *value, unsigned short *special, unsigned short *pressed)*; esta función cuenta con un parámetro de salida que retorna 1, cuando una tecla es leída con éxito, y retorna 0, cuando no se

detecta ninguna tecla pulsada. La función posee tres parámetros de entrada que permiten identificar las siguientes características: *value*, este es un apuntador a una variable de tipo *unsigned short* en la cual se guarda el valor de la tecla pulsada en código ASCII.

El parámetro *special*, es un apuntador a una variable de tipo *unsigned short* que especifica si la tecla que se está pulsando es de origen especial como: F1, F2, ESC, ENTER, etc. Si esta variable retorna un 1, significa que el valor codificado en *value*, no es un código ASCII, si no un código asignado a la tecla especial que está regido por la siguiente tabla:

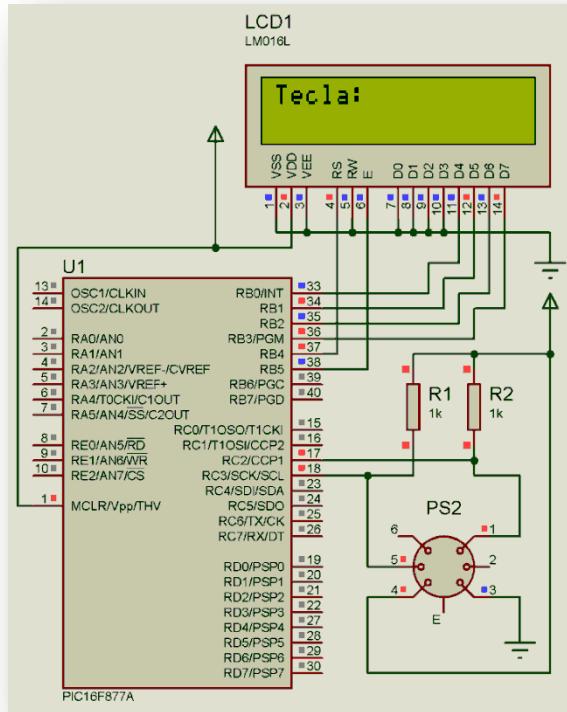
| Tecla pulsada | Retorno en value |
|----------------------|-------------------------|
| <i>F1</i> | 1 |
| <i>F2</i> | 2 |
| <i>F3</i> | 3 |
| <i>F4</i> | 4 |
| <i>F5</i> | 5 |
| <i>F6</i> | 6 |
| <i>F7</i> | 7 |
| <i>F8</i> | 8 |
| <i>F9</i> | 9 |
| <i>F10</i> | 10 |
| <i>F11</i> | 11 |
| <i>F12</i> | 12 |
| <i>Enter</i> | 13 |
| <i>Page Up</i> | 14 |
| <i>Page Down</i> | 15 |
| <i>Backspace</i> | 16 |
| <i>Insert</i> | 17 |
| <i>Delete</i> | 18 |
| <i>Windows</i> | 19 |
| <i>Ctrl</i> | 20 |
| <i>Shift</i> | 21 |
| <i>Alt</i> | 22 |
| <i>Print Screen</i> | 23 |
| <i>Pause</i> | 24 |
| <i>Caps Lock</i> | 25 |
| <i>End</i> | 26 |
| <i>Home</i> | 27 |
| <i>Scroll Lock</i> | 28 |
| <i>Num Lock</i> | 29 |
| <i>Left Arrow</i> | 30 |
| <i>Right Arrow</i> | 31 |
| <i>Up Arrow</i> | 32 |
| <i>Down Arrow</i> | 33 |
| <i>Escape</i> | 34 |
| <i>Tab</i> | 35 |

Tabla 6-3

Si el valor returned es 0 el código consignado en *value* es un carácter ASCII.

Por último el parámetro: *pressed*, determina si la tecla denotado por los anteriores parámetros es pulsada o soltada, si el valor returned en *pressed*, es 1 la tecla fue pulsada, si retorna 0 la tecla fue soltada.

Para el próximo ejemplo se debe tener presente el siguiente circuito electrónico de prueba:



Circuito 6-7

El código de programa para este ejercicio es el siguiente, observe y analice:

```
//Definición de pines para el teclado PS2.
sbit PS2_Data at RC2_bit;
sbit PS2_Clock at RC3_bit;
sbit PS2_Data_Direction at TRISC2_bit;
sbit PS2_Clock_Direction at TRISC3_bit;
```

```
//Definición pines para el LCD.
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;
```

```

//Definición de registros TRIS para el LCD.
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;

void main( void )
{
    //Declaración de variables.
    unsigned short Tecla = 0, Especial = 0, Pulsada_soltada = 0;

    //Configuración e inicialización del PIC.
    Lcd_Init(); //Inicializa el LCD.
    Lcd_Cmd(_LCD_CURSOR_OFF); //Se apaga el cursor.
    Lcd_Out(1, 1,"Tecla:"); //Se imprime texto.
    while(1)//Bucle infinito.
    {
        //Este ciclo while espera a que una tecla sea activada.
        while( Ps2_Key_Read( &Tecla, &Especial, &Pulsada_soltada)==0 );

        //La sentencia if evalúa el estado de Pulsada y muestra en el LCD
        //el resultado.
        if( Pulsada_soltada==1 )
            Lcd_Out(1, 7,"Pulsada");
        else
            Lcd_Out(1, 7,"Soltada");

        //La sentencia if evalúa el estado de Especial y muestra
        //su resultado en el LCD.
        if( Especial==1 )
            Lcd_Out(2, 3,"Especial");
        else
            Lcd_Out(2, 3,"ASCII  ");

        //Visualización el valor returnedo por el teclado.
        Lcd_Ch(2,1,Tecla);
    }
}

```

Después de realizar la prueba de este ejemplo en display LCD, mostrar los datos enviados por el teclado, y el estado de las teclas especiales.

7 Comunicaciones seriales

Los desarrollos con microcontroladores requieren en algunos casos la implementación de comunicaciones seriales para establecer transporte de datos con otros dispositivos tales como: memorias, sensores, ordenadores, e incluso otros microcontroladores. Con el fin de realizar las comunicaciones seriales, algunos microcontroladores cuentan con módulos seriales como: I²C, SPI, USART, y USB. Cada uno de estos formatos de comunicación, permiten establecer comunicaciones con módulos diferentes. El módulo I²C, es ideal para la comunicación con memorias seriales como la 24LC64, 24LC128, 24LC512, entre otras. El protocolo SPI, permite establecer comunicaciones con unidades de almacenamiento masivo como las memorias SD. El módulo USART es uno de los más utilizados; este módulo permite hacer comunicaciones con dispositivos como sensores, módulos de transmisión y recepción XBee, ordenadores personales, módulos GPS, y otros micros. Por último el módulo USB, que está incorporado en unos pocos micros, como el 18F2550, y el 18F4550, permite hacer la comunicación con un ordenador personal por medio de un puerto USB, definido como HID o dispositivo de interfaz humana.

7.1 Modulo serial I²C

Algunos microcontroladores poseen este módulo y de la misma forma el compilador MikroC PRO cuenta con una librería especializada para el uso fácil de este protocolo. La librería se reconoce como *I2C*. Esta librería cuenta con siete funciones que lideran el funcionamiento del módulo. El protocolo I²C, se caracteriza por tener una línea de datos bidireccional y una línea de reloj con drenador abierto. Por este motivo en estos dos pines se debe usar una resistencia de pull-up, generalmente de 10KΩ. Este usa una condición de inicio, un bloque de dirección para los dispositivos en la red, un bloque de datos y una condición de fin. De igual manera usa las condiciones de repetición de inicio y un bit de reconocimiento o *acknowledge* denotado como *ACK*. El protocolo I²C se puede usar para conectar diferentes dispositivos con direcciones diferentes en una topología en forma de bus. Por último este protocolo debe tener una velocidad de comunicación que está dada en bits por segundo.

La librería de MikroC PRO, cuenta con las siguientes funciones que hacen posible el dominio del protocolo I²C:

La función: *I2C1_Init(const unsigned long clock);*, inicializa los pines de comunicación en el PIC, y ajusta la velocidad de comunicación a la rata que denote el parámetro *clock*.

La función: *unsigned short I2C1_Start(void);*, está función genera la condición de inicio y retorna 0 si no hay ningún error en la transmisión, de otro forma retorna un valor diferente de 0.

La función: *void I2C1_Repeated_Start(void);*, está función genera la repetición de inicio.

La función: *unsigned short I2C1_Is_Idle(void);*, está función se usa para identificar si el bus de datos está ocupado, y retorna 1, si el bus está disponible o retorna 0, si está ocupado.

La función: *unsigned short I2C1_Rd(unsigned short ack);*, Esta función lee un dato del bus de datos y envía la confirmación *ack* por el bus. Si *ack* vale 1 la confirmación es positiva y si es 0 la confirmación es negativa o NO ACK.

La función: ***unsigned short I2C1_Wr(unsigned short data);***, está función envía el dato *data* por el bus y retorna 0 si la transmisión fue exitosa, o algo diferente de 0 si suceden errores.

La función: ***void I2C1_Stop(void);***, Genera la condición de final en el bus de comunicación.

Para entender y estudiar el funcionamiento de este módulo, el siguiente ejemplo se concentra en la lectura y escritura de una memoria serial 24LC00, que tiene una capacidad de 16 direcciones. En las siguientes gráficas se puede apreciar el protocolo de comunicación con la memoria 24LC00 para ser leída y escrita:

Protocolo de escritura

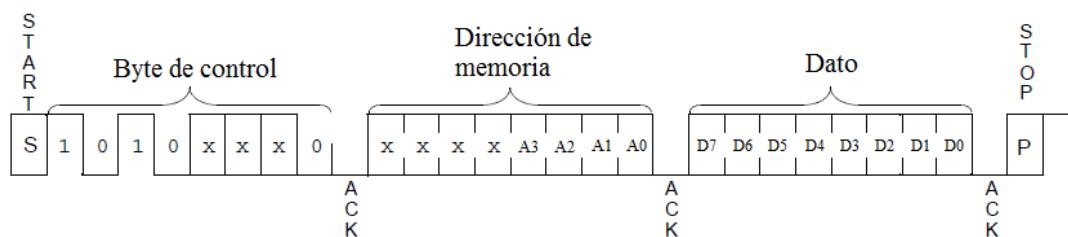


Figura 7-1

Protocolo de lectura

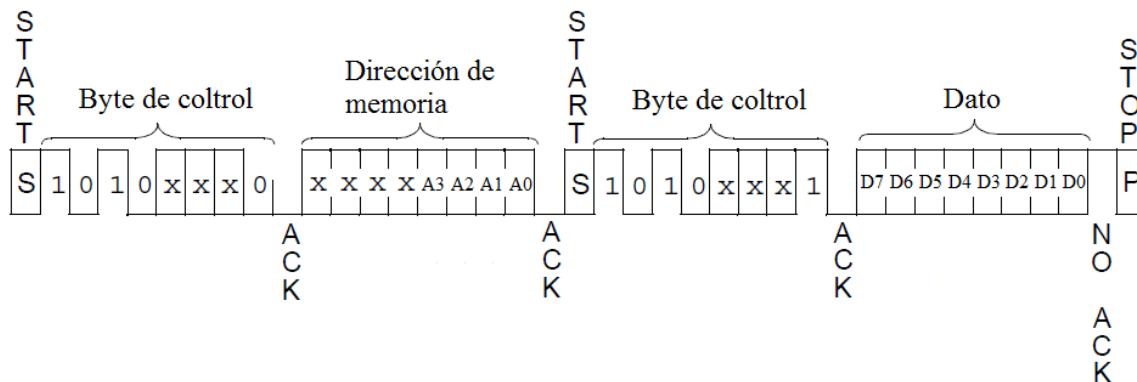
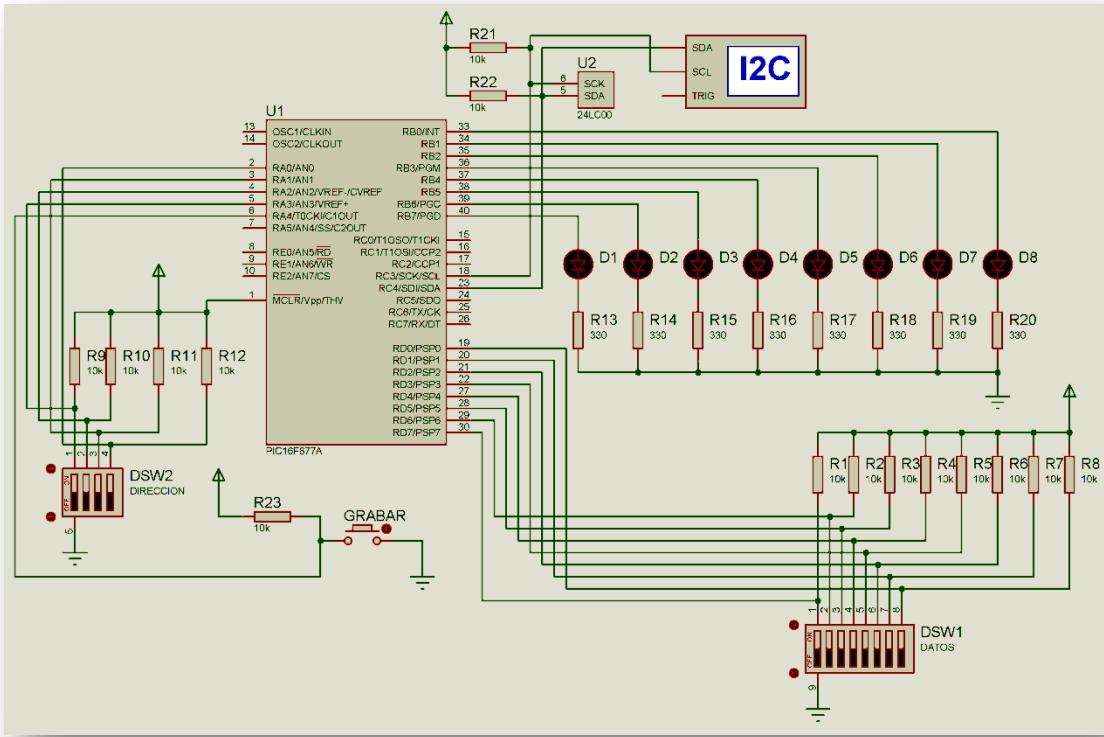


Figura 7-2

Para probar el ejemplo se debe implementar en ISIS, un circuito con los dispositivos: 16F877A, BUTTON, RES, LED-RED, 24LC00, DIPSWC_8, DIPSWC_4, y el instrumento de análisis del protocolo I2CDEBUGGER, este último se puede buscar en la paleta de herramientas ubicada en la parte izquierda del programa MikroC PRO, en el ícono de instrumentos: .

El circuito para armar es el siguiente:



Circuito 7-1

El código de programa correspondiente para esta simulación es el siguiente:

```

void main( void )
{
    //Declaración de variables.
    unsigned short DIRECCION;
    unsigned short DATO;
    //Configuración de puertos.
    ADCON1 = 6;
    TRISE = 7;
    TRISB = 0;
    PORTB = 0;
    //Inicio del módulo I2C a 100K bps.
    I2C1_Init(100000);
    while(1) //Bucle infinito.
    {
        DIRECCION = (~PORTA)&0x0F; //Captura de la dirección.
        //Protocolo de lectura de la dirección DIRECCION.
        I2C1_Start();
        I2C1_Wr(0b10100000);
        I2C1_Wr(DIRECCION);
        I2C1_Repeated_Start();
        I2C1_Wr(0b10100001);
        DATO=I2C1_Rd(0);
    }
}

```

```

I2C1_Stop();
//Visualización del DATO en el puerto B.
PORTB = DATO;
delay_ms(100); //Retardo de 100 ms para graficar.

//Sentencia if para evaluar si se pulsa el botón de grabar.
if( Button( &PORTA, 4, 50, 0 ) )
{
    DATO = ~PORTD; //Captura del valor del dato.
    //Protocolo de grabación.
    I2C1_Start();
    I2C1_Wr(0b10100000);
    I2C1_Wr(DIRECCION);
    I2C1_Wr(DATO);
    I2C1_Stop();
    //Retardo para esperar que la memoria grabe el dato.
    delay_ms(50);
}
}
}

```

Terminada la edición y la simulación en ISIS, se puede apreciar los valores consignados en la dirección que indica el DIP-SWITCH DIRECCION, sobre los LEDs. Cuando el botón GRABAR es pulsado, el valor del DIP-SWITCH, DATOS es grabado en la dirección actual. Los pines de Reloj, y Datos para este PIC, ya están predefinidos por el fabricante por los pines RC3, y RC4 respectivamente.

7.2 Módulo USART

La USART, es un módulo de comunicación serial estándar, de forma asíncrona, está característica lo hace muy apetecido dado que requiere un solo medio de transmisión para enviar información, y no requiere un medio para el reloj. La señal de reloj, o sincronismo lo deben asumir, independiente mente cada uno de los elementos, el transmisor y el receptor. Otra ventaja de este módulo es que cuenta con comunicación full-duplex, es decir que puede transmitir y recibir información al mismo tiempo. Para este propósito se usan dos medios de transmisión dedicados, uno solo para transmitir y uno solo para recibir.

La comunicación síncrona cuenta con las siguientes características: un bit de inicio o de *start*, que siempre es un 0 lógico, 8 o 9 bits de datos para el caso puntual de los PIC, y 1, 1.5 o 2 bits de fin o *stop*. Por último la velocidad de transmisión que debe estar definida con el mismo valor en los dos dispositivos que se comunican, está por defecto en casi toda comunicación es de 9600 bps, sin embargo esto no es una regla puede ser mayor o menor.

En la siguiente gráfica se puede apreciar el comportamiento de la transmisión de un dato con este protocolo:

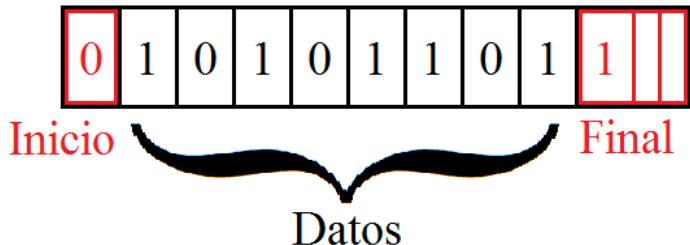


Figura 7-3

Este protocolo es utilizado por los ordenadores personales y otros dispositivos, y se conoce como RS232, para el caso puntual de este los niveles de tensión eléctrica son diferentes al microcontrolador. El protocolo RS232, representa el valor de un 0 lógico con una tensión de +12 voltios, y el valor de un 1 lógico con -12 voltios. Los ordenadores personales y los demás dispositivos que implementan el puerto RS232, usan un conector DB9, que tiene 9 pines, los cuales se identifican en la siguiente figura:

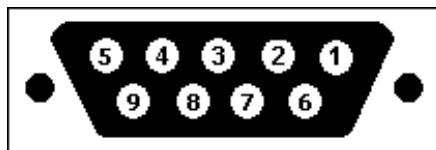


Figura 7-4

| Pin | Uso |
|------------|-----------------|
| 1 | <i>DCD</i> |
| 2 | <i>RXD</i> |
| 3 | <i>TXD</i> |
| 4 | <i>DTR</i> |
| 5 | <i>GND</i> |
| 6 | <i>DSR</i> |
| 7 | <i>RTS</i> |
| 8 | <i>CTS</i> |
| 9 | <i>No usado</i> |

Tabla 7-1

Para fines de comunicación asíncrona se debe configurar las conexiones de la siguiente forma:

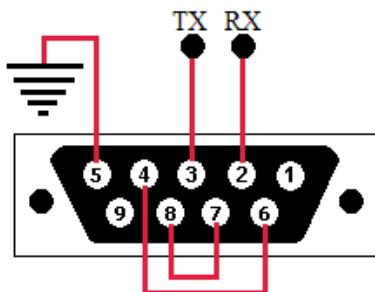
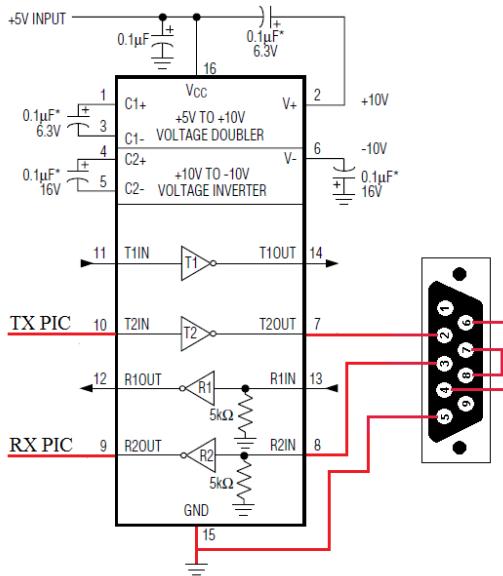


Figura 7-5

Para establecer y acoplar la comunicación de un PIC, con un dispositivo RS232, se debe usar un convertidor conocido como MAX232. La implementación de este, se debe hacer como se aprecia en la siguiente figura:



Circuito 7-2

Para el uso de este protocolo de comunicación el compilador MikroC PRO, cuenta con la librería: **UART**, está disponible en la paleta de librerías y cuenta con las siguientes funciones para su uso:

UART1_Init(*const unsigned long baud_rate*);, Esta función inicializa el módulo USART, y establece la velocidad de comunicación definida en el parámetro: *baud_rate*.

La función: **char UART1_Data_Ready();**, está función determina si hay un dato listo para ser leído, en el búfer de llegada del módulo, si la función retorna 1 el dato puede ser leído, de lo contrario no hay datos nuevos en el búfer.

La función: **char UART1_Tx_Idle();**, está función establece si el búfer de transmisión se encuentra ocupado enviando un dato, para esto retorna 1 si el búfer está ocupado, o 0 si el módulo está disponible para enviar un nuevo dato.

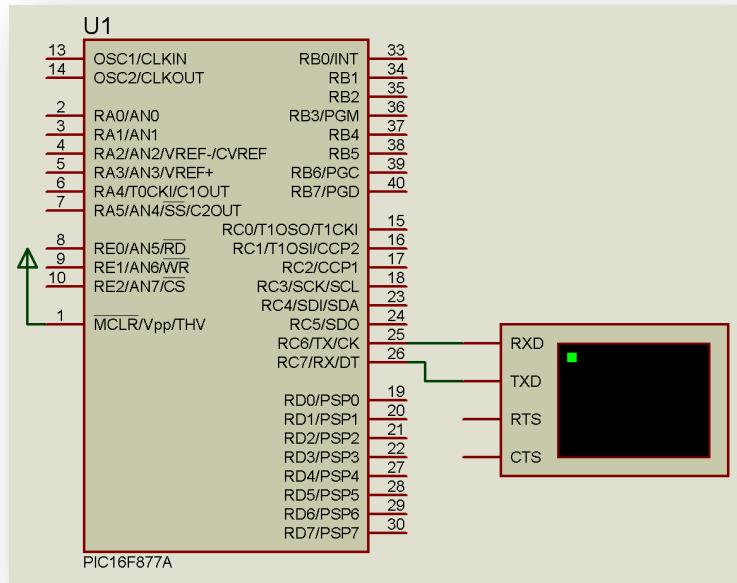
La función: **char UART1_Read();**, está función retorna el valor del búfer de entrada, es decir que sirve para leer un dato de entrada.

La función: **UART1_Read_Text(*char *Output, char *Delimiter, char Attempts*);**, está función lee una cadena de caracteres y la guarda en el apuntador *Output*, el apuntador *Delimiter*, es un apuntador a una cadena de caracteres que contiene el texto definido como fin de la cadena de texto de entrada, el parámetro *Attempts*, define la longitud de la cadena de caracteres del delimitador de fin de cadena *Delimiter*.

La función: **UART1_Write(*char _data*);**, transmite el dato *_data* ingresado en el parámetro de entrada, por la USART.

La función: `UART1_Write_Text(char * UART_text);`, Transmite una cadena de texto finalizada con el carácter nulo o 0. Esta cadena de texto es entregada por medio del parámetro `UART_text`.

Para realizar el siguiente ejemplo se debe implementar en ISIS, el circuito con los siguientes dispositivos: PIC 16F877A, y el instrumento virtual: VIRTUAL TERMINAL. Con estos dos elementos se construye el siguiente circuito:



Circuito 7-3

El instrumento VIRTUAL TERMINAL, es un simulador de comunicación serial y tiene un comportamiento similar a la herramienta HiperTerminal de Windows. Este instrumento permite editar las características de la comunicación como: el número de bits de datos, la velocidad de transmisión, el número de bits de parada, entre otros. Sin embargo el instrumento está configurado por defecto con una velocidad de 9600 bps, y características listas para usar con la USART del PIC.

Para comprobar el funcionamiento de este módulo se puede compilar y simular el siguiente ejemplo:

```
void main( void )
{
    //Declaración de variables.
    char DATO;
    UART1_Init(9600); //Inicio del módulo USART.
    //Se transmite el texto: de bienvenida.
    UART1_Write_Text("Bienvenido al simulador:");
    UART1_Write(13); //Se transmite el ASCII del ENTER.
    UART1_Write(10); //Se transmite el ASCII del retroceso del carro.
    //Se transmite el texto de pulsar tecla.
    UART1_Write_Text("Pulse una tecla!... ");
}
```

```

UART1_Write(13); //Se transmite el ASCII del ENTER.
UART1_Write(10); //Se transmite el ASCII del retroceso del carro.
while(1)//Bucle infinito.
{
    //La sentencia if evalúa si un dato está listo para leer.
    if(UART1_Data_Ready()==1)
    {
        //Se lee el DATO del bufer.
        DATO = UART1_Read();
        //Se imprime el texto de realimentación.
        UART1_Write_Text("Usted pulso la tecla: ");
        UART1_Write(DATO); //Se transmite el DATO recibido.
        UART1_Write(13); //Se transmite el ASCII del ENTER.
        UART1_Write(10); //Se transmite el ASCII del retroceso del carro.
    }
}
}

```

Durante la simulación el usuario puede enviar datos por el terminal virtual y ver la realimentación que el PIC, emite.

7.3 Módulo USB

La comunicación USB, es de gran utilidad para realizar aplicaciones que impliquen la transmisión de datos con un ordenador personal. La desventaja más notable de este módulo es que solo está disponible en pocos microcontroladores como los PIC 18F2550 y 18F4550. Sin embargo estos microcontroladores tienen características poderosas que los hacen ideales para la mayoría de las aplicaciones.

En las siguientes gráficas se puede apreciar las terminales de los conectores USB:

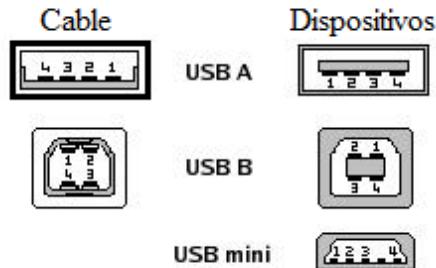


Figura 7-6

| Pin | Uso |
|------------|----------------|
| 1 | Vcc +5 Voltios |
| 2 | Dato - |
| 3 | Dato + |
| 4 | Gnd |

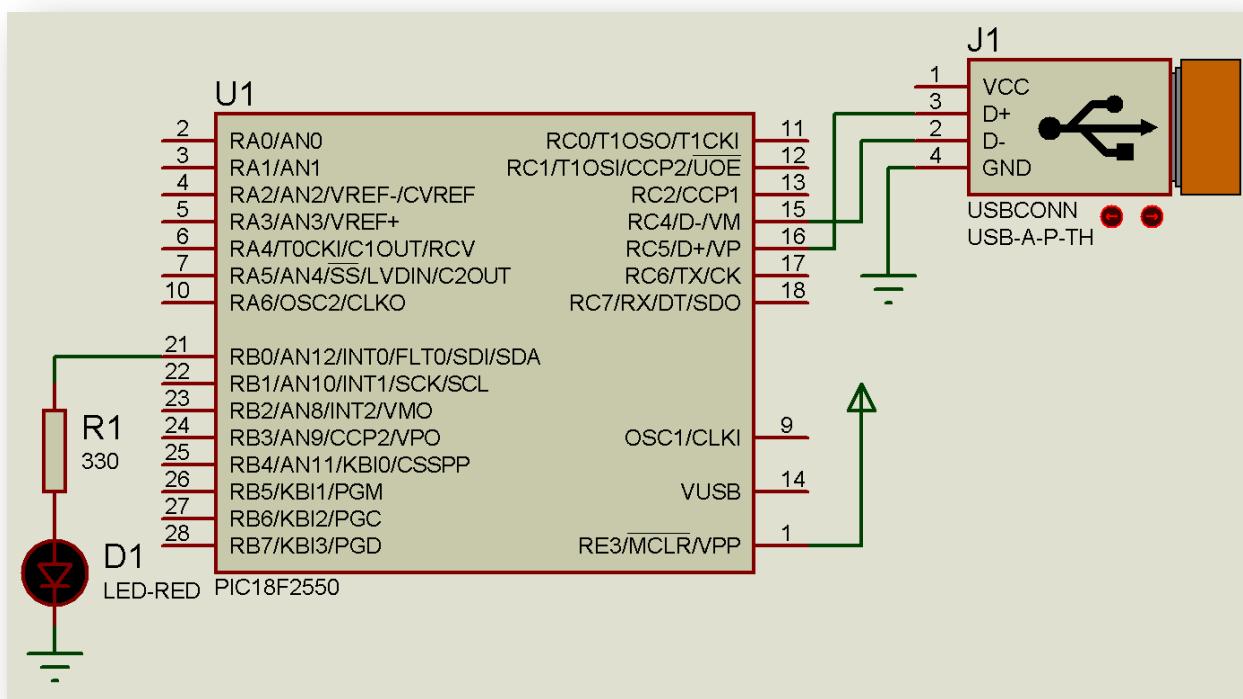
Tabla 7-2

Una ventaja de la conexión USB, es que puede entregar una alimentación de 5 voltios con una corriente máxima de 500mA, esto significa que si el desarrollo no supera este consumo, no es necesario implementar una fuente de poder externa.

El paso siguiente es realizar un circuito en ISIS, que permita correr la simulación del ejemplo. Para este caso se usa un PIC 18F2550, para este fin se requiere este dispositivo, el puerto virtual USBCONN, un LED LED-RED, y RES. Para usar este tipo de simulación se debe asegurar la instalación de los controladores USB, del simulador ISIS. Para hacer esta instalación se busca el directorio de instalación de Proteus, y correr la instalación de los controladores. La ubicación por defecto de estos controladores es la siguiente:

C:/labcenter Electronics/Proteus 7 Professional/USB Drivers/installer.exe

Posteriormente se debe hacer el siguiente circuito:



Circuito 7-4

Para fines prácticos en el terminal VUSB, se debe instalar un capacitor de 33uF conectado a referencia. Y se debe usar un cristal de cuarzo adecuado a las características de diseño. La opción de edición para los fusibles del PIC 18F2550, es de suma complejidad en comparación con los micros de baja gama. La configuración se edita picando el menú: *project* en el compilador MikroC PRO, y dentro de este menú se pica el submenú, *Edit Project...* está acción despliega una ventana de edición la cual debe quedar de la siguiente forma:

| | Configuration Registers |
|--|---|
| 96MHz PLL Prescaler | No Divide (4MHz input) |
| CPU System Clock Postscaler | [OSC1/OSC2 Src: /1][96MHz PLL Src: /2] |
| Full-Speed USB Clock Source Selection | Clock src from 96MHz PLL/2 |
| Oscillator | HS: HS+PLL, USB-HS |
| Fail-Safe Clock Monitor Enable | Disabled |
| Internal External Switch Over Mode | Disabled |
| Power Up Timer | Disabled |
| Brown Out Detect | Disabled in hardware, SBOREN disabled |
| Brown Out Voltage | 4.5V |
| USB Voltage Regulator | Enabled |
| Watchdog Timer | Disabled-Controlled by SWDTEN bit |
| Watchdog Postscaler | 1:32768 |
| CCP2 Mux | RC1 |
| PortB A/D Enable | PORTB<4:0> configured as digital I/O on RESET |
| Low Power Timer1 Osc enable | Disabled |
| Master Clear Enable | MCLR Enabled,RE3 Disabled |
| Stack Overflow Reset | Disabled |
| Low Voltage Program | Disabled |
| Extended Instruction Set Enable bit | Disabled |
| Background Debug | Disabled |
| Code Protect 00800-01FFF | Disabled |
| Code Protect 02000-03FFF | Disabled |
| Code Protect 04000-05FFF | Disabled |
| Code Protect 06000-07FFF | Disabled |
| Code Protect Boot | Disabled |
| Data EE Read Protect | Disabled |
| Table Write Protect 00800-01FFF | Disabled |
| Table Write Protect 02000-03FFF | Disabled |
| Table Write Protect 04000-05FFF | Disabled |
| Code Protect 06000-07FFF | Disabled |
| Code Protect Boot | Disabled |
| Data EE Read Protect | Disabled |
| Table Write Protect 00800-01FFF | Disabled |
| Table Write Protect 02000-03FFF | Disabled |
| Table Write Protect 04000-05FFF | Disabled |
| Table Write Protect 06000-07FFF | Disabled |
| Config. Write Protect | Disabled |
| Table Write Protect Boot | Disabled |
| Data EE Write Protect | Disabled |
| Table Read Protect 00800-01FFF | Disabled |
| Table Read Protect 02000-03FFF | Disabled |
| Table Read Protect 04000-05FFF | Disabled |
| Table Read Protect 06000-07FFF | Disabled |
| Table Read Protect Boot | Disabled |

Figura 7-7

Para fines prácticos se debe asegurar en la simulación una fuente de reloj de 48Mhz y en un circuito real se usa un cristal de 4Mhz, esto se debe a que el microcontrolador posee un PLL interno que multiplica la frecuencia.

El código fuente de MikroC PRO, es el siguiente:

```
//Declaración de variables globales
//como búfer de entrada y salida en el USB.
unsigned short Br[64];
unsigned short Bw[64];

//Función de rutina USB
//dentro de las interrupciones.
void HID_Inte( void )
{
    asm CALL _Hid_InterruptProc
    asm nop
}

//Función de interrupciones.
void interrupt( void )
{
    HID_Inte();
}

void main( void )
{
    //Declaración de variables.
    unsigned short n;
    char DATO;
    //Configuración del PIC, e interrupciones.
    INTCON = 0;
    INTCON2 = 0xF5;
    INTCON3 = 0xC0;
    RCON.IPEN = 0;
    PIE1 = 0;
    PIE2 = 0;
    PIR1 = 0;
    PIR2 = 0;
    //Configuración de puertos.
    TRISB = 0;
    PORTB = 0;

    Hid_Enable(&Br,&Bw); //Función de inicio para el puerto USB.
    //Ciclo for para inicializar el búfer en 0.
    for(n=0; n<64; n++) Bw[n]=0;

    while(1) //Bucle infinito.
    {
```

```

//Función para verificar los datos de entrada.
n=Hid_Read();
//Sentencia if para verificar el número bytes de entrada.
if( n!=0 )
{
    PORTB = 255; //Se prende todo el puerto B
    DATO = Br[0]; //Se lee el dato de entrada.
    //Se copia el texto, en el búfer de salida.
    strcpy( Bw, "Se recibio el dato: " );
    //Se copia el dato de entrada en el búfer.
    Bw[20]=DATO;
    //Se anexa el código ASCII del ENTER en el búfer de salida.
    Bw[21]=13;
    //Se anexa el código ASCII del retroceso del carro en el búfer de salida.
    Bw[22]=10;
    //Se envían los datos por el USB.
    HID_Write(Bw,64);
    delay_ms(100); //Retardo para el parpadeo del LED.
    PORTB = 0; //Se apaga el puerto B.
}
}

```

La configuración de los proyectos que usen la librería USB, es tediosa pero finalmente solo recurren a cuatro funciones para su uso. Estás funciones son las siguientes:

Hid_Enable(unsigned *readbuff, unsigned *writebuff);, está función configura el módulo USB, y establece la comunicación con la computadora. Incluye los parámetros de entrada *readbuff* y *writebuff*, que son apuntadores a los arreglos de memoria que guardan y transmiten la información por el puerto USB.

La función: **unsigned char Hid_Read(void);**, está función retorna la cantidad de datos listos para ser leídos del puerto USB.

La función: **unsigned short Hid_Write(unsigned *writebuff, unsigned short len);**, está función escribe en el puerto USB, la cantidad de bytes definidos en el parámetro *len*, que están consignados en el apuntador *writebuff*.

La función: **void Hid_Disable(void);**, está función deshabilita el uso del puerto USB.

Para verificar el funcionamiento de este ejemplo se debe correr la simulación y esperar a que el dispositivo USB, se instale automáticamente, dado que es un dispositivo de interfaz humana, compatible con Windows. Para enviar datos a la simulación se requiere correr una aplicación que viene con las herramientas de MikroC PRO. Esta herramienta se encuentra en *Tools*, y se pica el submenú, *HID Terminal*. Para garantizar la correcta compilación del proyecto, y el uso de la librería USB, se requieren tres archivos adicionales: *USBdsc.c*, *Definit.h*, y *VARs.h*. El archivo *USBdsc.c* contiene las definiciones con las que el ordenador reconoce el dispositivo HID. Tiene parámetros como el nombre del fabricante, el nombre del producto, los identificadores numéricos de fabricante y producto: VIP y PID. También contiene el tamaño de los búfer de entrada y

salida. El *HID Terminal* cuenta con una pestáña *descriptor*. En esta pestáña se editan las características del dispositivo HID, y luego se pulsa el botón *Save Descriptor*. Esta acción permite guardar el archivo que deben ser archivados en el directorio donde se guarda todo el proyecto, además debe ser anexado al proyecto por medio del siguiente ícono:  de la misma forma se deben pegar en el directorio los archivos: *Definit.h*, y *VARs.h*, estos no requieren ser anexados solo deben ser pegados. Estos archivos contienen el siguiente código:

Definit.h:

```
#defineEP0_PACKET_SIZE 8 // EP0 In & Out transfer size
#defineHID_PACKET_SIZE 64 // EP1 In & Out transfer size

//*****************************************************************************
//
// Definitions
//
//*****************************************************************************

#defineUSB_BUS_ATTACHED 1
#defineUSB_BUS_DETACHED 0

#defineUSB_DEVICE_DESCRIPTOR_LEN 0x12
#defineUSB_CONFIG_DESCRIPTOR_LEN 0x09
#defineUSB_INTERF_DESCRIPTOR_LEN 0x09
#defineUSB_ENDP_DESCRIPTOR_LEN 0x07
#defineUSB_HID_DESCRIPTOR_LEN 0x09

#defineUSB_DEVICE_DESCRIPTOR_TYPE 0x01
#defineUSB_CONFIG_DESCRIPTOR_TYPE 0x02
#defineUSB_STRING_DESCRIPTOR_TYPE 0x03
#defineUSB_INTERFACE_DESCRIPTOR_TYPE 0x04
#defineUSB_ENDPOINT_DESCRIPTOR_TYPE 0x05
#defineUSB_POWER_DESCRIPTOR_TYPE 0x06
#defineUSB_HID_DESCRIPTOR_TYPE 0x21

#defineUSB_ENDPOINT_TYPE_CONTROL 0x00
#defineUSB_ENDPOINT_TYPE_ISOCRONOUS 0x01
#defineUSB_ENDPOINT_TYPE_BULK 0x02
#defineUSB_ENDPOINT_TYPE_INTERRUPT 0x03

#defineDSC_DEV 0x01
#defineDSC_CFG 0x02
#defineDSC_STR 0x03
#defineDSC_INTF 0x04
#defineDSC_EP 0x05

//*****************************************************************************
```

```

// Definitions (added 19.06.2007)
//
//*****
#define ConfigDescr_OFFSET 0x24 // 0x12
#define HID_Descriptor_OFFSET 0x48 // 0x24
#define HID_ReportDesc_OFFSET 0x76 // 0x3B

#define USB_DEVICE_DESCRIPTOR_ALL_LEN 0x6A

//*****
// USBdrv.h
//*****
//      * UCFG Initialization Parameters
#define _PPBM0 0x00 // Pingpong Buffer Mode 0
#define _PPBM1 0x01 // Pingpong Buffer Mode 1
#define _PPBM2 0x02 // Pingpong Buffer Mode 2
#define _LS 0x00 // Use Low-Speed USB Mode
#define _FS 0x04 // Use Full-Speed USB Mode
#define _TRINT 0x00 // Use internal transceiver
#define _TREXT 0x08 // Use external transceiver
#define _PUEN 0x10 // Use internal pull-up resistor
#define _OEMON 0x40 // Use SIE output indicator
#define _UTEYE 0x80 // Use Eye-Pattern test

//      * UEPn Initialization Parameters
#define EP_CTRL 0x06 // Cfg Control pipe for this ep
#define EP_OUT 0x0C // Cfg OUT only pipe for this ep
#define EP_IN 0x0A // Cfg IN only pipe for this ep
#define EP_OUT_IN 0x0E // Cfg both OUT & IN pipes for this ep
#define HSHK_EN 0x10 // Enable handshake packet

#define OUT 0
#define IN 1

#define PIC_EP_NUM_MASK      0x78
#define PIC_EP_DIR_MASK 0x04

#define EP00_OUT 0x00    // (0x00<<3) | (OUT<<2)
#define EP00_IN   0x04    // (0x00<<3) | (IN<<2)
#define EP01_OUT 0x08    // (0x01<<3) | (OUT<<2)
#define EP01_IN   0x0C    // (0x01<<3) | (IN<<2)
#define EP02_OUT 0x10    // (0x02<<3) | (OUT<<2)
#define EP02_IN   0x14    // (0x02<<3) | (IN<<2)
#define EP03_OUT 0x18    // (0x03<<3) | (OUT<<2)
#define EP03_IN   0x1C    // (0x03<<3) | (IN<<2)

```

```

#defineEP04_OUT 0x20 // (0x04<<3) | (OUT<<2)
#defineEP04_IN 0x24 // (0x04<<3) | (IN<<2)
#defineEP05_OUT 0x28 // (0x05<<3) | (OUT<<2)
#defineEP05_IN 0x2C // (0x05<<3) | (IN<<2)
#defineEP06_OUT 0x30 // (0x06<<3) | (OUT<<2)
#defineEP06_IN 0x34 // (0x06<<3) | (IN<<2)
#defineEP07_OUT 0x38 // (0x07<<3) | (OUT<<2)
#defineEP07_IN 0x3C // (0x07<<3) | (IN<<2)
#defineEP08_OUT 0x40 // (0x08<<3) | (OUT<<2)
#defineEP08_IN 0x44 // (0x08<<3) | (IN<<2)
#defineEP09_OUT 0x48 // (0x09<<3) | (OUT<<2)
#defineEP09_IN 0x4C // (0x09<<3) | (IN<<2)
#defineEP10_OUT 0x50 // (0x0A<<3) | (OUT<<2)
#defineEP10_IN 0x54 // (0x0A<<3) | (IN<<2)
#defineEP11_OUT 0x58 // (0x0B<<3) | (OUT<<2)
#defineEP11_IN 0x5C // (0x0B<<3) | (IN<<2)
#defineEP12_OUT 0x60 // (0x0C<<3) | (OUT<<2)
#defineEP12_IN 0x64 // (0x0C<<3) | (IN<<2)
#defineEP13_OUT 0x68 // (0x0D<<3) | (OUT<<2)
#defineEP13_IN 0x6C // (0x0D<<3) | (IN<<2)
#defineEP14_OUT 0x70 // (0x0E<<3) | (OUT<<2)
#defineEP14_IN 0x74 // (0x0E<<3) | (IN<<2)
#defineEP15_OUT 0x78 // (0x0F<<3) | (OUT<<2)
#defineEP15_IN 0x7C // (0x0F<<3) | (IN<<2)
//*****
//  

// USBmmap.h  

//  

//*****
/* Buffer Descriptor Status Register Initialization Parameters  

#define_BC89 0x03 // Byte count bits 8 and 9  

#define_BSTALL 0x04 // Buffer Stall enable  

#define_DTSEN 0x08 // Data Toggle Synch enable  

#define_INCDIS 0x10 // Address increment disable  

#define_KEN 0x20 // SIE keeps buff descriptors enable  

#define_DAT0 0x00 // DATA0 packet expected next  

#define_DAT1 0x40 // DATA1 packet expected next  

#define_DTSMASK 0x40 // DTS Mask  

#define_USIE 0x80 // SIE owns buffer  

#define_UCPU 0x00 // CPU owns buffer  

// * USB Device States - To be used with [byte usb_device_state]  

#defineDETACHED_STATE 0  

#defineATTACHED_STATE 1  

#definePOWERED_STATE 2  

#defineDEFAULT_STATE 3  

#defineADR_PENDING_STATE 4  

#defineADDRESS_STATE 5  

#defineCONFIGURED_STATE 6  

// * Memory Types for Control Transfer - used in USB_DEVICE_STATUS

```

```

#define _RAM 0
#define _ROM 1
#defineRemoteWakeups 0
#definectrl_trf_mem 1
//*****************************************************************************
//
// USBctrlTrf.h
//
//*****************************************************************************
//      * Control Transfer States
#define WAIT_SETUP 0
#define CTRL_TRF_TX 1
#define CTRL_TRF_RX 2
//      * USB PID: Token Types - See chapter 8 in the USB specification
#define SETUP_TOKEN 0x0D
#define OUT_TOKEN 0x01
#define IN_TOKEN 0x09
//      * bmRequestType Definitions
#define HOST_TO_DEV 0
#define DEV_TO_HOST 1

#define STANDARD 0x00
#define CLASS 0x01
#define VENDOR 0x02

#define RCPT_DEV 0
#define RCPT_INTF 1
#define RCPT_EP 2
#define RCPT_OTH 3
//*****************************************************************************
//
// USBcfg.h
//
//*****************************************************************************
//      * MUID = Microchip USB Class ID
//      * Used to identify which of the USB classes owns
//          the current session of control transfer over EP0
#defineMUID_NULL 0
#defineMUID_USB9 1
#defineMUID_HID 2
//*****************************************************************************
//
// USB9.h
//
//*****************************************************************************
//      * Standard Request Codes, USB 2.0 Spec Ref Table 9-4
#defineGET_STATUS 0
#defineCLR_FEATURE 1
#defineSET_FEATURE 3

```

```

#defineSET_ADR      5
#defineGET_DSC      6
#defineSET_DSC      7
#defineGET_CFG      8
#defineSET_CFG      9
#defineGET_INTF 10
#defineSET_INTF 11
#defineSYNCH_FRAME 12
//      * Standard Feature Selectors
#define DEVICE_REMOTE_WAKEUP 0x01
#define ENDPOINT_HALT 0x00
//*********************************************************************
//
// HID.h
//
//*********************************************************************
#defineHID_INTF_ID 0x00
//      * Class-Specific Requests
#defineGET_REPORT 0x01
#defineGET_IDLE 0x02
#defineGET_PROTOCOL 0x03
#defineSET_REPORT 0x09
#defineSET_IDLE 0x0A
#defineSET_PROTOCOL 0x0B
//      * Class Descriptor Types
#defineDSC_HID     0x21
#defineDSC_RPT     0x22
#defineDSC_PHY     0x23
//*********************************************************************

```

VARs.h:

```

//*********************************************************************
//
// Bank 4 GPR Variables in region 0x400 - 0x4FF
//
//*********************************************************************
extern unsigned char BDT[16];
extern unsigned char SetupPkt[8];
extern unsigned char CtrlTrfData[8];
extern unsigned char hid_report_feature[8];

extern unsigned char Byte_tmp_0[2];
extern unsigned char Byte_tmp_1[2];
extern unsigned char param_Len;
extern unsigned char param_buffer[2];
extern unsigned char USTAT_latch;
extern unsigned char usb_device_state;
extern unsigned char usb_active_cfg;

```

```

extern unsigned char usb_alt_intf[2];
extern unsigned char usb_stat;
extern unsigned char idle_rate;
extern unsigned char active_protocol;
extern unsigned char hid_rpt_rx_len;
extern unsigned char ctrl_trf_state;
extern unsigned char ctrl_trf_session_owner;
extern unsigned char pSrc[2];
extern unsigned char pDst[2];
extern unsigned char wCount[2];
extern unsigned char byte_to_send[2];
extern unsigned char byte_to_read[2];
extern unsigned char number_of_bytes_read;

extern unsigned char USB_CD_Ptr[4];
extern unsigned char USB_SD_Ptr[8];

extern char FSR0reg[2];
extern char FSR1reg[2];
extern char FSR2reg[2];

extern unsigned int HID_ReadBuff_Ptr;
extern unsigned int HID_WriteBuff_Ptr;

extern unsigned char hid_report_out[64];
extern unsigned char hid_report_in[64];

//*****
// Setup packet structures in EP0 Out Buffer
//*****

extern unsigned char SetupPkt_bmRequestType;
extern unsigned char SetupPkt_bRequest;
extern unsigned int SetupPkt_wValue;
extern unsigned int SetupPkt_wIndex;
extern unsigned int SetupPkt_wLength;
extern unsigned int SetupPkt_W_Value;
extern unsigned int SetupPkt_W_Index;
extern unsigned int SetupPkt_W_Length;
extern unsigned char SetupPkt_Recipient;
extern unsigned char SetupPkt_RequestType;
extern unsigned char SetupPkt_DataDir;
extern unsigned char SetupPkt_bFeature;
extern unsigned char SetupPkt_bDscIndex;
extern unsigned char SetupPkt_bDscType;
extern unsigned int SetupPkt_wLangID;
extern unsigned char SetupPkt_bDevADR;
extern unsigned char SetupPkt_bDevADRH;
extern unsigned char SetupPkt_bCfgValue;
extern unsigned char SetupPkt_bCfgRSD;

```

```

extern unsigned char SetupPkt_bAltID;
extern unsigned char SetupPkt_bAltID_H;
extern unsigned char SetupPkt_bIntfID;
extern unsigned char SetupPkt_bIntfID_H;
extern unsigned char SetupPkt_bEPID;
extern unsigned char SetupPkt_bEPID_H;
extern unsigned char SetupPkt_EPNum;
extern unsigned char SetupPkt_EPDir;
//*****
// Buffer Descriptors Table
//*****
extern unsigned char BD0STAT;
extern unsigned char BD0CNT;
extern unsigned char BD0ADRL;
extern unsigned char BD0ADRH;
extern unsigned char BD1STAT;
extern unsigned char BD1CNT;
extern unsigned char BD1ADRL;
extern unsigned char BD1ADRH;
extern unsigned char BD2STAT;
extern unsigned char BD2CNT;
extern unsigned char BD2ADRL;
extern unsigned char BD2ADRH;
extern unsigned char BD3STAT;
extern unsigned char BD3CNT;
extern unsigned char BD3ADRL;
extern unsigned char BD3ADRH;
extern unsigned char BD4STAT;
extern unsigned char BD4CNT;
extern unsigned char BD4ADRL;
extern unsigned char BD4ADRH;
extern unsigned char BD5STAT;
extern unsigned char BD5CNT;
extern unsigned char BD5ADRL;
extern unsigned char BD5ADRH;
extern unsigned char BD6STAT;
extern unsigned char BD6CNT;
extern unsigned char BD6ADRL;
extern unsigned char BD6ADRH;
extern unsigned char BD7STAT;
extern unsigned char BD7CNT;
extern unsigned char BD7ADRL;
extern unsigned char BD7ADRH;
//*****
//*****
// Initialization Function
//*****
void InitVARs();
//*****

```

Para la creación del proyecto el editor HID Terminal debe tener una vista como la siguiente antes de realizar la creación del archivo USBdsc.c:

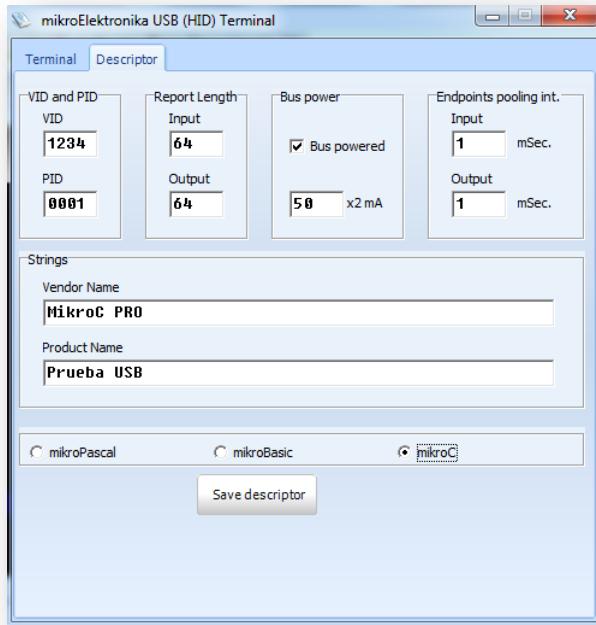


Figura 7-8

Después de correr la simulación se debe correr el *HID Terminal*, y seleccionar la pestáña: *Terminal*. Esta aplicación permite enviar y recibir datos por medio del puerto USB. Para este propósito, se debe seleccionar el dispositivo, Picando el nombre del dispositivo en la lista *HID Devices*, para este caso el nombre del dispositivo es: *Prueba USB*. Posteriormente se escribe un carácter en la casilla de envío de datos y se pulsa el botón *Send*. Seguidamente el PIC retransmite un texto con el carácter enviado, y el LED del circuito hace un parpadeo. Una vista de esta acción en el *HID Terminal* tiene la siguiente apariencia:

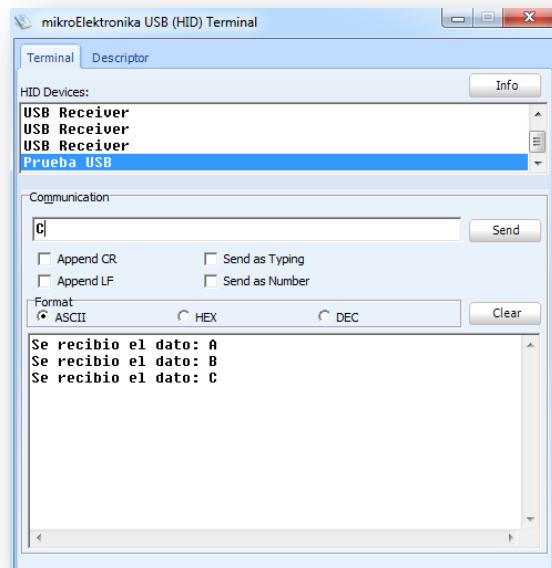


Figura 7-9

De igual manera si se pulsa el botón: *info*, el *HID Terminal* muestra en una ventana las características con las que está configurado en dispositivo HID, una vista de esta acción se puede apreciar en la siguiente figura:

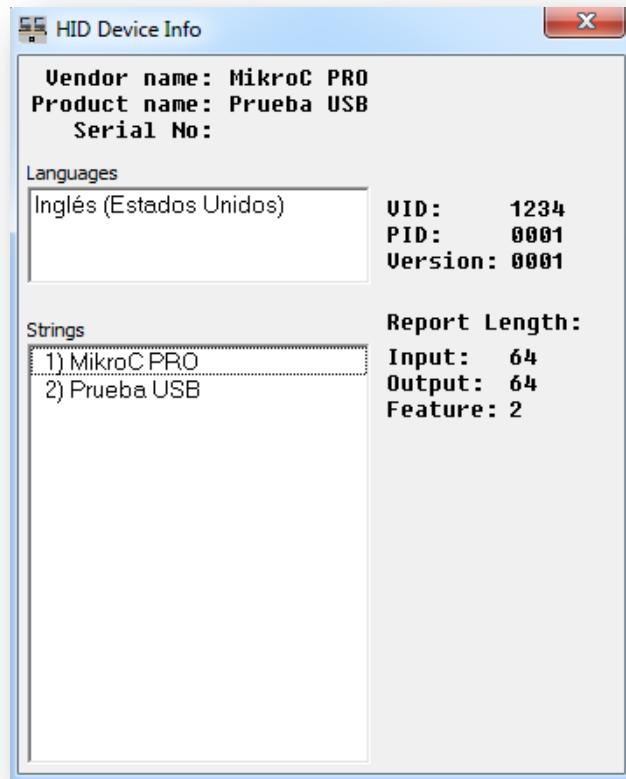


Figura 7-10

8 Conversión AD y DA

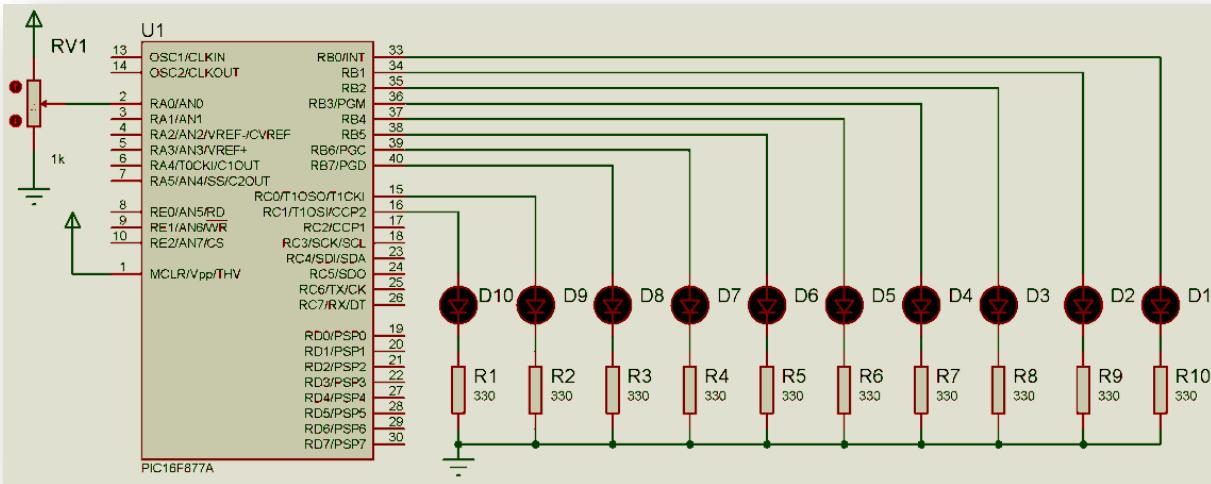
La conversión análogo digital y digital análogo es un proceso por el cual se puede tomar o entregar muestras de una señal continua de voltaje. El uso de estas conversiones es de gran utilidad para realizar procesamiento digital de señales. La conversión análogo digital, o ADC, se puede realizar con algunos microcontroladores que tienen implícito un convertidor de este estilo. El proceso de conversión digital análogo es posible con elementos externos de fácil implementación o, incluso, es posible realizar esta conversión con los módulos PWM incorporados en algunos microcontroladores.

8.1 Conversión AD, o ADC

Este proceso se realiza con el convertidor interno de los microcontroladores. Este módulo está incorporado en la mayoría de los microcontroladores de gama media y alta. La conversión implementada por los PICMicro cuenta con una resolución de 10 bits, lo que permite obtener un número con un rango de 0 a 1023, que es proporcional a los valores de referencia, que por defecto son 0 voltios y 5 voltios. Esto significa que si una entrada análoga, tiene una tensión de 0 voltios su resultado es 0, y si la tensión es de 5 voltios el resultado de la conversión es 1023 de igual manera si la tensión es de 2.5 voltios, el resultado será 512. Dependiendo de la complejidad del microcontrolador un PIC puede tener hasta 8 entradas de señal análoga. Sin embargo cabe denotar que el módulo de conversión interna de los microcontroladores es sólo uno, y los múltiples canales se pueden leer pero no al mismo tiempo. Para realizar este tipo de conversiones el compilador MikroC PRO cuenta con una librería definida: *ADC* para hacer la conversión. Esta librería cuenta con una sola función denominada *unsigned int ADC_Read(unsigned short channel)*. Esta función retorna el resultado de la conversión del canal especificado por el parámetro *channel*. Para contextualizar el uso de esta librería se puede observar y analizar el siguiente ejemplo:

```
void main( void )
{
    //Declaración de variables.
    unsigned int Dato;
    //Inicialización de puertos.
    TRISB = 0;
    TRISC = 0;
    PORTB = 0;
    PORTC = 0;
    while(1) //Bucle infinito.
    {
        Dato = ADC_Read(0); //Se hace conversión sobre el canal 0.
        //Se muestran los 8 bits de menor peso por el puerto B.
        PORTB = Dato&0xFF;
        //Se muestran los 2 bits de mayor peso por el puerto C.
        PORTC = (Dato>>8)&0x03;
    }
}
```

Para la simulación de este ejemplo se deben implementar los dispositivos: 16F877A, RES, LED-RED, POT-HG, en el siguiente circuito de ISIS:



Circuito 8-1

Después de correr la simulación los LEDs muestran en código binario del valor de la conversión análogo digital, de la diferencia de potencial que entrega el potenciómetro.

8.2 Conversión DA o DAC

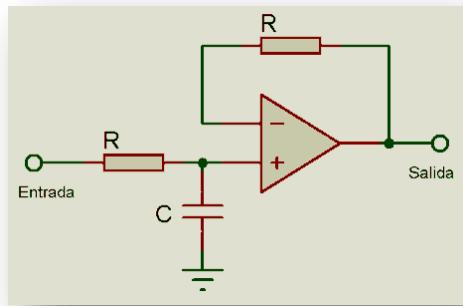
Este tipo de conversión es posible por medio de dos estrategias, la primera es usar el módulo PWM, del microcontrolador, y la segunda es la implementación de un arreglo externo de resistencias para obtener la diferencia de potencial.

8.2.1 Conversión DA con PWM

La conversión digital análogo, con el módulo PWM, consiste en tomar la señal modulada por ancho de pulso y realizar la demodulación por medio de un filtro pasa bajas. Este filtro debe tener como frecuencia de corte un valor mucho menor a la frecuencia de muestreo en una proporción cercana a 10 veces, dado que su objetivo es eliminar la frecuencia de muestreo. El cálculo de la frecuencia de corte de este filtro está regido por la siguiente fórmula:

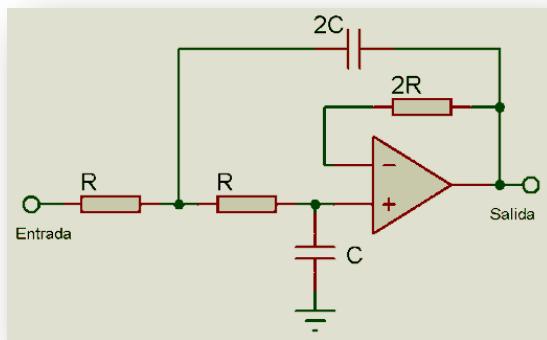
$$F_c = \frac{1}{2\pi RC} \quad \text{Ecuación 8-1}$$

El circuito a implementar de un filtro pasa bajas de primer orden es el siguiente:



Circuito 8-2

Implementación del filtro pasa bajas de segundo orden:



Circuito 8-3

La ecuación para el cálculo en este filtro es:

$$Fc = \frac{1}{2\pi RC\sqrt{2}} \quad Ecuación 8-2$$

La frecuencia que se asigne a la portadora de modulación PWM, debe ser mucho mayor a la frecuencia de muestreo de la señal moduladora. Para demostrar el funcionamiento en esta conversión se designarán 20 muestras que corresponden a un ciclo de una señal seno. Las muestras y la forma de onda se pueden apreciar en la siguiente figura:

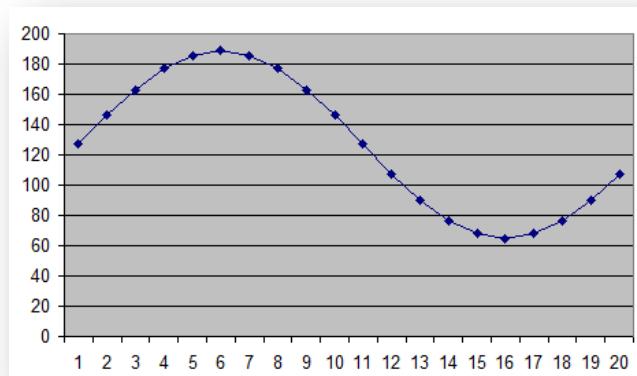


Figura 8-1

La implementación de la librería PWM, se hace por medio de cuatro funciones que son: *PWM1_Init(const long freq);*, está función inicializa el módulo PWM, a la frecuencia de la portadora *freq*. La función: *PWM1_Set_Duty(unsigned short duty_ratio);*, está función establece el ciclo útil por medio del parámetro *duty_ratio*, este parámetro puede tomar valores de 0 a 255, donde 0 representa el 0%, y 255 el 100% del ciclo útil. La siguiente imagen ilustra el comportamiento de una señal PWM, en función del ciclo útil, la relación *Fpwm* y *Tpwm*:

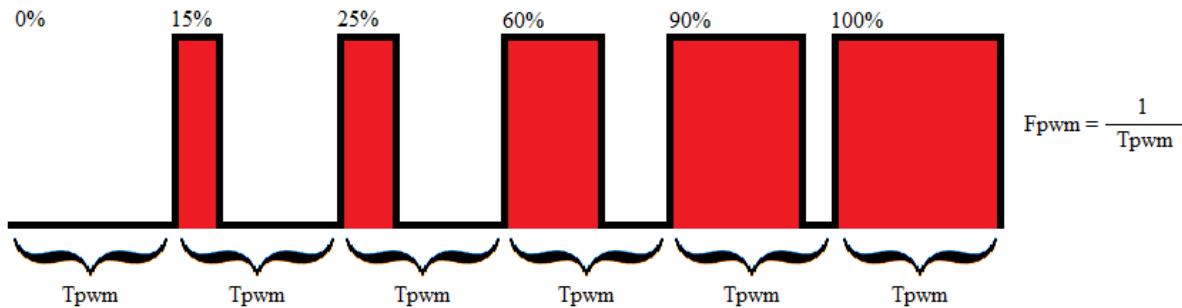


Figura 8-2

Por último se cuenta con las funciones: *PWM1_Start();* y *PWM1_Stop();*, Las cuales activan y desactivan respectivamente la señal PWM.

Para realizar la simulación se debe hacer un proyecto en MikroC PRO, con el siguiente código fuente:

```
//Declaración de constantes
//para la señal seno.
const unsigned short Seno[20] =
{
    127, 146, 163, 177, 185, 189, 185,
    177, 163, 146, 127, 107, 90, 76,
    68, 65, 68, 76, 90, 107
};

void main( void )
{
    //Declaración de variables.
    unsigned short n=0;
    //Configuración del módulo PWM a Fpwm=15.625K Hz.
    PWM1_Init(15625);
    //Inicio de señal PWM.
    PWM1_Start();
    while(1) //Bucle infinito.
    {
        //Bucle para recorres las 20 muestras
        //de un ciclo para la onda seno.
        for( n=0; n<20; n++ )
```

```

    {
        //Cambio del ciclo útil del PWM.
        PWM1_Set_Duty( Seno[n] );
        //Retardo de 50u seg.
        delay_us(50);
    }
}

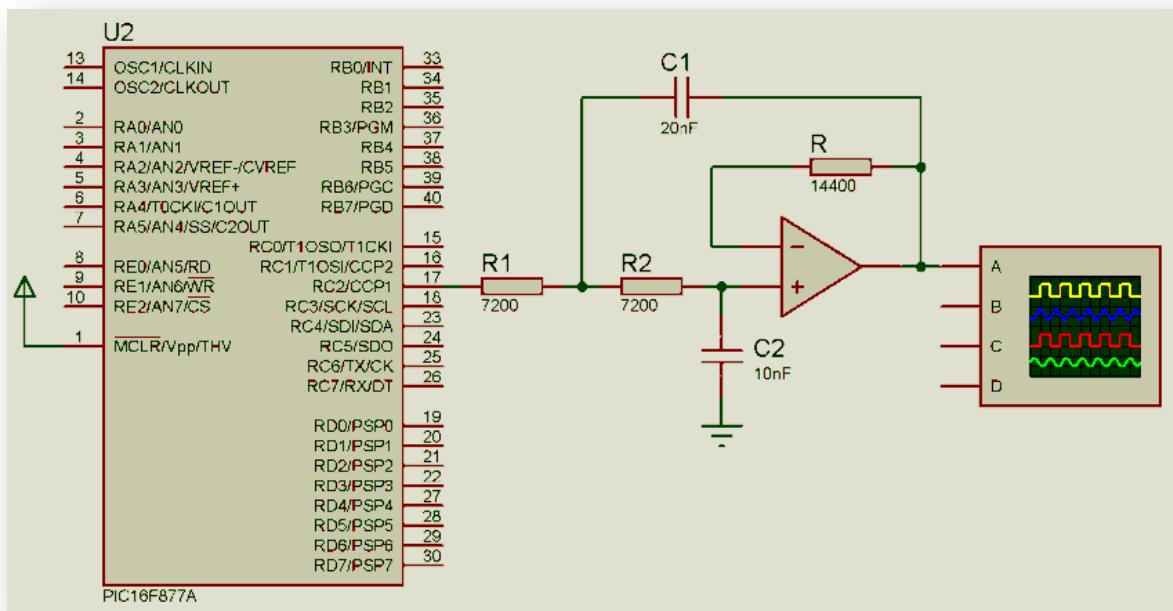
```

La ejecución de la simulación requiere de los dispositivos: 16F877A, RES, CAP, OP1P, y el instrumento virtual: OSCILLOSCOPE. Este último es un osciloscopio virtual de 4 canales simultáneos, ideal para visualizar señales análogas e incluso digitales.

Este circuito implementa un filtro pasa bajas para eliminar la frecuencia F_{pwm} portadora que en este caso es de 15.6KHz, el filtro cuenta con una frecuencia de corte de 1.5KHz aproximadamente.

En futuras aplicaciones se recomienda para los cálculos de los filtros, definir un valor arbitrario para el condensador C, entre 100nF y 100pF y calcular el valor de R, por medio de las ecuaciones antes mencionadas.

Con los anteriores elementos se construye el siguiente circuito:



Circuito 8-4

Sobre la marcha de la simulación se debe apreciar como emerge el osciloscopio virtual mostrando la señal análoga de forma seno producto de la reconstrucción digital del PIC, por medio del módulo PWM, y el filtro pasa bajas, la apariencia visual del osciloscopio es la que se puede ver en la siguiente figura:

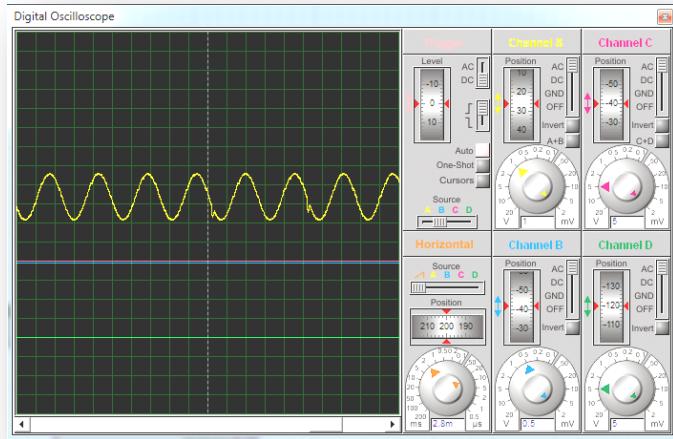
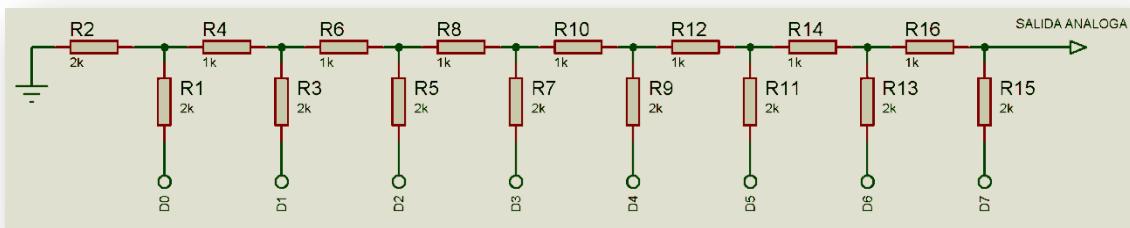


Figura 8-3

8.2.2 Conversión DA con arreglo R-2R

El arreglo de resistencias R-2R, permite hacer la conversión de un número binario a un valor proporcional de voltaje. Este arreglo permite implementar un número indeterminado de bits, en la conversión a diferencia del módulo PWM, que se limita a 8 bits. Esto quiere decir que con el arreglo R-2R, se pueden hacer conversiones de 8, 16, 32, 64, o un número n de bits en función de la cantidad de pines disponibles en un microcontrolador. La implementación del convertidor R-2R, es de fácil desarrollo dado que consiste en un arreglo de conexiones de resistencias, donde una es el doble de la otra, de ahí su nombre R-2R.

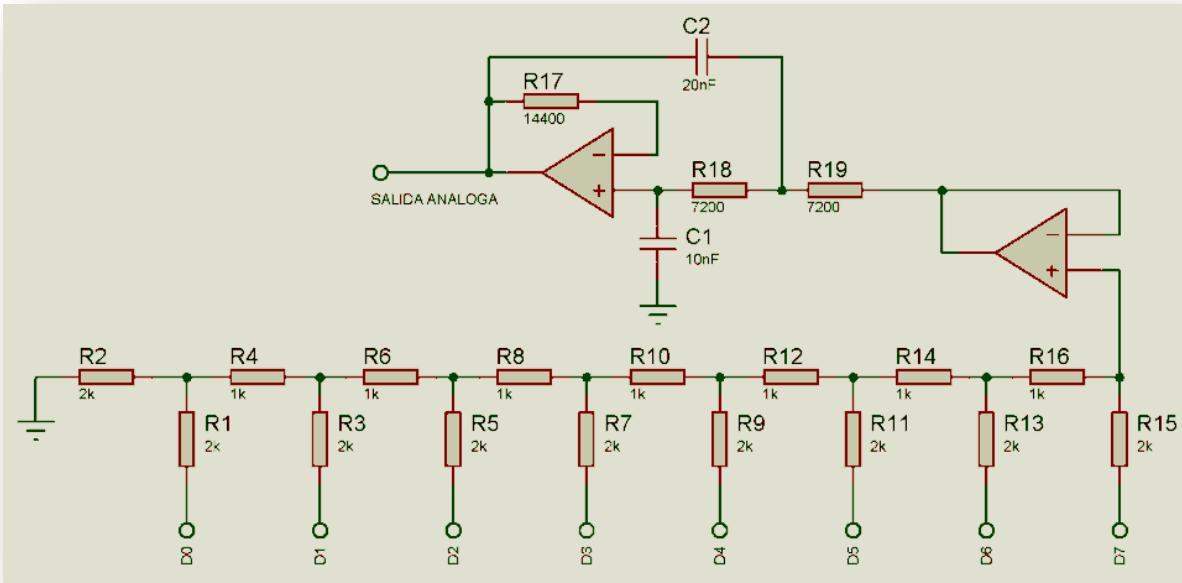
Cabe resaltar que cuantos más bits, posea la conversión mayor será su resolución y por consiguiente mayor la calidad de la señal reconstruida. Las desventajas notables de este arreglo son, el incremento del hardware requerido y el uso de una cantidad mayor de pines de los puertos. En la siguiente gráfica se puede apreciar la configuración de las resistencias para una conversión de 8 bits:



Circuito 8-5

Este arreglo puede ser expandido con la misma arquitectura para conseguir hacer un convertidor de mayor resolución, aumentando el número de entradas D, o lo que es igual el número de bits. Otra característica de este tipo de conversión es que no requiere de librerías especializadas, solo basta con colocar el valor numérico a convertir en un puerto, y el valor análogo hará presencia en la salida. De la misma forma que se hace la conversión por PWM, es importante suprimir la frecuencia de muestreo para evitar componentes de señal impuras.

Una forma completa del convertidor con un acople de impedancia y su respectivo filtro pasa bajas de 1.5K Hz, es el siguiente:



Circuito 8-6

Para demostrar la aplicación de esta técnica de conversión se modificará el ejemplo de conversión con PWM, para este fin observe y analice el siguiente código fuente:

```
//Declaración de constantes
//para la señal seno.
const unsigned short Seno[20] =
{
    127, 146, 163, 177, 185, 189, 185,
    177, 163, 146, 127, 107, 90, 76,
    68, 65, 68, 76, 90, 107
};

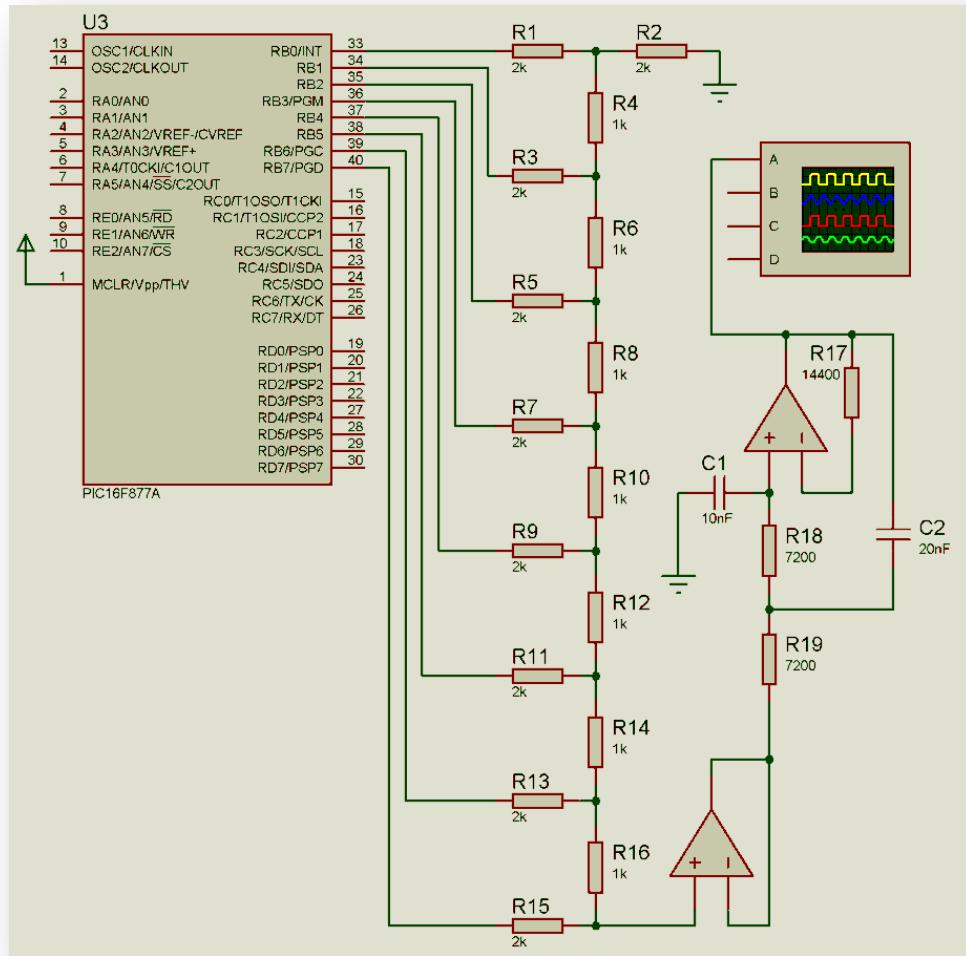
void main( void )
{
    //Declaración de variables.
    unsigned short n=0;
    //Configuración de puertos.
    TRISB = 0;
    PORTB = 127;
    while(1) //Bucle infinito.
    {
        //Bucle para recorres las 20 muestras
        //de un ciclo para la onda seno.
        for( n=0; n<20; n++ )
        {
            //Cambio de muestra en el puerto B.
```

```

PORTB = Seno[n];
//Retardo de 50u seg.
delay_us(50);
}
}
}

```

Editado y compilado este programa se procede a simular en ISIS, con el circuito que se puede apreciar en el siguiente circuito:



Circuito 8-7

El resultado esperado en esta simulación es igual a la simulación del convertidor con PWM, la salida del osciloscopio debe mostrar una gráfica similar.

9 Memorias EEPROM y FLASH

Las memorias EEPROM y FLASH de los microcontroladores son espacios de memoria que se pueden programar y borrar eléctricamente, estas dos memorias son campos de información no volátil; es decir, que si la energía de alimentación del microcontrolador se desconecta la información no se pierde. Esta información puede ser reprogramada alrededor de 100000 veces para el caso de la memoria FLASH y algo cercano a 1000000 de veces para el caso de las memorias EEPROM. De igual manera su información puede ser garantizada hasta por 40 años sin energía eléctrica.

9.1 Memoria EEPROM

La memoria EEPROM es un campo relativamente pequeño que tienen los microcontroladores para que el desarrollador guarde información como configuraciones y datos variables que no deben perderse. La capacidad de memoria varía de un microcontrolador a otro dependiendo de su gama y tamaño. La manipulación de esta memoria se puede hacer con MikroC PRO, por medio de una librería llamada: *EEPROM*, esta cuenta con dos funciones que permiten leer y escribir una dirección de memoria. Las funciones son: ***unsigned short EEPROM_Read(unsigned int address);***, Esta función lee y retorna el valor guardado en la dirección especificada en el parámetro *address*.

La función: ***EEPROM_Write(unsigned int address, unsigned short data);***, esta función graba la dirección definida en el parámetro *address*, con el valor entregado en el parámetro *data*.

Para entender de forma concreta el funcionamiento de esta librería se puede observar y analizar el siguiente código fuente:

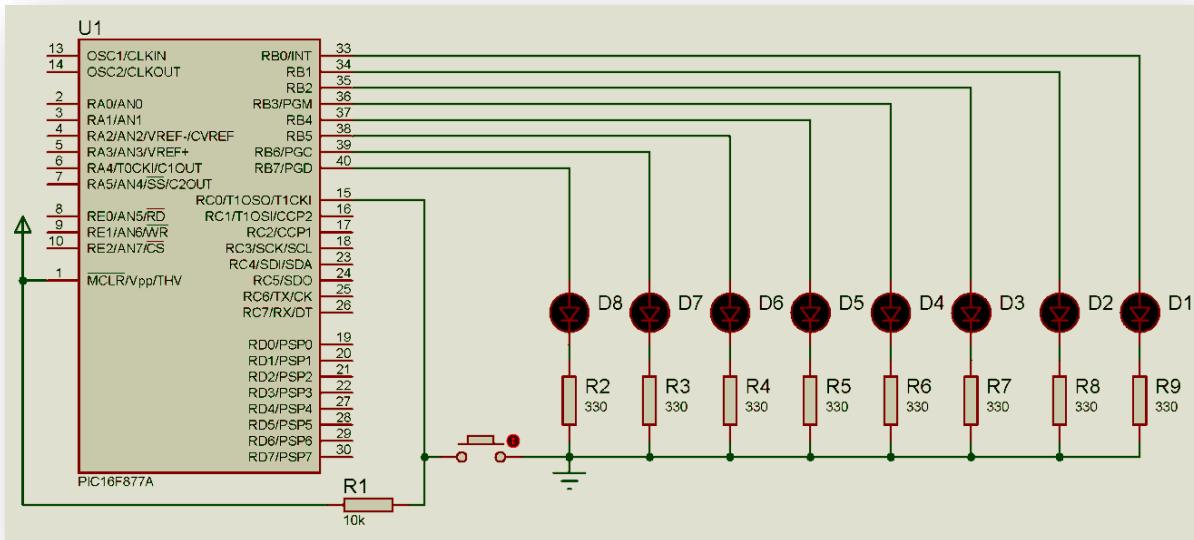
```
void main( void )
{
    //Configuración de puertos.
    TRISB = 0;
    PORTB = 0;
    while(1) //Bucle infinito.
    {
        //Se muestra en el puerto B el valor
        //de la dirección 0 de la memoria EEPROM.
        PORTB = EEPROM_Read(0);
        //La sentencia if evalúa si se pulsa el botón.
        if( !PORTC.F0 )
        {
            //Se incrementa el valor del puerto B.
            PORTB++;
            //Se guarda el nuevo valor del puerto
            //en la dirección 0 de la EEPROM.
            EEPROM_Write(0,PORTB);
```

```

//Se espera a que se suelte el botón.
while( !PORTC.F0 );
}
}
}

```

La simulación de este ejemplo se realiza con los dispositivos: 16F877A, RES, LED-RED, y BUTTON. Para correr la simulación se debe construir el siguiente circuito en ISIS:



Circuito 9-1

El circuito al ser simulado, mostrará en los LEDs el valor de la dirección 0 de la memoria EEPROM, este valor es incrementado cada vez que se pulsa el botón, y este es actualizado en la memoria EEPROM. Durante la simulación se puede detener la misma, para ver el efecto de la suspensión de energía, de tal forma que al correr de nuevo la simulación se puede ver como el valor de la información se conserva.

9.2 Memoria FLASH

La memoria FLASH es el campo de información de mayor capacidad de los microcontroladores es en esta memoria donde se guarda las instrucciones de todo el programa del PIC. Esto significa que es de sumo cuidado usar esta memoria ya que es posible dañar el programa principal si se altera alguna de sus instrucciones. La programación de la memoria FLASH es de utilidad para guardar segmentos de información de tamaño relativamente grande. En esta memoria se puede guardar pequeños archivos del orden de 1 o más kilo bytes dependiendo de la capacidad de memoria de cada uno de los microcontroladores, y del tamaño total del programa, dado que a medida que el tamaño del programa aumenta, la posibilidad de usar la memoria FLASH, se reduce.

La implementación de esta librería se fundamenta en dos funciones similares a las funciones de la memoria EEPROM, estas funciones son: **FLASH_Write(unsigned address, unsigned int* data);**, en esta función se ingresa el parámetro *address*, el cual contiene la dirección de la primera

dirección a grabar, y el apuntador *data*, que contiene los datos que serán grabados en la memoria FLASH. La siguiente función es: ***unsigned FLASH_Read(unsigned address);***, está función retorna el valor guardado en la dirección definida por el parámetro *address*.

Para contextualizar el uso de esta librería se puede crear y analizar el siguiente código de programa:

```
void main( void )
{
    //Declaración de variables.
    char Dato;
    unsigned int buf[10];
    unsigned int Dir;
    //Configuración de puertos.
    TRISB = 0;
    PORTB = 0;
    PORTB = 1;
    //Configuración del puerto serial.
    UART1_Init(9600);
    //Bienvenida al programa.
    UART1_Write_Text("Digite su archivo FLASH");
    UART1_Write(13);
    UART1_Write(10);
    //Se configura la dirección inicial de la FLASH.
    Dir=0x0200;
    while(1) //Bucle infinito.
    {
        //Sentencia if para evaluar si hay datos para leer
        //en el puerto serial.
        if( UART1_Data_Ready() )
        {
            //Se lee el puerto serial.
            Dato = UART1_Read();
            //Estructura Switch case para evaluar los datos de entrada.
            switch( Dato )
            {
                //Caso de solicitud de envío del archivo
                //con el carácter #
                case '#': UART1_Write(13);
                            UART1_Write(10);
                            UART1_Write_Text("INIDIO DEL ARCHIVO:");
                            UART1_Write(13);
                            UART1_Write(10);
                //Se configura la dirección inicial de la FLASH.
                Dir=0x0200;
                //Bucle do while para leer el archivo
                //y enviar la información.
                do
                {
```

```

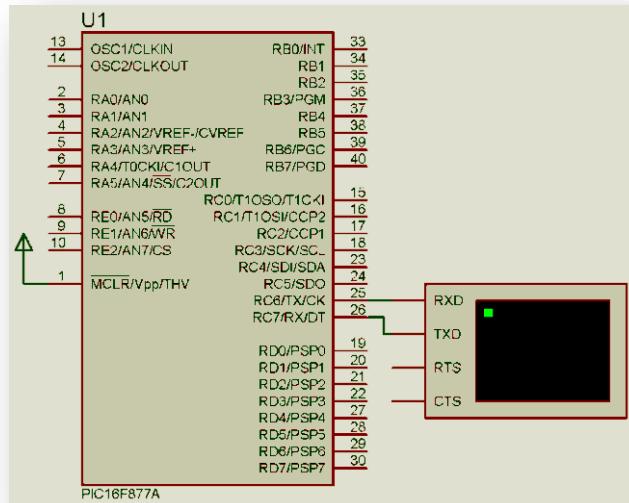
    //Lectura de la dirección en FLASH.
    buf[0]=FLASH_Read(Dir++);
    Dato = buf[0];
    //Envío del Dato por el puerto serial.
    UART1_Write(Dato);
}while( Dato!=0 );
UART1_Write(13);
UART1_Write(10);
UART1_Write_Text("FIN DEL ARCHIVO.");
UART1_Write(13);
UART1_Write(10);
//Se configura la dirección inicial de la FLASH.
Dir=0x0200;
break;

default: //Caso por defecto en donde se guardan
//los datos en la FLASH.
Buf[0]=Dato;
Buf[1]=0; //Carácter fin de cadena NULL.
//Envío de eco por el puerto serial.
UART1_Write(Dato);
//Se guarda en FLASH el dato.
FLASH_Write( Dir, Buf );
//Incremento de la dirección.
Dir++;
}

}
}
}
}

```

Para la simulación en ISIS, se debe implementar los dispositivos 16F877A, y el VIRTUAL TERMINAL, en un circuito como el que se puede apreciar en la siguiente figura:



Circuito 9-2

10 Módulos Timer

Los módulos Timer son unidades que permiten hacer el conteo de tiempos precisos, la cantidad de módulos Timer en cada microcontrolador depende de la gama del PIC, algunos pueden tener hasta 4 Timers, sin embargo la mayoría de los microcontroladores posee al menos uno que generalmente se denomina Timer 0. El uso y configuración de cada módulo Timer depende del desarrollador y del tamaño del conteo, este conteo puede ser de 8 o 16 bits. El intervalo de tiempo con el cual se incrementa el Timer, depende de la arquitectura y configuración que se asigne en los registros de control del Timer. Para el caso del PIC 16F877A, la arquitectura del Timer 0 es como se puede apreciar en la siguiente figura:

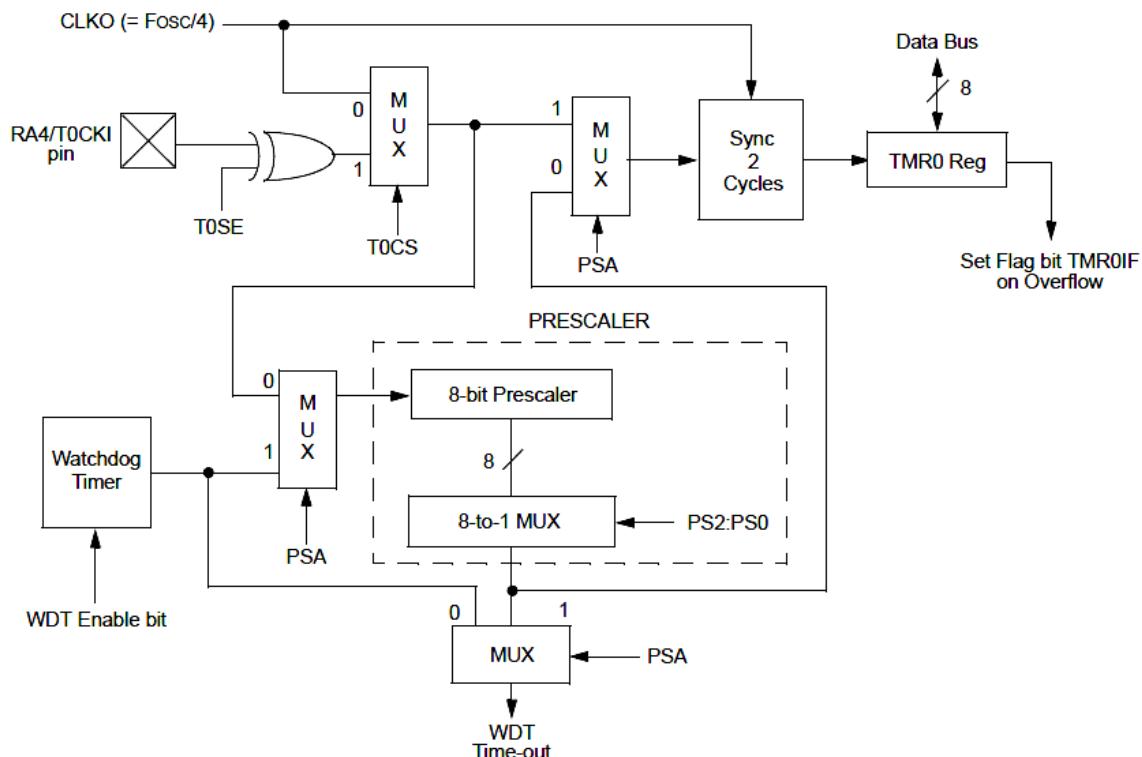


Figura 10-1

Este módulo Timer, puede tener dos fuentes de pulsos de reloj, uno es el reloj propio del procesador, o una fuente externa de reloj, por medio del pin RA4. De igual manera la señal de reloj puede usar una subdivisión, para crear tiempo de reloj menor. Las subdivisiones posibles para este timer son las siguientes: 1, 2, 4, 8, 16, 32, 64, 128, y 256. Para el caso de el Timer 0 su registro es de 8 bits, esto quiere decir que puede hacer un conteo de 0 a 255, y cuando el timer pasa de 255 a 0, se dice que el Timer se desborda o se dispara.

En función de estas características el tiempo de desborde o disparo se puede calcular con la siguiente relación: $P = 256x(\text{Subdivisión})x4xTosc$, donde Tosc es el periodo de oscilación del reloj del procesador, si el Timer 0 se configura con la fuente externa la relación es:

$P = 256x(\text{Subdivisión})xText$, donde Text, es el periodo de oscilación del reloj externo.

Para un reloj de 4MHz, el mayor tiempo puede ser: $P=256x256x4x1\mu s$, $P=65,536m Seg.$

La configuración del Timer 0 se logra con los bits: T0CS, PSA, PS0, PS1, y PS2. La calibración de la subdivisión del Timer 0 se hace con los bits, PS0, PS1, y PS2, está configuración obedece a la siguiente relación:

| Subdivisión Timer 0 | |
|---------------------|-------------|
| PS2:PS1:PS0 | Subdivisión |
| 000 | 2 |
| 001 | 4 |
| 010 | 8 |
| 011 | 16 |
| 100 | 32 |
| 101 | 64 |
| 110 | 128 |
| 111 | 256 |

Tabla 10-1

Para seleccionar la subdivisión 1, se debe configura el bit PSA, el cual asumen la subdivisión 1, cuando este bit vale 1, y a sume la subdivisión de la tabla anterior, cuando este bit vale 0.

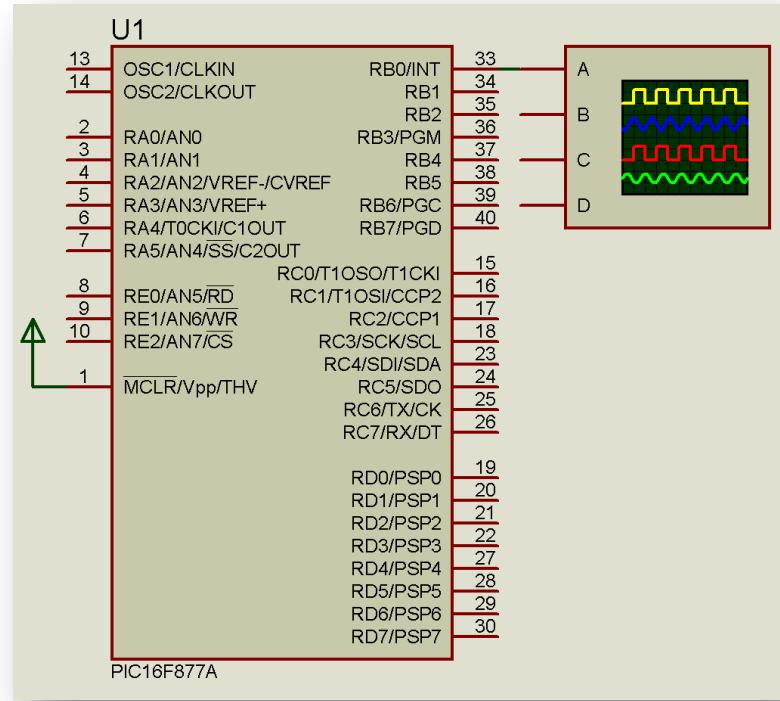
La configuración de los bits PS0 al PS2, se realiza sobre el registro *OPTION_REG*, en los tres bits de menor peso, el bit PSA, corresponde al cuarto bit del registro *OPTION_REG*, y el bit T0SC corresponde al sexto bit del mismo registro. El mapa de bits del registro *OPTION_REG*, es el siguiente:

| OPTION_REG | | | | | | | |
|------------|--------|-------|-------|-------|-------|-------|-------|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| RBPU | INTEDG | T0CS | TOSE | PSA | PS2 | PS1 | PS0 |

Para demostrar el uso del Timer 0, se puede observar y analizar el siguiente código fuente:

```
void main( void )
{
    TRISB = 0; //Configuración de puertos.
    PORTB = 0;
    OPTION_REG=0b11000111; //Configuración del Timer 0.
    while(1) //Bucle infinito.
    {
        if(TMR0==0) //Se evalúa si el Timer 0 vale 0.
        {
            if(PORTB==0)//Se conmuta el valor del puerto B.
                PORTB=1;
            else
                PORTB=0;
            while(TMR0==0); //Se espera a que el Timer 0 cambie de valor.
        }
    }
}
```

Para la simulación de este circuito en ISIS, se implementan los dispositivos 16F877A, y el osciloscopio virtual, OSCILLOSCOPE, en el siguiente arreglo:



Circuito 10-1

Al correr la simulación se puede apreciar en el osciloscopio, que hay un cambio de estado en el pin RB0, cada 65.536m Seg. La vista del osciloscopio debe ser la siguiente:

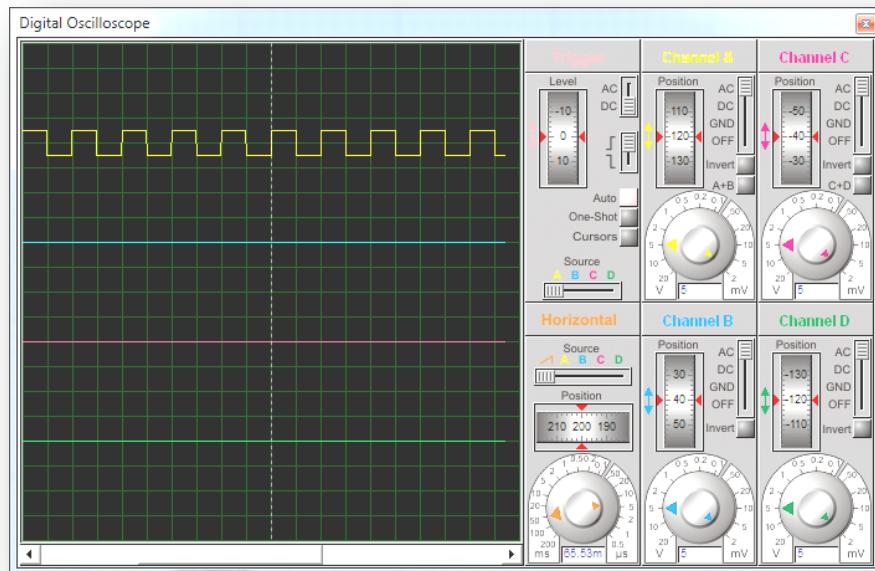


Figura 10-2

11 Interrupciones

Las interrupciones en los microcontroladores son eventos programados que hacen que el PIC suspenda su rutina y ejecute un fragmento de programa asociado al evento. Cuando la rutina de la interrupción se termina el programa del PIC continúa ejecutando el flujo del programa en el mismo punto en donde se suspendió la ejecución. Las interrupciones pueden tener fuentes variadas que dependen de la gama y complejidad del microcontrolador. Por ejemplo el PIC 16F877A cuenta con fuentes de interrupción como el Timer0, cuando el timer se desborda, externa cuando se hace un cambio de flanco sobre el pin RB0, serial cuando un dato llega por la USART, ADC, cuando una conversión termina, entre otras. Para la ejecución de las interrupciones el compilador MikroC PRO, cuenta con una función predefinida llamada: **void interrupt(void);** esta función debe ser declarada antes de la función **main**. Cuando una interrupción se dispara la función *interrupt* es invocada automáticamente por el programa, dentro de esta función se debe evaluar cual de las interrupciones se dispara por medio de las banderas de cada interrupción.

Para la configuración de las interrupciones se deben activar los bits correspondientes en los registros de interrupción En el caso particular del PIC 16F877A se usan los registros: INTCON, PIR1, PIR2, PIE1, y PIE2. Los mapas de bits de estos registros en el PIC 16F877A son los siguientes:

| INTCON | | | | | | | |
|--------|-------|--------|-------|-------|--------|-------|-------|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| GIE | PEIE | TMR0IE | INTE | RBIE | TMR0IF | INTF | RBIF |

| PIR1 | | | | | | | |
|-------|-------|-------|-------|-------|--------|--------|--------|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| PSPIF | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF |

| PIR2 | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|--------|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| - | CMIF | - | EEIF | BCLIF | - | - | CCP2IF |

| PIE1 | | | | | | | |
|-------|-------|-------|-------|-------|--------|--------|--------|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| PSPIE | ADIE | RCIE | TXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE |

| PIE2 | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|--------|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| - | CMIE | - | EEIE | BCLIE | - | - | CCP2IE |

La implementación de las interrupciones implica seguir un protocolo para la correcta configuración de las mismas. Primero se deben activar las interrupciones que se desean usar, por medio de los bit IE, como: TMR0IE, INTE, RCIE, entre otros. Las banderas de disparo de las interrupciones debe ser desactivadas, como: TMR0IF, INTF, RCIF, entre otras. Se deben activar las interrupciones periféricas, cuando éstas se requieran, por ejemplo las que están citadas en el registro: PIE1, y PIE2. Por último se debe activar el bit de interrupciones en general, que es: GIE. Para el siguiente ejemplo se usarán las interrupciones por: recepción serial, Timer 0, y externa, para este fin se puede observar y analizar el siguiente código fuente:

```
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    //Declaración de variables usadas en
    //la función de interrupciones.
    char Dato;
    //Se evalúa si la interrupción disparada es
    //por Timer 0, TMR0IF
    if( INTCON.F2==1 )
    {
        //Se complementa el valor del bit RB1.
        if(PORTB.F1==1)
            PORTB.F1=0;
        else
            PORTB.F1=1;
        //Se apaga la bandera de Timer 0.
        INTCON.F2=0;
    }

    //Se evalúa si la interrupción disparada es externa.
    // INTF
    if( INTCON.F1==1 )
    {
        //Se complementa el valor del bit RB2.
        if(PORTB.F2==1)
            PORTB.F2=0;
        else
            PORTB.F2=1;
        //Se apaga la bandera de interrupción externa.
        INTCON.F1=0;
    }

    //Se evalúa si la interrupción disparada por
    //recepcción serial.
    if( PIR1.F5 )
    {
        //Se lee el dato de entrada.
    }
}
```

```

Dato = UART1_Read();
//Se envía información de confirmación.
UART1_Write_Text("Dato de entrada: ");
//Se envía el dato recibido.
UART1_Write(Dato);
UART1_Write(13); //Se envía código ASCII del enter.
UART1_Write(10); //Se envía código de retroceso del carro.
//Se apaga la bandera por recepción serial.
PIR1.F5 = 0;
}
}

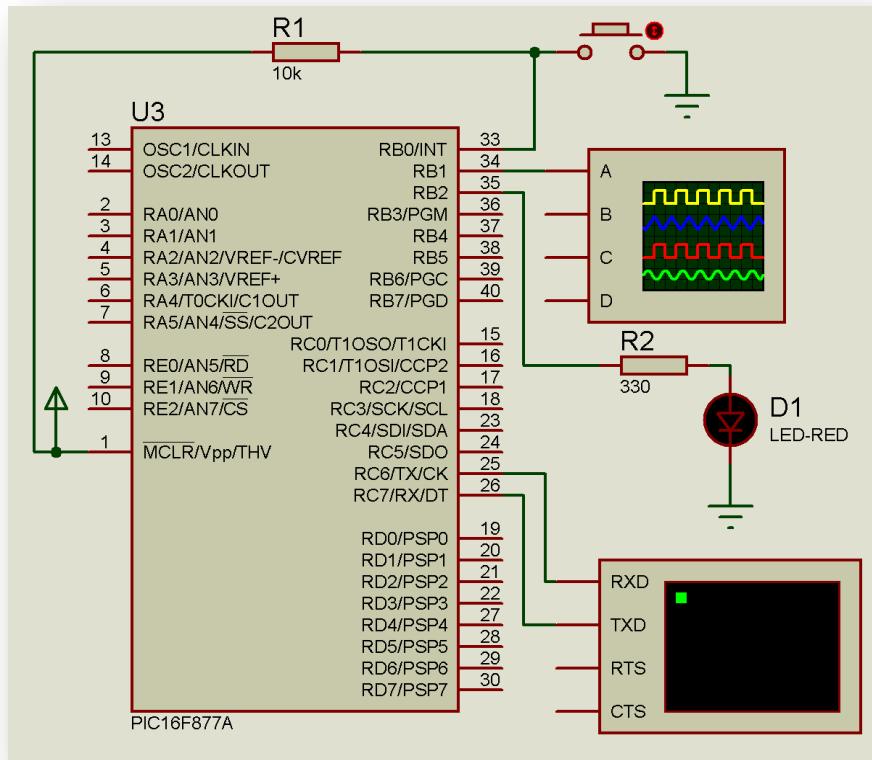
```

```

void main( void )
{
    //Se configuran los puertos.
    TRISB = 0b00000001;
    PORTB = 0;
    //Se activa la interrupción por
    //recepción serial.
    PIE1 = 0b00100000;
    //Se desactivan las demás fuentes de interrupción.
    PIE2 = 0;
    //Se apagan las banderas de interrupción.
    PIR2 = 0;
    PIR1 = 0;
    //Configuración del Timer 0 a 65,535m Seg.
    OPTION_REG=0b11000111;
    //Configuración del puerto serial a 9600 bps.
    UART1_Init(9600);
    //Se activan las interrupciones globales,
    //por Timer 0, y externa.
    INTCON = 0b11110000;
    while(1) //Bucle infinito.
    {
        }
}

```

Para la simulación de este ejemplo en ISIS, se implementarán los siguientes dispositivos: 16F877A, RES, BUTTON, LED-RED, y los instrumentos virtuales: VIRTUAL TERMINAL y OSCILLOSCOPE. Después de correr la simulación se debe poder interactuar con el virtual terminal, al enviar datos, con el pulsador se debe comutar el estado del LED, y la salida del osciloscopio debe mostrar una señal cuadrada con cambios cada 65,535m Segundos, de acuerdo con el Timer 0. Para correr la simulación se debe hacer el siguiente circuito:



Circuito 11-1

12 Sensores

La implementación de sensores en sistemas microcontrolados es de gran utilidad en muchas de sus aplicaciones, estos permiten obtener lecturas de variables como temperatura, aceleración, presión, humedad, velocidad, luminosidad, contraste, entre otras. Su aplicación es importante en sistemas de control, robótica, e instrumentación.

12.1 Sensor de temperatura LM35

El sensor LM35, es un dispositivo activo de 3 terminales que permite adquirir la temperatura ambiente en rangos de -55 a 150 grados Celsius o centígrados. Este dispositivo es de fácil implementación dado que solo cuenta con dos terminales de polarización, y una salida de voltaje directamente proporcional a la temperatura. Este sensor puede ser polarizado de 4 a 30 voltios y tiene una salida de 10m voltios por cada grado Celsius. La apariencia física del sensor y su distribución de pines, así como la vista en ISIS, son las que se pueden ver en las siguientes figuras:



Figura 12-1

Para realizar la lectura del voltaje de salida del sensor se implementa en el microcontrolador el módulo ADC. La máxima salida del sensor es 1,5 voltios, cuando la temperatura es 150 grados Celsius. Por esto es importante cambiar el valor de referencia positiva del convertidor análogo digital, con el fin de mejorar la resolución de la medida de voltaje. Para el ejemplo de este capítulo se configurará el voltaje de referencia positivo del ADC, en 2,5 voltios. Cambiando la referencia positiva a 2,5 voltios el convertidor entregará un resultado binario de 1023 cuando el voltaje a convertir es de 2,5 voltios. Para el caso de este sensor, se verá definido por las siguientes relaciones:

$$\frac{1023}{2,5V} = \frac{R_{adc}}{V_{adc}} \quad \text{Ecuación 12-1}$$

Donde R_{adc} es el resultado binario de la conversión AD. De esta ecuación se puede deducir que el voltaje V_{adc} , leído por el convertidor AD, es:

$$V_{adc} = \frac{(2,5V)(R_{adc})}{1023} \quad \text{Ecuación 12-2}$$

Trabajando con la relación del sensor que es: 10m voltios por cada grado Celsius, se puede plantear la siguiente ecuación:

$$\frac{10mV}{1^{\circ}C} = \frac{Vadc}{n^{\circ}C} \text{ Ecuación 12-3}$$

Donde n es la temperatura en grados Celsius, que está registrando el sensor, de esta ecuación se puede deducir que la temperatura n es:

$$n^{\circ}C = \frac{(1^{\circ}C)(Vadc)}{10mV} \text{ Ecuación 12-4}$$

Reemplazando la ecuación (12.2), en (12.4), se obtiene la siguiente relación:

$$n^{\circ}C = \frac{(2,5)(Radc)}{10,23} = 0,244Radc \text{ Ecuación 12-5}$$

Esta relación debe ser implementada en la conversión AD, en el programa del PIC.

Para contextualizar el uso de este sensor se puede observar y analizar el siguiente código fuente para un PIC 16F877A:

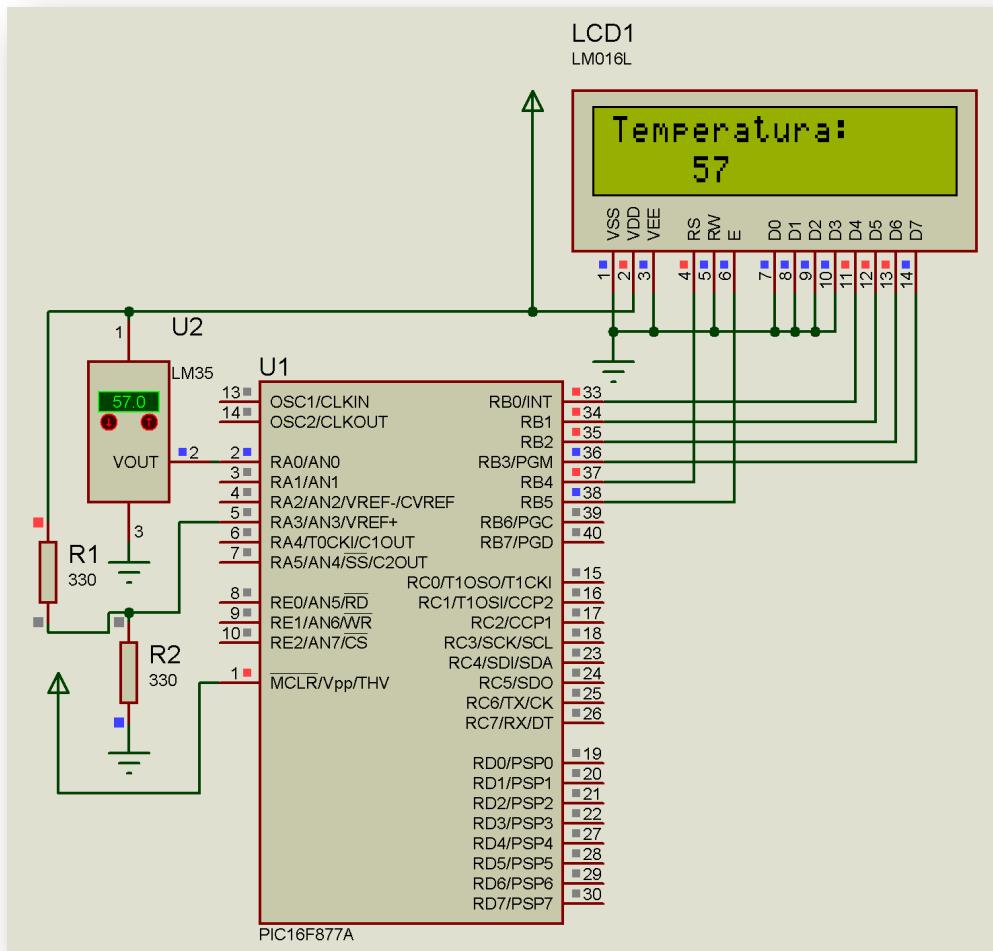
```
//Definición de pines del LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;
//Definición de los TRIS del LCD
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;
void main( void )
{
    //Declaración de variables.
    unsigned int Radc, TemI;
    float Tem;
    char Text[16];
    //Configura el módulo ADC con el pin AN3
    //como voltaje de referencia positiva.
    ADCON1 = 0b11000001;
    //Inicio del LCD.
    Lcd_Init();
    //Borrado del cursor.
    Lcd_Cmd(_LCD_CURSOR_OFF);
    //Impresión de texto.
    Lcd_Out( 1, 1, "Temperatura:");
}
```

```

while(1) //Bucle infinito.
{
    //Lectura del canal 0 del ADC.
    Radc = ADC_Read(0);
    //Uso de la ecuación (13.5).
    Tem = 0.244*Radc;
    //Se convierte el resultado a un número entero.
    TemI = Tem;
    //Se convierte el número entero a una cadena de caracteres.
    IntToStr( TemI, Text );
    //Se imprime el resultado.
    Lcd_Out( 2, 1, Text );
    //Retardo de 100m segundos.
    delay_ms(100);
}
}

```

Terminada la edición y compilación del programa se debe construir un circuito en ISIS con los siguientes dispositivos: 16F877A, RES, LM35, y LM016L, este se puede apreciar en la siguiente figura:



Circuito 12-1

El arreglo de resistencias de 330Ω permite hacer un divisor de voltaje para crear la referencia de 2,5 voltios. Durante la marcha de la simulación se puede cambiar el valor de la temperatura en el sensor LM35, para ver su funcionamiento.

12.2 Sensores de presión

La presión es una variable física, representada por un vector que está definido como la fuerza aplicada en una superficie específica. La presión se denota en diferentes unidades dependiendo del sistema, la presión se puede dar en: psi, Pascal, Atmósferas, centímetros de mercurio, etc. Para los fines prácticos de este capítulo se trabajará con el sensor de presión MPX4115, este sensor está caracterizado en Kilo Pascal, y permite medir presiones entre 15 y 115 KPa, o entre 2,18 y 16,7 psi. La apariencia física, la distribución de pines y la vista en ISIS, de este dispositivo es la siguiente:

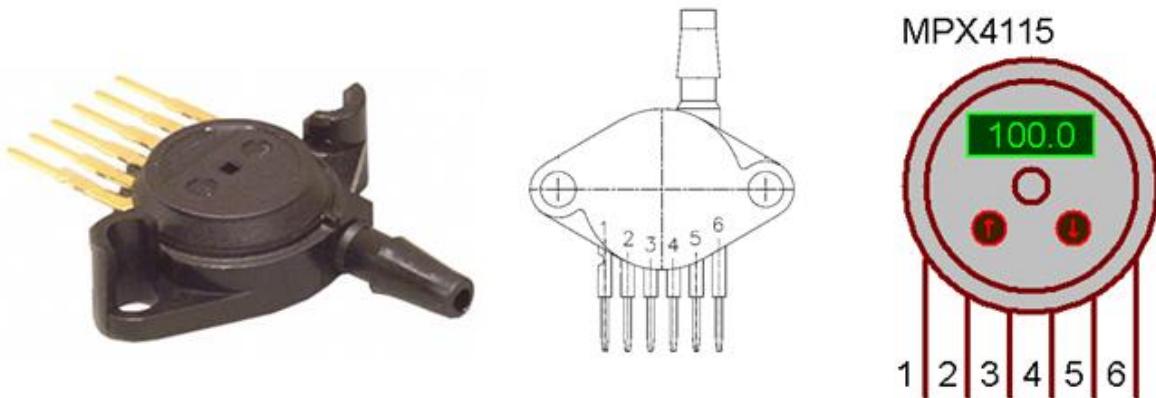


Figura 12-2

| Pin | Función |
|---------|-----------------|
| 1 | V_{out} |
| 2 | GND, Referencia |
| 3 | V_s , Poder |
| 4, 5, 6 | No implementado |

Tabla 12-1

Independiente del sensor que se implemente, cada uno debe contar con una relación de salida o función de transferencia, en el caso particular del sensor MPX4115 la función de transferencia especificada por el fabricante es:

$$V_{out} = V_s(0,009P - 0,095) \text{ Ecuación 12-6}$$

Donde V_{out} es el voltaje de salida del sensor, P es la presión en Kilo Pascal, y V_s el voltaje de alimentación del sensor. Despejando de la ecuación (13.6), la presión P , se obtiene:

$$P = \frac{111,11V_{out}}{V_s} + 10,555 \text{ Ecuación 12-7}$$

En función del convertidor AD, del PIC configurado con 10 bits de resolución se puede concluir que:

$$\frac{5V}{1023} = \frac{nV}{Radc} \quad Ecuación\ 12-8$$

Por lo tanto despejando el voltaje n de la ecuación (12.8) y remplazando en la ecuación (12.7), se puede obtener la relación o ecuación que se debe usar en el microcontrolador para hacer la lectura del sensor, está relación es la siguiente:

$$P = \frac{0,54306Radc}{Vs} + 10,555 \quad Ecuación\ 12-9$$

Para el caso particular de este ejemplo se usará una fuente de poder de 5 voltios la cual alimenta el sensor y el micro, por lo tanto Vs es igual a 5 voltios, y la relación final queda de la siguiente forma:

$$P = 0,10861Radc + 10,555 \quad Ecuación\ 12-10$$

Para demostrar el funcionamiento práctico de este tipo de sensores se puede observar y analizar el siguiente código fuente:

```
//Definición de pines del LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;

//Definición de los TRIS del LCD
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;

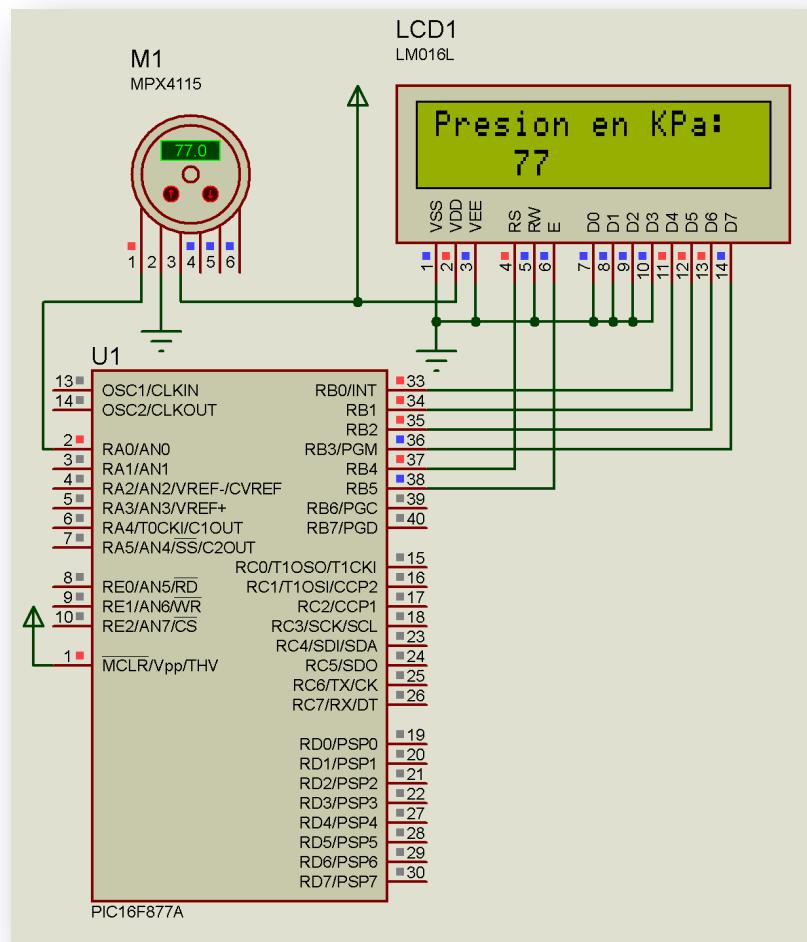
void main( void )
{
    //Declaración de variables.
    unsigned int Radc, PreI;
    float Pre;
    char Text[16];
    //Inicio del LCD.
    Lcd_Init();
    //Borrado del cursor.
    Lcd_Cmd(_LCD_CURSOR_OFF);
    //Impresión de texto.
    Lcd_Out( 1, 1, "Presion en KPa:");
}
```

```

while(1) //Bucle infinito.
{
    //Lectura del canal 0 del ADC.
    Radc = ADC_Read(0);
    //Uso de la ecuación (13.9).
    Pre = 0.10861*Radc+10,5555;
    //Se forza el resultado a la parte entera.
    PreI = Pre;
    //Se convierte el número entero a una cadena de caracteres.
    IntToStr( PreI, Text );
    //Se imprime el resultado.
    Lcd_Out( 2, 1, Text);
    //Retardo de 100m segundos.
    delay_ms(100);
}
}

```

Terminada la edición y compilación del programa se puede implementar en el simulador ISIS los dispositivos: 16F877A, MPX4115, y LM016L, para realizar la simulación del sistema, el circuito debe tener la siguiente apariencia:



Circuito 12-2

12.3 Sensores de distancia

Los sensores de distancia permiten medir una longitud desde el punto de ubicación del sensor hasta un obstáculo puesto en un punto dentro del rango de trabajo del sensor. La implementación de este tipo de dispositivos es de utilidad en sistemas de control, y en proyectos de robótica. Para el caso puntual de este capítulo se trabajará con el sensor GP2D12, este dispositivo usa como estrategia de medida un rayo de luz infrarroja y su reflejo para determinar la longitud. La apariencia física la distribución de pines y la vista en el simulador ISIS del sensor son los siguientes:

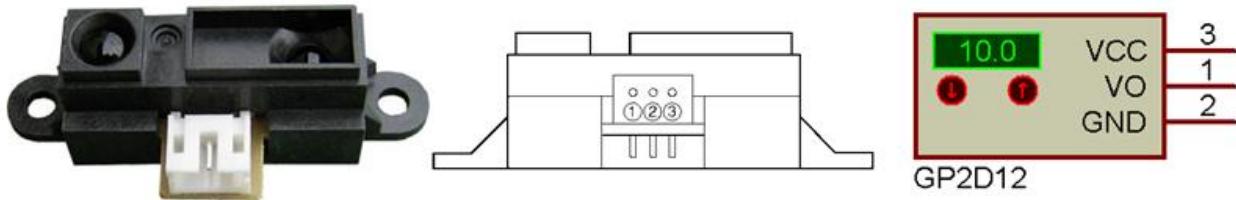


Figura 12-3

| Pin | Función |
|-----|-----------------|
| 1 | Vout |
| 2 | GND, Referencia |
| 3 | Vcc, Poder |

Tabla 12-2

Este tipo de sensores cuentan con un comportamiento no lineal, esto quiere decir que la salida no obedece a una función de transferencia lineal. Para comprender de forma clara este concepto se puede observar y analizar la siguiente gráfica que muestra el comportamiento de la salida en función de la distancia:

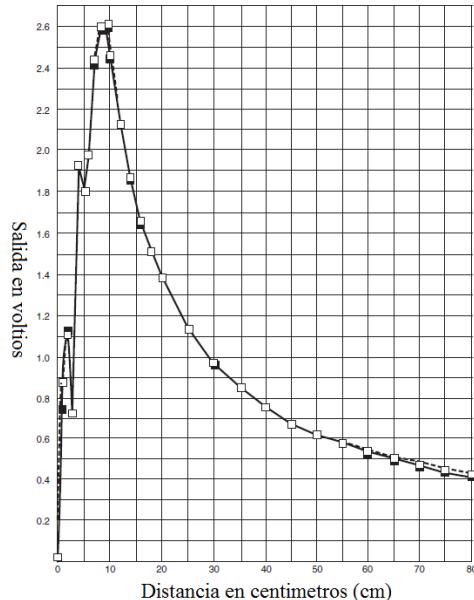


Figura 12-4

Este comportamiento en un sensor es de difícil aplicación, dado que en algunos casos los fabricantes no entregan una función de transferencia útil de tal forma que pueda ser usada en una sola ecuación. Sin embargo no es un obstáculo para el uso del sensor. Como se puede apreciar en la hoja técnica el rango de acción es de 10 a 80 centímetros tal como se puede apreciar en la gráfica. La gráfica mostrada está contenida en la hoja técnica del dispositivo y está es realizada experimentalmente. Para poder caracterizar el comportamiento del sensor y realizar una función de transferencia es indispensable realizar un proceso matemático conocido como linealización. Para este fin se hace una interpolación con una cantidad finita de puntos conocidos de la función, estos puntos se toman de las medidas experimentales. Cuantos más puntos se evalúen más precisa será la función de transferencia.

Para iniciar este proceso se escogen 2 puntos, que estén equitativamente distribuidos. Por ejemplo el punto a 10cm, y 80cm. Para cada uno de los puntos se realiza una ecuación de orden igual al número de puntos menos uno, en este caso serán ecuaciones de primer orden. Se asume como variable independiente x los datos de entrada que en este caso son el voltaje de salida del sensor, como variable dependiente y , la distancia en centímetros. Para comprender este concepto observe la siguiente ecuación que representa el comportamiento exponencial del sensor:

$$y = \frac{A}{x} + B \quad \text{Ecuación 12-11}$$

Recordando que y , representa la distancia y que x , el voltaje de salida del sensor, se usan los datos de dos puntos y se remplazan en las ecuaciones de la siguiente forma:

| <i>Distancia(y)</i> | <i>Vo, Salida sensor(x)</i> |
|---------------------|-----------------------------|
| 10cm | 2,35V |
| 80cm | 0,41V |

Tabla 12-3

Sistema de ecuaciones:

$$10 = \frac{A}{2,35} + B \quad \text{Ecuación 12-12}$$

$$80 = \frac{A}{0,41} + B$$

Sistema de ecuaciones con dos incógnitas:

$$\begin{aligned} 10 &= A0,425531 + B \\ 80 &= A2,43902 + B \end{aligned} \quad \text{Ecuación 12-13}$$

Solucionando el sistema de ecuaciones se puede determinar que las constantes son:

$$A=34,76546392$$

$$B=-4,793814433$$

De esta forma se puede formalizar la función de transferencia del sensor con la ecuación:

$$y = \frac{34,76546392}{x} + 4,793814433 \text{ Ecuación 12-14}$$

Como la variable x representa el voltaje de salida del sensor y teniendo presente que el valor del sensor no supera 2,5 voltios, la referencia del convertidor AD, se puede ajustar a 2,5 voltios. Para establecer la conversión se puede usar la ecuación: (12.15)

$$Vadc = \frac{2,5Radc}{1023} \text{ Ecuación 12-15}$$

Remplazando la ecuación (12.15) en la ecuación (12.14), obtenemos la siguiente relación o función de transferencia:

$$y = \frac{14226,02784}{Radc} - 4,793814433 \text{ Ecuación 12-16}$$

El paso siguiente es implementar un código fuente que use la función de transferencia y que posteriormente visualice el valor capturado, para este fin se puede observar y analizar el siguiente código fuente:

```
//Definición de pines del LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;

//Definición de los TRIS del LCD
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;

void main( void )
{
    //Declaración de variables.
    unsigned int Radc, DisI;
    float Dis;
    char Text[16];
    //Configura el módulo ADC con el pin AN3
    //como voltaje de referencia positiva.
    ADCON1 = 0b11000001;
    //Inicio del LCD.
    Lcd_Init();
    //Borrado del cursor.
```

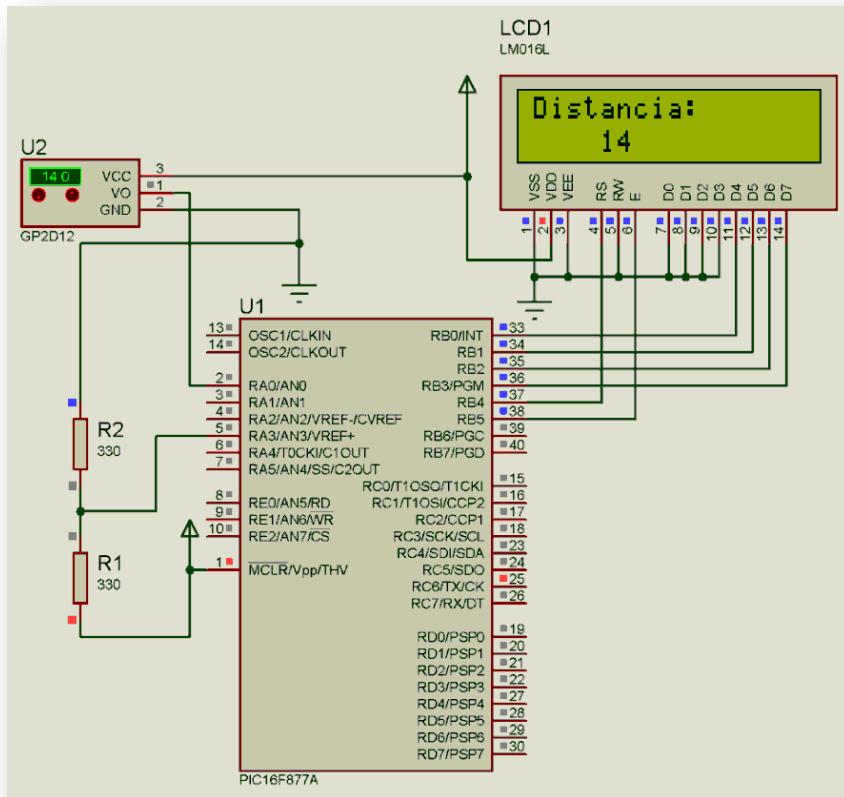
```

Lcd_Cmd(_LCD_CURSOR_OFF);
//Impresión de texto.
Lcd_Out( 1, 1, "Distancia:");

while(1) //Bucle infinito.
{
    //Lectura del canal análogo.
    Radc=ADC_Read(0);
    //Implementación de la función de transferencia (13.15).
    Dis = (14226.02784/Radc)-4.793814433;
    //Se para el resultado a un valor entero.
    DisI=Dis;
    //Se convierte el valor entero en cadena de texto.
    IntToStr( DisI, Text );
    //Se imprime la lectura del sensor.
    Lcd_Out( 2, 1, Text );
    //Retardo de 100m segundos.
    delay_ms(100);
}
}

```

Por último para verificar el comportamiento del sistema se implementa en ISIS, los dispositivos: 16F877A, RES, LM016L, y el sensor GP2D12, en el siguiente circuito:



Circuito 12-3

12.4 Sensores LDR

Los sensores LDR, son dispositivos que cambian su resistencia en función de la intensidad de luz, cuanto mayor sea la intensidad de la luz, menor es la resistencia que ofrece la LDR. Estos dispositivos son útiles para determinar, la presencia o ausencia de luz en el ambiente, es posible concebir con estos dispositivos controles de luminosidad. La apariencia física y las vistas posibles en el simulador ISIS, son las siguientes:

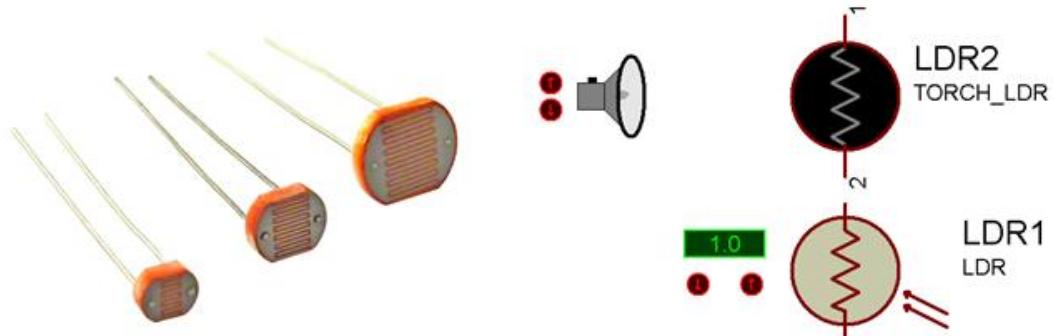
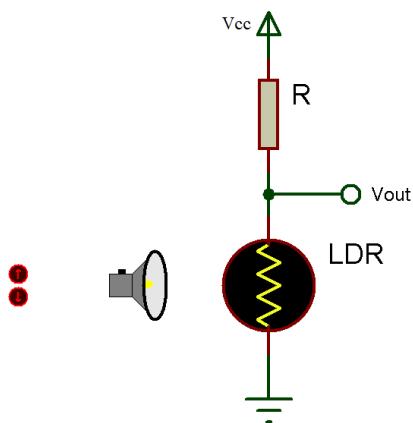


Figura 12-5

Las características eléctricas de estos dispositivos no son fácil de conseguir dado su simple funcionamiento y gran variedad de fabricantes, las LDR, o fotoresistencias, se adquieren comercialmente en tamaños diversos, esto implica que el rango de resistencia cambia en función del tamaño, una LDR, de tamaño grande tiene rangos de cambio menor que una LDR pequeña. Incluso los cambios de resistencia no son exactamente iguales en dos LDR, del mismo tamaño. Estas razones hacen que la forma de mayor simplicidad para su uso sea la lectura de su resistencia, o el voltaje que se desempeña en sus terminales. Para usar una LDR, la forma más simple es realizar un divisor de voltaje con una resistencia fija, para entender este concepto se puede observar el siguiente arreglo o circuito:



Circuito 12-4

Asumiendo la teoría básica de circuitos eléctricos para un divisor de voltaje se puede implementar la siguiente ecuación:

$$V_{adc} = \frac{LDR(V_{cc})}{LDR + R} \quad \text{Ecuación 12-17}$$

Despejando de la ecuación (12.17), el valor de la foto resistencia, LDR se obtiene la siguiente ecuación:

$$LDR = \frac{(V_{out})(R)}{V_{cc} - V_{out}} \quad Ecuación\ 12-18$$

Dado que el valor de salida V_{out} se puede ingresar a una entrada AD, del microcontrolador se puede asumir como V_{adc} , con la siguiente relación y un voltaje de referencia positivo de 5 voltios:

$$V_{adc} = \frac{(R_{adc})(5)}{1023} = (R_{adc})0,004887585 \quad Ecuación\ 12-19$$

Remplazando la ecuación (13.19) en la ecuación (13.18), y asumiendo que V_{cc} es de 5 voltios, obtenemos la siguiente relación:

$$LDR = \frac{(R_{adc})(0,04887585)(R)}{5 - (R_{adc})(0,004887585)} \quad Ecuación\ 12-20$$

Para el caso particular de la simulación de este ejercicio la resistencia del divisor de voltaje será de $10k\Omega$ de esta manera la ecuación para calcular la resistencia de la LDR, es:

$$LDR = \frac{(R_{adc})(48,87585)}{5 - (R_{adc})(0,004887585)} \quad Ecuación\ 12-21$$

Para la compilación y edición del programa se puede observar y analizar el siguiente código fuente:

```
//Definición de pines del LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;
//Definición de los TRIS del LCD
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;

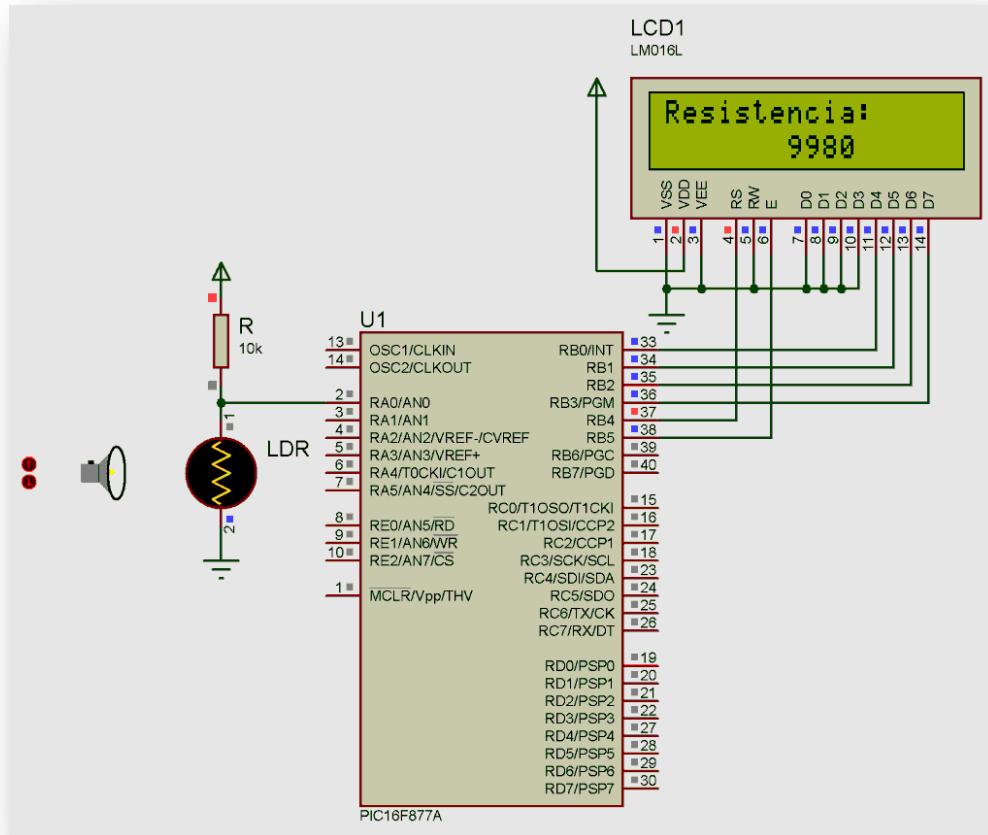
void main( void )
{
    //Declaración de variables.
    unsigned long Radc, DisI;
    char Text[16];
    //Inicio del LCD.
```

```

Lcd_Init();
//Borrado del cursor.
Lcd_Cmd(_LCD_CURSOR_OFF);
//Impresión de texto.
Lcd_Out( 1, 1, "Resistencia:");
while(1) //Bucle infinito.
{
    //Lectura del canal análogo.
    Radc=ADC_Read(0);
    //Implementación del cálculo de la LDR, ecuación (13.20).
    DisI = (Radc*48.87585533)/(5.0-Radc*0.004887585);
    //Se convierte el valor entero largo, en cadena de texto.
    LongToStr( DisI, Text );
    //Se imprime la lectura del sensor.
    Lcd_Out( 2, 1, Text );
    //Retardo de 100m segundos.
    delay_ms(100);
}
}

```

Para realizar la simulación en ISIS, se implementan los siguiente dispositivos: 16F877A, RES, LM016L, TORCH_LDR, en el siguiente circuito electrónico:



Circuito 12-5

12.5 Sensores de humedad y temperatura

La humedad y la temperatura son dos variables físicas de amplio uso en la industria y para su medida se pueden usar dispositivos que integran las dos lecturas simultáneamente, en un solo encapsulado. La familia de sensores SHT7x permiten hacer la lectura de ambos parámetros, con una calibración de fábrica que garantiza la fidelidad de las lecturas. La apariencia física y la vista de estos dispositivos en ISIS, así como su distribución de pines, es la siguiente:

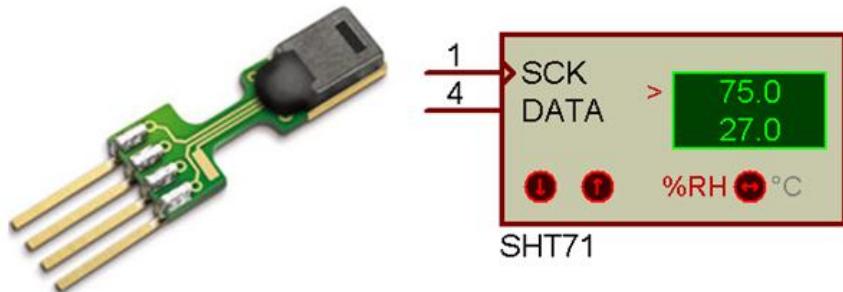


Figura 12-6

| Pin | Función |
|-----|-----------------|
| 1 | Reloj |
| 2 | Vdd, Poder |
| 3 | Gnd, Referencia |
| 4 | Datos |

Tabla 12-4

Estos sensores entregan la información por medio de un puerto serial similar al protocolo I²C, sin embargo no es exactamente igual, y por esta razón no es posible usarlo. Esto implica realizar por medio del código de programa, el protocolo de comunicación especificado en la hoja técnica del dispositivo. El protocolo incluye una condición de inicio, bloque de datos tanto de lectura como de escritura, y un bit de conformación o ACK. La comunicación se basa en dos pines uno de reloj que siempre debe ser de salida y un pin de datos bidireccional, el cual es de drenador abierto, lo que hace necesario implementar una resistencia pull-up en esta línea a Vcc de 10kΩ.

Para realizar la comunicación con el sensor, el desarrollador debe implementar su propia librería para el enlace de datos, a continuación se mostrará paso a paso un ejemplo que permite cumplir con este cometido:

```
//Declaración de pines de comunicación.  
sbit SHT_Reloj at RC0_bit;  
sbit SHT_Datos at RC1_bit;  
  
//Declaración de TRIS de los pines de comunicación.  
sbit SHT_Reloj_Dire at TRISC0_bit;  
sbit SHT_Datos_Dire at TRISC1_bit;  
  
//Declaración de constantes.  
const unsigned short RH_8bits_TEM_12bits = 1;
```

```

const unsigned short RH_12bits_TEM_14bits = 0;
const unsigned short Peso_bits[8] = {1,2,4,8,16,32,64,128};

//Bandera de resolución.
unsigned short RH_TEM;

//Función para establecer un estado
//lógico en el pin de datos
void SHT_Bus( unsigned short e )
{
    //Si el estado es 1 se impone en alta
    //impedancia el pin de datos.
    if( e )
        SHT_Datos_Dire = 1;
    else
    {
        //Si el estado es un 0 lógico
        //se establece la salida en 0.
        SHT_Datos_Dire = 0;
        SHT_Datos = 0;
    }
}

//Función para establecer el estado del reloj.
void SHT_Reloj_Salida( unsigned short e )
{
    //Se establece el estado lógico en el reloj.
    if( e )
        SHT_Reloj = 1;
    else
        SHT_Reloj = 0;
    //Periodo del reloj, para frecuencia de 1MHz.
    delay_us(1);
}

//Función para crear la condición de inicio.
void SHT_Inicio( void )
{
    //Secuencia de inicio.
    SHT_Reloj_Salida(1);
    SHT_Bus(0);
    SHT_Reloj_Salida(0);
    SHT_Reloj_Salida(1);
    SHT_Bus(1);
    SHT_Reloj_Salida(0);
    delay_ms(1);
}

```

```

//Función para escribir un dato en el bus.
void SHT_Escribe_Bus( unsigned short d )
{
    unsigned short n;
    //Bucle para el envío serial.
    for( n=7; n!=255; n-- )
    {
        //Se asigna el bit según su peso.
        if( (d>>n)&1 )
            SHT_Bus( 1 );
        else
            SHT_Bus( 0 );
        //Generación de pulsos de reloj.
        SHT_Reloj_Salida( 1 );
        SHT_Reloj_Salida( 0 );
    }
    //Generación de bit ACK.
    SHT_Bus( 1 );
    SHT_Reloj_Salida( 1 );
    SHT_Reloj_Salida( 0 );
}

```

```

//Función para leer un dato en el bus.
unsigned short SHT_Leer_Bus( void )
{
    unsigned short n, Dato_SHT=0;
    //Se configura el bus de entrada.
    SHT_Bus( 1 );
    //Bucle para recibir los bits de forma serial.
    for( n=7; n!=255; n-- )
    {
        //Pulso de reloj.
        SHT_Reloj_Salida( 1 );
        //Se reciben los bits según su peso.
        if( SHT_Datos )
            Dato_SHT += Peso_bits[n];
        SHT_Reloj_Salida( 0 );
    }
    //Se transmite la condición de ACK.
    SHT_Bus( 0 );
    SHT_Reloj_Salida( 1 );
    SHT_Reloj_Salida( 0 );
    //Se deja el bus de entrada.
    SHT_Bus( 1 );
    //Se retorna el valor de la lectura.
    return Dato_SHT;
}

```

//Función para grabar el registro Status.

```

void SHT_Status_Graba( unsigned short s )
{
    //Se transmite el valor del status.
    SHT_Inicio();
    SHT_Escribe_Bus( 0b00000110 );
    SHT_Escribe_Bus( s );
}

```

```

//Función para leer el valor del Status.
unsigned short SHT_Status_Leer( void )
{
    unsigned short st;
    SHT_Inicio();
    SHT_Escribe_Bus( 0b00000111 );
    st=SHT_Leer_Bus();
    SHT_Leer_Bus();
    return st;
}

```

```

//Función para leer la temperatura.
float SHT_Leer_Tem( void )
{
    float tem;
    unsigned int LEC=0;
    unsigned short DH, DL;
    //Se aplica el protocolo de lectura.
    SHT_Inicio();
    SHT_Escribe_Bus( 0b00000011 );
    //Espera para hacer la lectura.
    delay_ms(310);
    DH=SHT_Leer_Bus();
    DL=SHT_Leer_Bus();
    SHT_Leer_Bus();
    LEC = DL + DH*256;
    //Se implementan las ecuaciones para
    //la temperatura.
    if( RH_TEM )
        tem = -40.1 + 0.04*LEC;
    else
        tem = -40.1 + 0.01*LEC;
    //Se retorna la temperatura.
    return tem;
}

```

```

float SHT_Leer_Hum( void )
{
    float hum;
    unsigned int LEC=0;
    unsigned short DH, DL;
}

```

```

//Se aplica el protocolo de lectura.
SHT_Inicio();
SHT_Escribe_Bus( 0b00000101 );
//Espera para hacer la lectura.
delay_ms(310);
DH=SHT_Leer_Bus();
DL=SHT_Leer_Bus();
SHT_Leer_Bus();
LEC = DL + DH*256;
//Se implemetan las ecuacion para
//la humedad.
if( RH_TEM )
    hum = -4.3468 + 0.5872*LEC - 0.00040845*LEC*LEC;
else
    hum = -4.3468 + 0.0367*LEC - 0.0000015955*LEC*LEC;
//Se retorna la humedad.
return hum;
}

//Función para inicializar el sensor.
void Inicio_SHT7x( unsigned short St )
{
    unsigned short n;
    //Configuración de pines,
    //y estado inicial del bus de datos.
    SHT_Reloj_Dire = 0;
    //Se estable el pin de datos en 1.
    SHT_Bus(1);
    //Se inicia el reloj en 0.
    SHT_Reloj_Salida(0);
    delay_ms(100);
    //Se generan 9 ciclos de reloj, para reiniciar el sensor.
    for(n=0; n<9; n++)
    {
        SHT_Reloj_Salida(1);
        SHT_Reloj_Salida(0);
    }
    //Se configura el registro Status.
    SHT_Inicio();
    SHT_Status_Graba( St );
    delay_ms(1);
    RH_TEM = St;
}

```

Para el caso puntual de este ejemplo se complementa el código con la siguiente función main:

```

void main( void )
{
    //Declaración de variables.

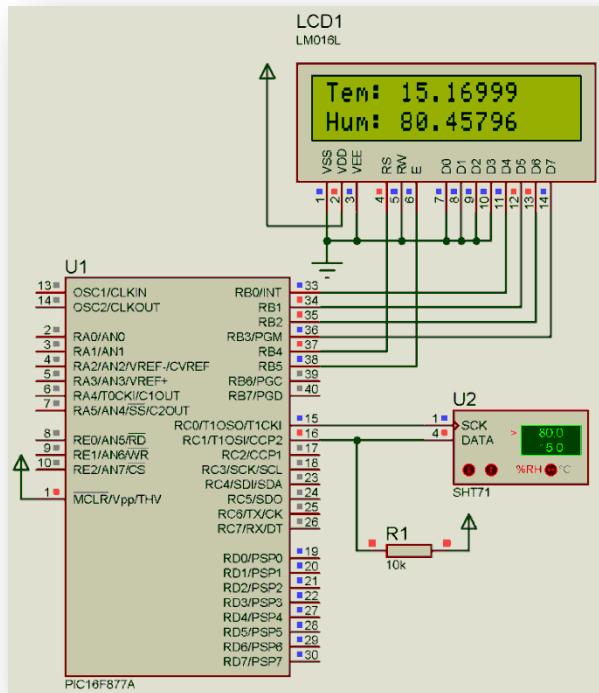
```

```

float Sen;
char Text[16];
//Inicio del LCD.
Lcd_Init();
//Se inicializa el sensor SHT7x.
Inicio_SHT7x(RH_12bits_TEM_14bits);
//Borrado del cursor.
Lcd_Cmd(_LCD_CURSOR_OFF);
//Impresión de texto.
Lcd_Out( 1, 1, "Tem:");
Lcd_Out( 2, 1, "Hum:");
while(1)//Bucle infinito.
{
    //Se lee la temperatura y se imprime en el LCD.
    Sen = SHT_Leer_Tem();
    FloatToStr( Sen, Text );
    Lcd_Out( 1, 6, Text );
    //Se lee la humedad y se imprime en el LCD.
    Sen = SHT_Leer_hum();
    FloatToStr( Sen, Text );
    Lcd_Out( 2, 6, Text );
}
}

```

Finalizada la edición y compilación del código fuente, se procede a realizar la simulación en ISIS, con los dispositivos: 16F877A, LM016L, RES, y SHT71, en el siguiente circuito electrónico:



Circuito 12-6

13 Comunicación con dispositivos

La comunicación con diversos dispositivos se hace indispensable para realizar tareas como la lectura de sensores, el sistema de control, y lectura de módulos como: relojes en tiempo real, GPS, puntos inalámbricos, entre otros. En función de la necesidad y de las capacidades del desarrollador, también es posible crear sus propios aplicativos de software para ordenadores personales, estas aplicaciones se pueden desarrollar en paquetes, como Visual Basic, Java, C++, etc. Para poder simular la conectividad con aplicativos propios e incluso con paquetes de software como Hyper Terminal, se puede recurrir a aplicativos que crean en el ordenador puertos seriales virtuales. De la misma forma el simulador ISIS permite implementar un puerto virtual con conectividad a los puertos seriales físicos del ordenador. Uno de los aplicativos que permite crear los puertos virtuales es Virtual Serial Port Kit, del cual se puede descargar una versión demostrativa en la página www.virtual-serial-port.com La apariencia visual de este paquete es la siguiente:

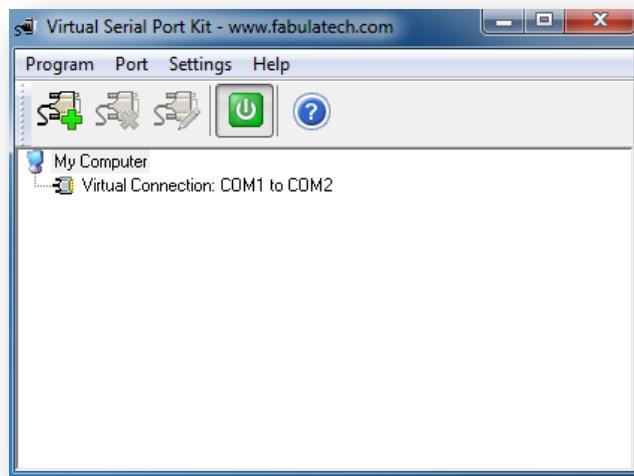


Figura 13-1

Sin embargo este es tan solo uno de los diversos aplicativos que cumplen con la misma función.

13.1 Módulos GPS

Los módulos GPS son dispositivos de comunicación inalámbrica que toman tramas de información suministrada por los satélites artificiales geoestacionarios del planeta. Los módulos GPS deben contar con la lectura de un mínimo de tres satélites para realizar un cálculo conocido como triangulación; este consiste en medir la diferencia en tiempo entre las lecturas para determinar la posición del GPS sobre la superficie de la tierra. La calidad de la lectura dependerá del número de satélites simultáneos conectados al GPS; la lectura será de mayor precisión cuantos más satélites estén conectados; sin embargo los GPS comerciales que se consiguen tienen un margen de error aproximado de 3 metros, lo que implica que con respecto al diámetro del globo terráqueo, el margen de error es casi del 0%. Los módulos GPS se pueden adquirir

comercialmente en varias presentaciones con encapsulado, o simplemente un circuito impreso con el módulo. La apariencia física de ellos se puede apreciar en las siguientes figuras:



Figura 13-2

Los módulos GPS cuentan con puertos seriales que transmiten periódicamente la lectura de las coordenadas geográficas leídas por el GPS. El periodo por defecto de los GPS, es un segundo. La lectura de un GPS, es el conjunto de coordenadas esféricas sobre el globo terráqueo, usando como referencia de medida el meridiano de Greenwich, y el paralelo del Ecuador. Para contextualizar este concepto se puede observar el siguiente gráfico:

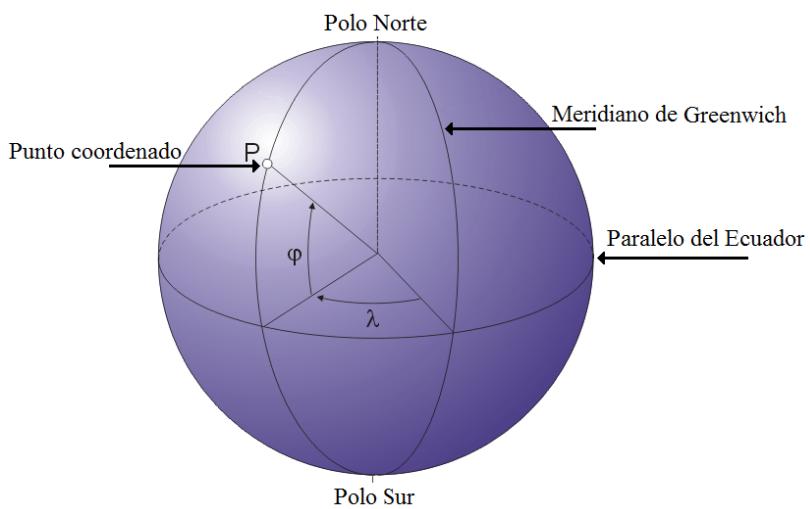


Figura 13-3

Longitud ángulo λ , Latitud ángulo φ .

Las coordenadas esféricas usan tres parámetros; el primero es la longitud, que es el ángulo λ , representado en la gráfica y se mide a partir del meridiano de Greenwich, este ángulo puede ser Este, u Oeste, y su magnitud puede variar de 0 a 90 grados. El segundo parámetro es la latitud, que es el ángulo φ , representado en la gráfica, su punto de referencia es el paralelo del Ecuador y puede ser Norte, o Sur, de la misma forma su magnitud varía de 0 a 90 grados. El último parámetro es conocido como altitud, este determina la altura del punto coordenado con respecto al nivel del mar, y su magnitud está definida en metros.

Los Módulos GPS, suministran una variedad de datos a demás de las coordenadas esféricas, como: Número de satélites detectados, especificaciones mínimas del GPS, y características de curso y velocidad, entre otras. Sin embargo para realizar la lectura de las coordenadas solo se

requiere la trama de información con el encabezado: \$GPGGA. El siguiente ejemplo muestra la lectura de las coordenadas:

Latitud: 30° 47' 43,47" Sur.

Longitud: 39° 57' 33,36" Oeste.

Altura: 2600 metros.

\$GPGGA,000000.000,3047.724500,S,03957.556000,W,1,8,0,2600,M,0,M,,*7B

Información de Latitud:

\$GPGGA,000000.000,**3047.724500,S,03957.556000,W,1,8,0,2600,M,0,M,,*7B**

Donde 3047.724500,S representa: 30° 47' 43,47" Sur. Es decir 30 grados, 47.7245 minutos, que es equivalente a 47 minutos y 43,47 segundos.

Información de Longitud:

\$GPGGA,000000.000,3047.724500,S,**03957.556000,W,1,8,0,2600,M,0,M,,*7B**

Donde: 03957.556000,W representa: 39° 57' 33,36" Oeste. Es decir 39 grados, 57.556 minutos, que es equivalente a 57 minutos y 33,36 segundos.

Información del número de satélites, conectados:

\$GPGGA,000000.000,3047.724500,S,03957.556000,W,1,8,0,2600,M,0,M,,*7B

Donde este campo 8, representa el número de satélites conectados al GPS, en este caso 8.

Información de altitud:

\$GPGGA,000000.000,3047.724500,S,03957.556000,W,1,8,0,**2600,M,0,M,,*7B**

Donde 2600,M representa la magnitud en metros y la altitud de 2600 metros.

Los campos de información siempre son fijos y estás separados por una coma (,) entre si.

El siguiente ejemplo muestra una trama completa de datos entregados por un GPS:

\$GPZDA,000000.000,30,12,1899,00,00*5F

\$GPGGA,000000.000,3354.673167,S,04125.539667,W,1,8,0,2600,M,0,M,,*78

\$GPGLL,3354.673167,S,04125.539667,W,000000.000,A,A*5F

\$GPVTG,9,T,0,M,4.86,N,9.00500277932185,K,A*2A

\$GPRMC,000000.000,A,3354.673167,S,04125.539667,W,4.86,9,301299,0,E,A*30

\$GPGSA,A,3,1,2,3,4,5,6,17,22,,,0,0,0*2D

\$GPGSV,2,1,8,1,0,24,0,2,0,24,0,3,0,24,0,4,0,24,0*46

\$GPGSV,2,2,8,5,0,24,0,6,0,24,0,17,0,24,0,22,0,24,0*44

\$GPRTE,1,1,C,0,*0B

Las tramas de información entregadas por un GPS, son de forma texto, por medio de caracteres ASCII, y finalizadas con los caracteres ASCII, enter: 13, y retroceso del carro 10.

Para fines de diseño es importante identificar la velocidad de transmisión del módulo GPS y ajustar la misma velocidad en el módulo USART del PIC.

Para iniciar el proceso de diseño con el microcontrolador es importante crear una rutina que identifique las tramas y los encabezados para poder filtrar los datos de interés, que para este caso son las coordenadas esféricas o geográficas. Es indispensable usar el módulo USART del PIC, en

este caso, y se establecerá una velocidad de transmisión de 4800 bits por segundo. Para evitar la pérdida de datos se implementará la lectura de los datos seriales por medio de las interrupciones. En primera instancia se hace un programa para el PIC 18F452, que tiene la capacidad de recibir y almacenar las tramas en un búfer de datos:

```

char Dato;
char Bufer[50];
unsigned short Pos=0, Bandera=0;

//Declaración de la función de interrupciones.
void interrupt ( void )
{
    //Se evalúa si la interrupción disparada por
    //recepción serial.
    if( PIR1.F5 )
    {
        //Se lee el dato de entrada.
        Dato = UART1_Read();
        switch( Dato )
        {
            case 13: Pos=0; //Se recibe el carácter Enter.
            Bandera = 1;
            break;
            case 10:break; //Se recibe el retroceso del carro.
            default: Bufer[Pos++]=Dato; //Se guardan los datos en el búfer.
            Bufer[Pos]=0; //Se establece el fin de cadena.
            break;
        }
    }
}

void main( void )
{
    //Se configuran los puertos.
    //
    //Se activa la interrupción por
    //recepción serial.
    PIE1 = 0b00100000;
    //Se desactivan las demás fuentes de interrupción.
    PIE2 = 0;
    //Se apagan las banderas de interrupción.
    PIR2 = 0;
    PIR1 = 0;
    //Configuración del puerto serial a 4800 bps.
    UART1_Init(4800);
    //Se activan las interrupciones globales,
    //y periféricas.
    INTCON = 0b11000000;
}

```

```

while(1) //Bucle infinito.
{
    //Se evalúa si una trama ha llegado.
    if( Bandera )
    {
        //Aquí se debe analizar el contenido de la información del búfer.
        Bandera=0; //Se apaga la bandera de llegada.
    }
}

```

El procedimiento siguiente es implementar una función que identifique la trama de información y los datos contenidos en ella, es importante recordar que los campos de información son fijos pero no necesariamente el contenido de cada campo. Para establecer la información de la lectura se usa una variable o estructura diseñada por el desarrollador:

```

typedef struct
{
    short Grados_longitud;
    float Minutos_longitud;
    char Longitud;
    short Grados_latitud;
    float Minutos_latitud;
    char Latitud;
    float Altitud;
    short Satelites;
}Coordenada;

```

A continuación se mostrará una función capaz de validar una trama para la lectura de las coordenadas:

```

//Función para detectar la trama de coordenadas.
short EsGPGGA( char *trama )
{
    //Está función retorna 1 si la trama es válida y 0 de lo contrario.
    if( trama[0]=='$' && trama[1]=='G' && trama[2]=='P'
        && trama[3]=='G' && trama[4]=='G' && trama[5]=='A' )
        return 1;
    else
        return 0;
}

```

A continuación se muestra un ejemplo de una función diseñada para leer los datos de las coordenadas:

```

//Función para adquirir los datos del campo.
Coordenada LecturaGPS( char *trama )
{
    //Declaración de variables.

```

```

    Coordenada C;
    char Texto[50];
    unsigned short j,k;
    //Valor inicial de las variables.
    C.Grados_longitud=0;
    C.Minutos_longitud=0;
    C.Longitud=0;
    C.Grados_latitud=0;
    C.Minutos_latitud=0;
    C.Latitud=0;
    C.Altitud=0;
    C.Satelites=0;

    //Se verifica que el encabezado contenga el texto $GPGGA.
    if( !EsGPGGA( trama ) )return C;
    //Se filtra el campo 9 que contiene la altitud.
    BuscarCampo( Trama, Texto, 9 );
    //Se convierte el texto en el valor numérico.
    C.Altitud = atof(Texto);
    //Se filtra el campo 7 que contiene el número de satélites detectados.
    BuscarCampo( Trama, Texto, 7 );
    //Se convierte el texto en el valor numérico.
    C.Satelites = atoi(Texto);
    //Se filtra el campo 2 que contiene los minutos de la latitud.
    BuscarCampo( Trama, Texto, 2 );
    //Se convierte el texto en el valor numérico.
    C.Minutos_latitud = atof( Texto + 2 );
    //Se truncan los minutos de la latitud.
    Texto[2]=0;
    //Se convierte el texto en el valor numérico de los grados de la latitud.
    C.Grados_latitud = atoi(Texto);
    //Se filtra el campo 3 que contiene la orientación de la latitud.
    BuscarCampo( Trama, Texto, 3 );
    C.Latitud = Texto[0];
    //Se filtra el campo 4 que contiene los minutos de la longitud.
    BuscarCampo( Trama, Texto, 4 );
    //Se convierte el texto en el valor numérico los minutos de la longitud.
    C.Minutos_longitud = atof( Texto + 3 );
    Texto[3]=0;
    //Se convierte el texto en el valor numérico de los grados de la longitud.
    C.Grados_longitud = atoi( Texto );
    //Se filtra el campo 2 que contiene.
    BuscarCampo( Trama, Texto, 5 );
    C.Longitud = Texto[0];
    return C;
}

```

La siguiente función permite buscar dentro de la trama enviada por el GPS, un campo de información determinado:

```

//Función para buscar y filtrar un campo en la trama.
void BuscarCampo( char *Fuente, char *Destino, short campo )
{
    unsigned short j=0,k=0;
    //Bucle para buscar el campo.
    while( j<campo )if( Fuente[k++]==' ')j++;
    j=0;
    //Bucle para copiar el campo.
    while( !(Fuente[k]==',' || Fuente[k]=='*') )
    {
        Destino[j++]=Fuente[k++];
        Destino[j]=0;
    }
}

```

Está última función permite convertir la información de una lectura en una cadena de coordenadas entendible en grados, minutos, segundos, y orientación:

```

//Función para encadenar los valores de una coordenada.
void Convertir_coordenada( char *coor, int grados, float minutos, char ll )
{
    //Declaración de variables de la función.
    char Text[20];
    short j=0,k=0;
    short g;
    float s;

    //Se pega en la cadena de texto, el valor de los grados.
    IntToStr( grados, Text );
    while( Text[j]!=0 )
    {
        if( Text[j]!=' ' )
            coor[k++]=Text[j];
        j++;
    }
    coor[k++]=':';
    //Se separan los segundos de los minutos.
    g = (short)minutos;
    s = minutos - (float)g;
    //Se pega en la cadena de texto, el valor de los minutos.
    IntToStr( g, Text );
    j=0;
    while( Text[j]!=0 )
    {
        if( Text[j]!=' ' )
            coor[k++]=Text[j];
        j++;
    }
}

```

```

coor[k++]=':';
//Se calculan los segundos, de la fracción de minutos.
s *= 60.0;
//Se pega en la cadena de texto, el valor de los segundos.
FloatToStr( s, Text );
j=0;
while( Text[j]!=0 )
{
    if( Text[j]!='' )
        coor[k++]=Text[j];
    j++;
}
//Se pega en la cadena de texto, la orientación.
coor[k++]=' '; coor[k++]=ll; coor[k]=0;
}

```

Complementando las funciones antes citadas y la estructura final del programa con un display LCD de caracteres y la comunicación del puerto serial se tiene el siguiente código fuente final:

```

// Definición de las conexiones del LCD.
sbit LCD_RS at RD4_bit;
sbit LCD_EN at RD5_bit;
sbit LCD_D4 at RD0_bit;
sbit LCD_D5 at RD1_bit;
sbit LCD_D6 at RD2_bit;
sbit LCD_D7 at RD3_bit;

//Definición de los registros TRIS del LCD.
sbit LCD_RS_Direction at TRISD4_bit;
sbit LCD_EN_Direction at TRISD5_bit;
sbit LCD_D4_Direction at TRISD0_bit;
sbit LCD_D5_Direction at TRISD1_bit;
sbit LCD_D6_Direction at TRISD2_bit;
sbit LCD_D7_Direction at TRISD3_bit;

//Estructura para la lectura de una coordenada esférica.
typedef struct
{
    short Grados_longitud;
    float Minutos_longitud;
    char Longitud;
    short Grados_latitud;
    float Minutos_latitud;
    char Latitud;
    float Altitud;
    short Satelites;
}Coordenada;

//Declaración de variables.

```

```

char Dato;
char Bufer[100];
unsigned short Pos=0;
unsigned short Bandera=0;
//Función para detectar la trama de coordenadas.
short EsGPGGA( char *trama )
{
    //Está función retorna 1 si la trama es valida y 0 de lo contrario.
    if( trama[0]=='$' && trama[1]=='G' && trama[2]=='P'
        && trama[3]=='G' && trama[4]=='G' && trama[5]=='A' )
        return 1;
    else
        return 0;
}

//Función para buscar y filtrar un campo en la trama.
void BuscarCampo( char *Fuente, char *Destino, short campo )
{
    unsigned short j=0,k=0;
    //Bucle para buscar el campo.
    while( j<campo )if( Fuente[k++]==',' )j++;
    j=0;
    //Bucle para copiar el campo.
    while( !(Fuente[k]==',' || Fuente[k]=='*') )
    {
        Destino[j++]=Fuente[k++];
        Destino[j]=0;
    }
}

//Función para adquirir los datos del campo.
Coordenada LecturaGPS( char *trama )
{
    //Declaración de variables.
    Coordenada C;
    char Texto[50];
    unsigned short j,k;
    //Valor inicial de las variables.
    C.Grados_longitud=0;
    C.Minutos_longitud=0;
    C.Longitud=0;
    C.Grados_latitud=0;
    C.Minutos_latitud=0;
    C.Latitud=0;
    C.Altitud=0;
    C.Satelites=0;

    //Se verifica que el encabezado contenga el texto $GPGGA.
    if( !EsGPGGA( trama ) )return C;
}

```

```

//Se filtra el campo 9 que contiene la altitud.
BuscarCampo( trama, Texto, 9 );
//Se convierte el texto en el valor numérico.
C.Altitud = atof(Texto);
//Se filtra el campo 7 que contiene el número de satélites detectados.
BuscarCampo( trama, Texto, 7 );
//Se convierte el texto en el valor numérico.
C.Satelites = atoi(Texto);
//Se filtra el campo 2 que contiene los minutos de la latitud.
BuscarCampo( trama, Texto, 2 );
//Se convierte el texto en el valor numérico.
C.Minutos_latitud = atof( Texto + 2 );
//Se truncan los minutos de la latitud.
Texto[2]=0;
//Se convierte el texto en el valor numérico de los grados de la latitud.
C.Grados_latitud = atoi(Texto);
//Se filtra el campo 3 que contiene la orientación de la latitud.
BuscarCampo( trama, Texto, 3 );
C.Latitud = Texto[0];
//Se filtra el campo 4 que contiene los minutos de la longitud.
BuscarCampo( trama, Texto, 4 );
//Se convierte el texto en el valor numérico los minutos de la longitud.
C.Minutos_longitud = atof( Texto + 3 );
Texto[3]=0;
//Se convierte el texto en el valor numérico de los grados de la longitud.
C.Grados_longitud = atoi( Texto );
//Se filtra el campo 2 que contiene.
BuscarCampo( trama, Texto, 5 );
C.Longitud = Texto[0];
return C;
}

//Función para encadenar los valores de una coordenada.
void Convertir_coordenada( char *coor, int grados, float minutos, char ll )
{
    //Declaración de variables de la función.
    char Text[20];
    short j=0,k=0;
    short g;
    float s;

    //Se pega en la cadena de texto, el valor de los grados.
    IntToStr( grados, Text );
    while( Text[j]!=0 )
    {
        if( Text[j]!=' ' )
            coor[k++]=Text[j];
        j++;
    }
}

```

```

coor[k++]=':';
//Se separan los segundos de los minutos.
g = (short)minutos;
s = minutos - (float)g;
//Se pega en la cadena de texto, el valor de los minutos.
IntToStr( g, Text );
j=0;
while( Text[j]!=0 )
{
    if( Text[j]!=' ' )
        coor[k++]=Text[j];
    j++;
}
coor[k++]=':';
//Se calculan los segundos, de la fracción de minutos.
s *= 60.0;
//Se pega en la cadena de texto, el valor de los segundos.
FloatToStr( s, Text );
j=0;
while( Text[j]!=0 )
{
    if( Text[j]!=' ' )
        coor[k++]=Text[j];
    j++;
}
//Se pega en la cadena de texto, la orientación.
coor[k++]=' '; coor[k++]=ll; coor[k]=0;
}

//Declaración de la función de interrupciones.
void interrupt ( void )
{
    //Se evalúa si la interrupción disparada por
    //recepción serial.
    if( PIR1.F5 )
    {
        //Se lee el dato de entrada.
        Dato = UART1_Read();
        switch( Dato )
        {
            case 13: Pos=0; //Se recibe el carácter Enter.
                Bandera = 1;
                break;
            case 10:break; //Se recibe el retroceso del carro.
            default:Bufer[Pos++]=Dato; //Se guardan los datos en el búfer.
                Bufer[Pos]=0; //Se establece el fin de cadena.
                break;
        }
    }
}

```

```

}

void main( void )
{
    Coordenada C;
    char Texto[20];
    //Se configuran los puertos.
    TRISB = 0;
    PORTB = 0;
    //Se activa la interrupción por
    //recepción serial.
    PIE1 = 0b00100000;
    //Se desactivan las demás fuentes de interrupción.
    PIE2 = 0;
    //Se apagan las banderas de interrupción.
    PIR2 = 0;
    PIR1 = 0;
    //Configuración del puerto serial a 4800 bps.
    UART1_Init(4800);
    //Se activan las interrupciones globales,
    //y periféricas.
    INTCON = 0b11000000;
    //Inicialización del display LCD.
    Lcd_Init();
    Lcd_Cmd(_LCD_CURSOR_OFF);
    while(1) //Bucle infinito.
    {
        //Se evalúa si una trama ha llegado.
        if( Bandera )
        {
            //Se verifica que la trama sea valida para las coordenadas.
            if( EsGPGGA( Bufer ) )
            {
                //Se leen los datos de las coordenadas y se guardan en la variable C.
                C=LecturaGPS( Bufer );
                //Se transcriben los datos de la latitud.
                Convertir_coordenada( Texto, C.Grados_latitud, C.Minutos_latitud, C.Latitud );
                //Se imprimen los datos de la latitud en el display LCD.
                Lcd_Out(1,1,"Lat           ");
                Lcd_Out(1,5,Texto);
                //Se transcriben los datos de la longitud.
                Convertir_coordenada(Texto,C.Grados_longitud,C.Minutos_longitud,C.Longitud);
                //Se imprimen los datos de la longitud en el display LCD.
                Lcd_Out(2,1,"Lon           ");
                Lcd_Out(2,5,Texto);
                //Se transcriben el valor de la altitud.
                FloatToStr(C.Altitud,Texto);
                //Se imprimen el valor de la altitud en el display LCD.
                Lcd_Out(3,1,"Altitud:      ");

```

```

    Lcd_Out(3,9,Texto);
    //Se transcriben el valor del número de satélites conectados.
    IntToStr(C.Satellites,Texto);
    //Se imprimen el número de satélites en el display LCD.
    Lcd_Out(4,1,"Satelites:      ");
    Lcd_Out(4,11,Texto);
}
Bandera=0; //Se apaga la bandera de llegada.
}
}
}

```

Después de editar y compilar el programa con fuente de reloj de 20 Mhz, se procede a crear en ISIS, un circuito con los siguientes dispositivos: PIC 18F452, LM044L, y el puerto virtual, COMPIM. Este último se debe configurar a 4800 bps, tanto en el puerto virtual como el puerto real, la vista de configuración del puerto virtual COMPIM debe ser la siguiente:

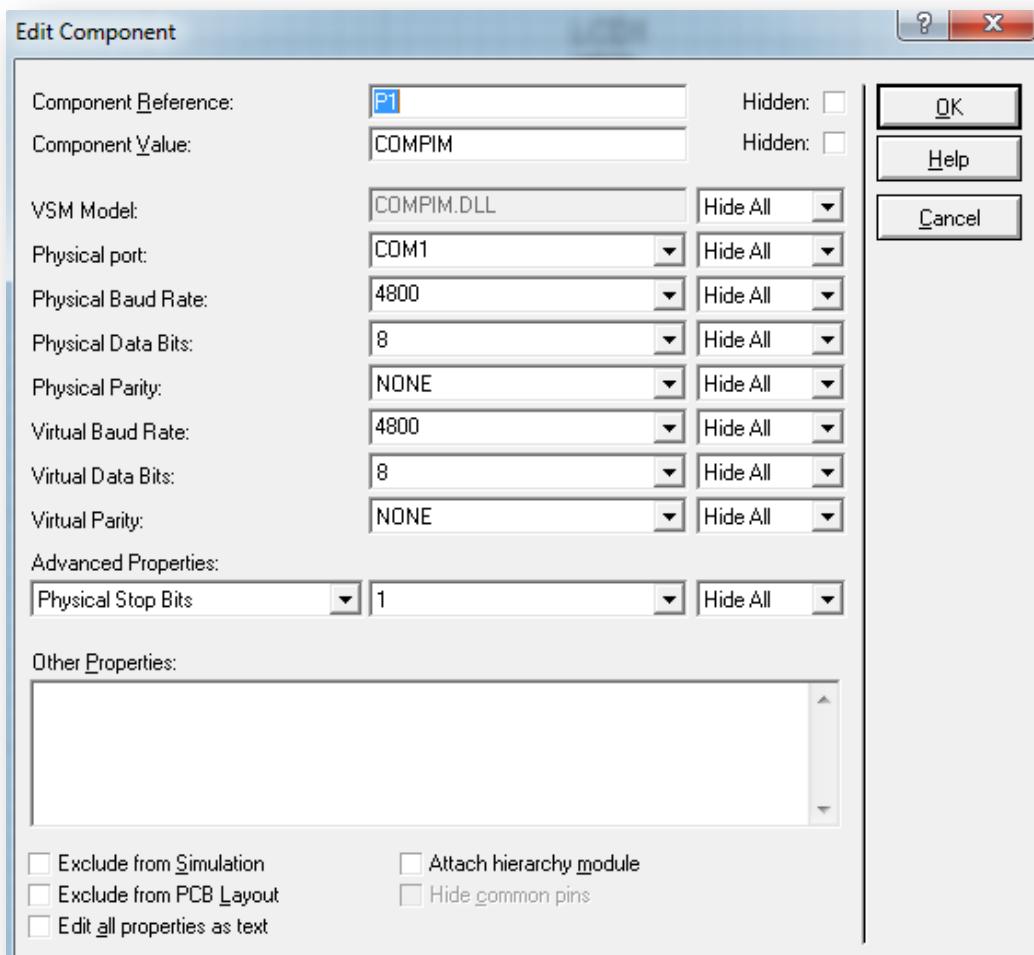
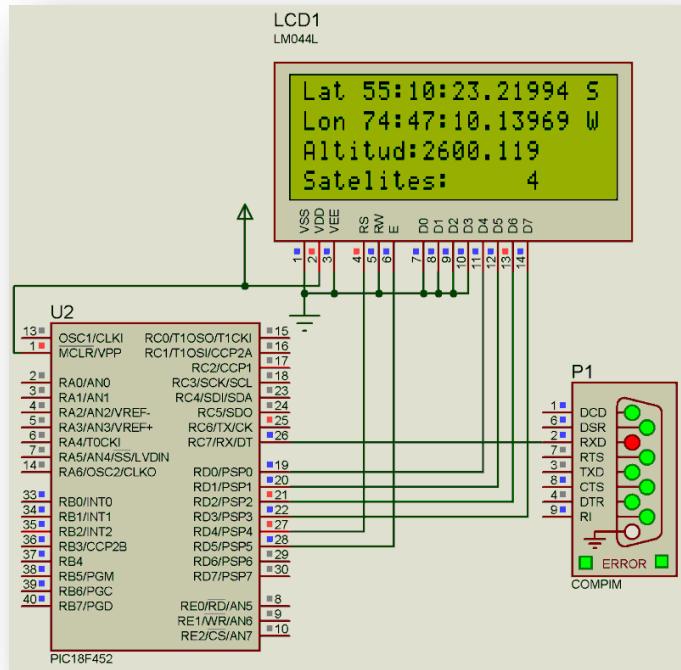


Figura 13-4

La configuración del puerto físico que en este caso está determinado como COM1, puede variar en función de los puertos disponibles en el ordenador.

La construcción del circuito en ISIS debe ser como se aprecia en la siguiente gráfica:



Circuito 13-1

La conexión del GPS, en este caso se puede hacer por medio de un simulador de GPS, o por medio de un GPS real conectado al puerto físico. Para la simulación del GPS, se puede usar el paquete de software: Virtual GPS, o cualquier otro que funcione de la misma forma, la vista del simulador de GPS, es la siguiente:

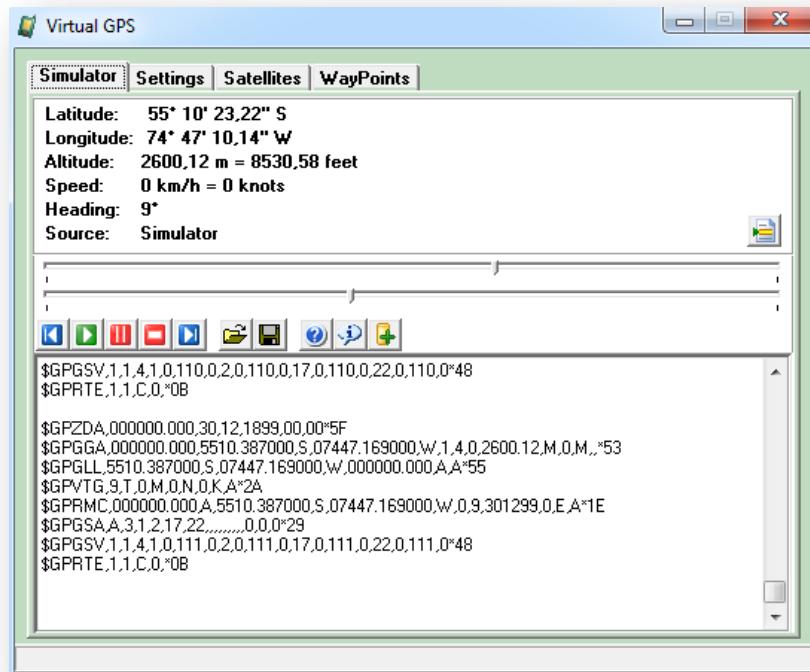


Figura 13-5

En este caso se pueden comparar los datos emitidos por el simulador y su verificación en el circuito hecho en ISIS. De la misma forma se pueden cambiar los datos enviados por el simulador de GPS. Se debe recordar hacer la creación de dos puertos virtuales cruzados, si se desea hacer la simulación de todo el sistema con un simulador de GPS.

13.2 Módulos inalámbricos unidireccionales

Las comunicaciones inalámbricas entre dos puntos hacen parte fundamental de muchas aplicaciones en el ámbito del control y el monitoreo de variables o estados. La lectura de un sensor, o el control de actuadores son posibles por medio de enlaces inalámbricos. La comunicación usada por estos dispositivos es unidireccional o simplex. Los módulos inalámbricos útiles para este fin funcionan con modulación ASK, o FSK, y tienen velocidades de transmisión hasta 9600 bits por segundo. Este tipo de dispositivos se pueden conseguir de fabricantes como Laipac Tech, o Linx Technologies, entre otros. La apariencia física de estos dispositivos es la siguiente:



Figura 13-6

Estos dispositivos solo son un medio de transmisión inalámbrico por lo cual no tienen ningún sistema de corrección ni detección de errores de transmisión, por esta razón es importante hacer dentro de la programación alguna rutina de detección de errores. Para fines de este ejemplo se implementarán los radios Laipac. De igual manera las frecuencias portadoras de estos dispositivos son variadas, y funcionan dentro del espectro de bandas libres para enlaces de comunicación. La distribución de pines de estos dispositivos es la siguiente:

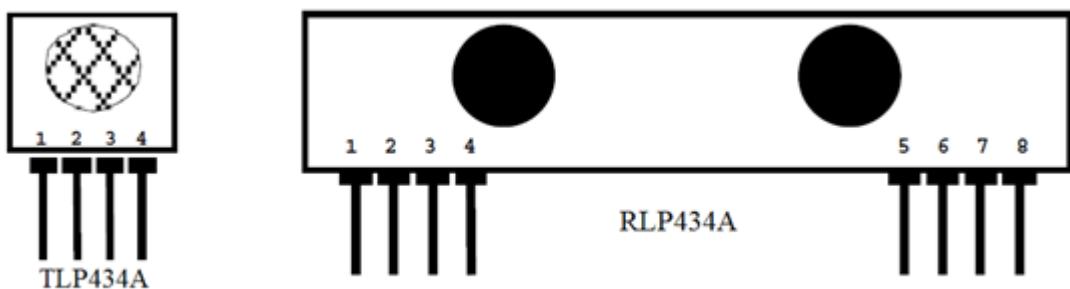


Figura 13-7

| TLP434A | |
|----------------|------------------|
| PIN | Función |
| 1 | GND, Referencia |
| 2 | Entrada serial |
| 3 | Vcc, de 5 a 12 V |
| 4 | Antena |
| RLP434A | |
| PIN | Función |
| 1 | GND, Referencia |
| 2 | Salida Serial |
| 3 | Salida de prueba |
| 4 | Vcc |
| 5 | Vcc 5 V |
| 6 | GND, Referencia |
| 7 | GND, Referencia |
| 8 | Antena |

Tabla 13-1

Para el envío de bits este ejemplo usa la modulación por ancho de pulso o PWM. El programa representa un 1 lógico con un pulso en alto de 2m segundos de duración seguido por 1m segundo en bajo, el 0 lógico se transmite con un pulso en alto de 1m segundo y 1m segundo en bajo, y como estrategia de sincronismo se implementa un pulso en alto de 500u segundos, y 500u segundos en bajo. Para este ejemplo se envían 16 pulsos de sincronismo, posteriormente se envían 8 bits con el byte que se desea transmitir, y por último se transmiten 4 bits con un número de 0 a 8 que contiene el número de unos que contiene el byte que se está transmitiendo como estrategia de verificación. Este sistema no está en capacidad de corregir errores de transmisión, pero si permite verificar el dato recibido, en otras palabras el sistema entrega un dato que es recibido correctamente y si el dato se detecta con errores se descarta. Cuando un dato es recibido correctamente se entrega un byte que puede valer de 0 a 255, y si el dato está dañado se entrega un valor igual a -1.

Para la demostración de esta comunicación se implementará una librería que transmite bytes con las reglas antes mencionadas, de igual manera los microcontroladores implementados en esta librería son los PIC 16F628A con frecuencia de reloj de 4M Hz, sin embargo puede ser usada en cualquier otro microcontrolador, la librería básica es la siguiente:

```
//Declaración de pines de comunicación.
sbit RXPIN at RB7_bit;
sbit RXPIN_TRIS at TRISB7_bit;
sbit TXPIN at RB6_bit;
sbit TXPIN_TRIS at TRISB6_bit;
//Declaración de constantes de peso binario.
const unsigned int Peso[12] = {1,2,4,8,16,32,64,128,256,512,1024,2048};
```

//Función para iniciar los pines de comunicación.

```

void Inicio_Radio_Tx( void )
{
    TXPIN_TRIS=0;
    TXPIN=0;
}

//Función para iniciar los pines de comunicación.
void Inicio_Radio_Rx( void )
{
    RXPIN_TRIS=1;
}

//Función para determinar que bit se está recibiendo.
char Entra_Bit( void )
{
    unsigned short T=0;
    while( !RXPIN ); //Se espera el estado 1 lógico.
    //Se mide el tiempo en alto de la modulación.
    while( RXPIN ){ delay_us(50); T++; }
    //Se clasifica el bit recibido en función de su tiempo.
    if( T > 38 && T < 42 )return '1'; //Aprox. 2m Seg.
    if( T > 18 && T < 22 )return '0'; //Aprox. 1m Seg.
    if( T > 8 && T < 12 )return 'S'; //Aprox. 500u Seg.
    return 'E'; // Tiempo errado.
}

//Función para enviar un byte.
int Resibir_Dato( void )
{
    unsigned short D, BIT, N_Bits;
    int DATO;
    DATO=0;
    BIT=0;
    //Se espera un bit de sincronismo.
    while( Entra_Bit()!='S' );
    //Se espera el último BIT de sincronismo.
    while( (D=Entra_Bit())=='S' );
    //Bucle para archivar el byte de entrada.
    while( 1 )
    {
        //Se evalúa el BIT que llegó.
        switch( D )
        {
            case '1': DATO += Peso[BIT++];
                break;
            case '0': BIT++;
                break;
            case 'E':
            case 'S': return -1;
        }
    }
}

```

```

        }

    //Si el número bits recibidos es 12, se
    //evalúan los errores.
    if( BIT==12 )
    {
        //Se cuentan los bits del dato de información.
        N_Bits=0;
        for( BIT=0; BIT<8; BIT++ )
            if( (DATO>>BIT)&1 )N_Bits++;
        //Se verifica que el número de unos sea igual.
        //y se retorna el byte de entrada.
        if( N_Bits == (DATO>>8)&0x0F )return DATO&0x00FF;
        //Ocurre un error y se espera una nueva trama.
        return -1;
    }
    //Se lee el siguiente bit.
    D=Entra_Bit();
}

//Función para transmitir un byte.
void Enviar_Dato( unsigned short d )
{
    unsigned short BITS, N_Bits=0;
    //Se cuentan el número de bits a enviar.
    for( BITS=0; BITS<8; BITS++ )
        if( (d>>BITS)&1 )N_Bits++;

    //Se envían 16 bits de sincronismo.
    for( BITS=0; BITS<16; BITS++ )
    {
        TXPIN = 1; delay_us(500);
        TXPIN = 0; delay_us(500);
    }

    //Se envían 8 bits de datos, el byte.
    for( BITS=0; BITS<8; BITS++ )
    {
        TXPIN=1;
        if( (d>>BITS)&1 )
            delay_ms(2);
        else
            delay_ms(1);
        TXPIN=0;
        delay_ms(1);
    }

    //Se envían 4 bits de verificación
    //con el número de unos en el byte.
}

```

```

for( BITS=0; BITS<4; BITS++)
{
    TXPIN=1;
    if( (N_Bits>>BITS)&1 )
        delay_ms(2);
    else
        delay_ms(1);
    TXPIN=0;
    delay_ms(1);
}
//Retardo final.
delay_ms(3);
}

```

Para concretar el ejemplo se realizan dos programas, uno para la transmisión, y otro para la recepción, este sistema transmite el estado de 4 pulsadores y se puede visualizar en el receptor por medio de 4 LEDs. De igual manera la simulación puede soportar los dos microcontroladores al mismo tiempo, pero no puede simular los radios. Para fines de simulación se implementa una conexión directa, sin embargo el desarrollador puede posteriormente hacer una prueba real con los radios.

Cada uno de los programas debe tener anexa la librería anteriormente citada, y anexar las siguientes funciones main, en cada uno de los códigos fuente:

Función main para el transmisor:

```

void main( void )
{
    //Declaración de variables.
    unsigned char DATO;
    //Configuración de puertos.
    TRISB = 0xFF;
    //Activación de las resistencias PULL-UP del puerto B.
    OPTION_REG = 0;
    //Inicio del puerto de transmisión.
    Inicio_Radio_Tx();

    while(1) //Bucle infinito.
    {
        //Adquisición de estados de los pulsadores.
        DATO = (~PORTB)&0x0F;
        //Transmisión del byte.
        Enviar_Dato( DATO );
    }
}

```

Función main para el receptor:

```

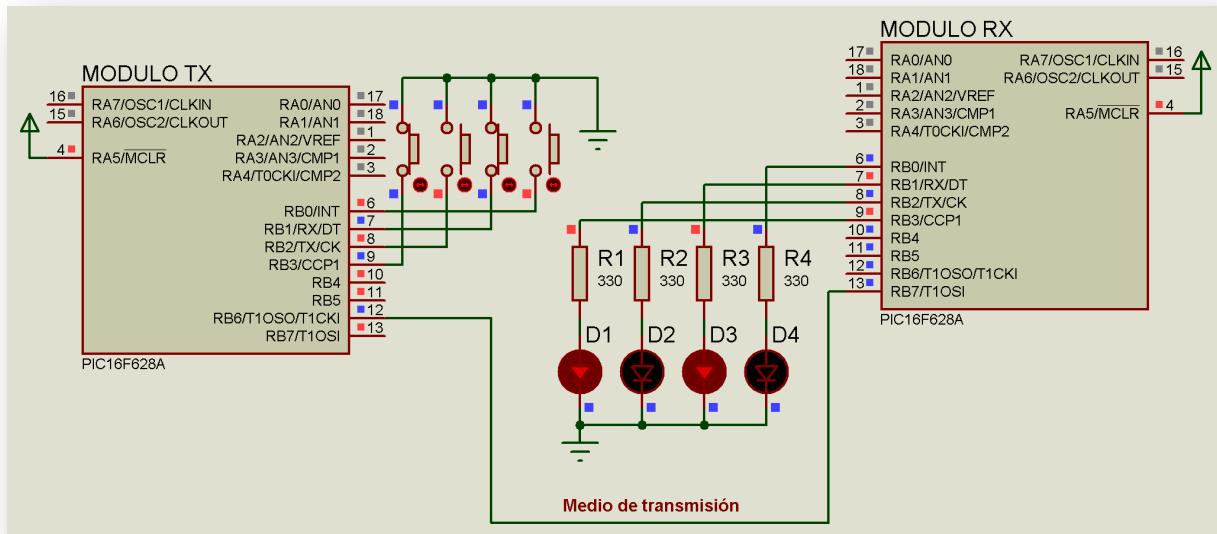
void main( void )
{

    //Declaración de variables.
    int DATO;
    //Configuración de puertos.
    TRISB = 0;
    PORTB = 0;
    //Inicio del puerto de recepción.
    Inicio_Radio_Rx();

    while(1) //Bucle infinito.
    {
        //Lectura del byte de entrada.
        DATO = Resibir_Dato();
        //Verificación de errores, y visualización del byte.
        if( DATO!= -1 )
            PORTB = DATO;
    }
}

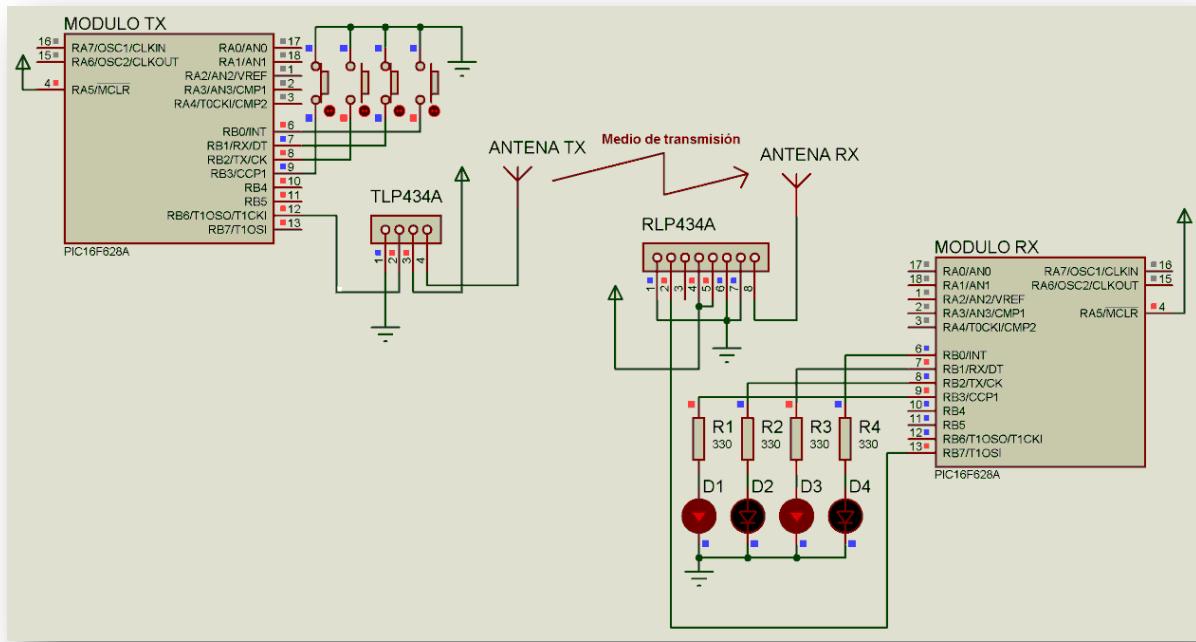
```

Para realizar la simulación se deben usar los siguientes dispositivos: PIC 16F628A, RES, LED-RED, y BUTTON, en el circuito que se puede ver a continuación:



Circuito 13-2

Para fines prácticos con los radios Laipac, se debe hacer un par de circuitos como los que se pueden ver en la siguiente gráfica:

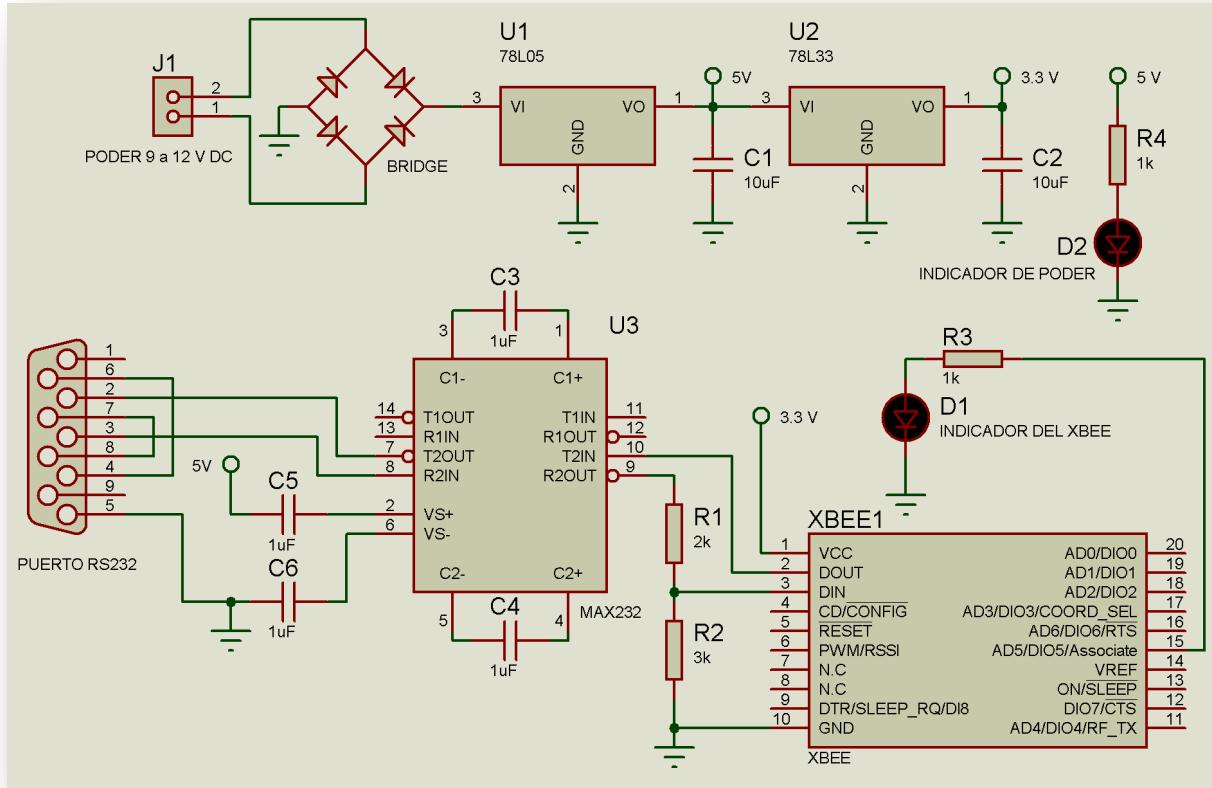


Circuito 13-3

13.3 Módulos inalámbricos bidireccionales

En algunos casos las comunicaciones deben ser imperativamente bidireccionales, como el caso de los controles realimentados y los sistemas de comunicación full duplex. Para cumplir con el propósito de realizar la comunicación full duplex se pueden implementar módulos de radio prediseñados como los XBee, o XBee PRO, estos dispositivos son módulos MODEM, que trabajan sobre la banda de 2,4GHz, e implementan el estándar IEEE 801.15.4. Estos son dispositivos que trabajan con tecnología de 3 voltios de bajo consumo, y velocidades de transmisión de 1200 a 115200 bits por segundo. El alcance de los radios depende de la antena y de la referencia de los mismos. Los módulos XBee tienen un alcance de 100 metros con una línea de vista directa y 1000 metros en los módulos XBee PRO. Los módulos XBee, son de fácil instalación y programación por medio de un paquete de software, suministrado por el fabricante, este se denomina: X-CTU, este aplicativo se puede descargar gratuitamente en la página: www.digi.com, este programa establece con el radio la comunicación para su programación por medio de un puerto serial COM. La simulación de estos radios no es posible con ISIS; pero dado el óptimo desempeño de los mismos es equivalente en ISIS a realizar una conexión cruzada entre Tx y Rx, de los microcontroladores. Los módulos XBee incorporan dentro de su hardware y software un procedimiento de detección y corrección de errores en la transmisión. De la misma forma es posible establecer una conexión inalámbrica entre un microcontrolador con un ordenador personal por medio de un puerto serial COM. Para trabajar y programar los radios XBee es necesario crear o implementar un acople de hardware para hacer la conexión con un puerto serial COM, para este propósito se debe tener presente el siguiente circuito electrónico:

Módulo de acople con un ordenador personal:



Circuito 13-4

El anterior circuito electrónico puede ser construido en una protoboard, sin embargo se puede construir un circuito impreso como el que se puede apreciar en las siguientes gráficas:

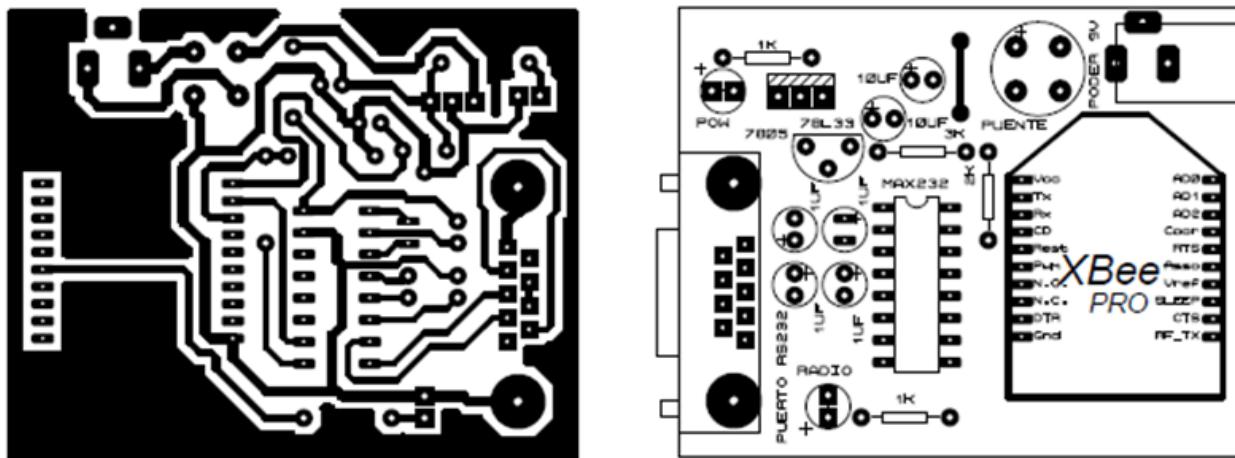


Figura 13-8

La vista de este circuito impreso finalizado y del módulo XBee, es la siguiente:



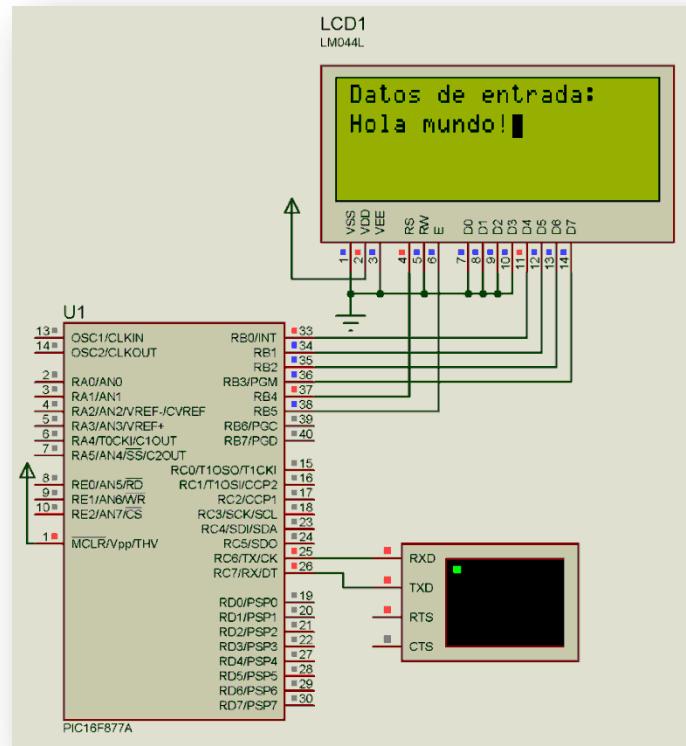
Figura 13-9

Los módulos XBee por defecto vienen programados para establecer una comunicación a 9600 bits por segundo. Sin embargo se puede cambiar por medio del programa X-CTU. La reprogramación del radio XBee, permite cambiar los parámetros de velocidad, canal de comunicación entre 16 posibles, se puede asignar un identificador del radio y asignar dirección de trabajo en una red Zigbee.

Para hacer una demostración del funcionamiento con los módulos XBee, se puede editar y analizar el siguiente código fuente:

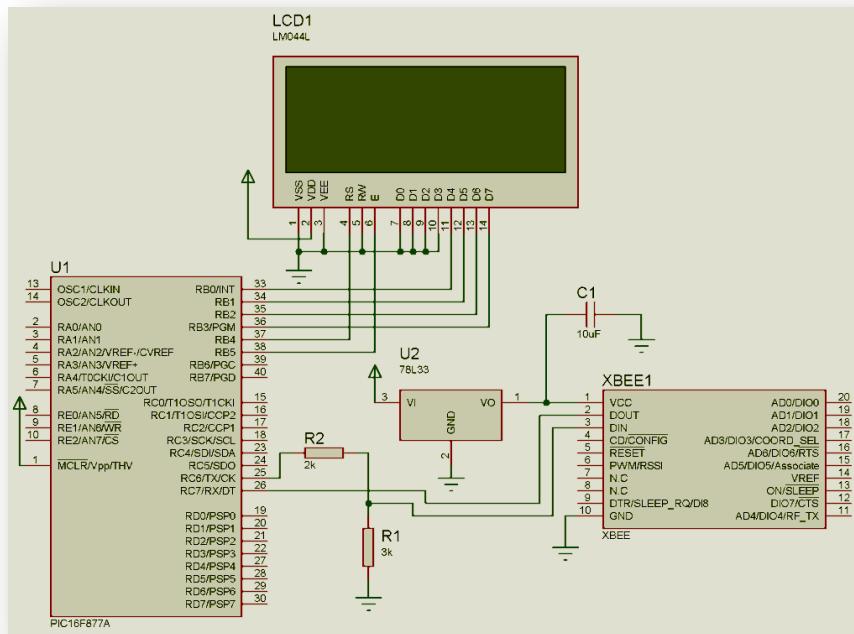
```
//Declaración de pines de control para LCD.  
sbit LCD_RS at RB4_bit;  
sbit LCD_EN at RB5_bit;  
sbit LCD_D4 at RB0_bit;  
sbit LCD_D5 at RB1_bit;  
sbit LCD_D6 at RB2_bit;  
sbit LCD_D7 at RB3_bit;  
sbit LCD_RS_Direction at TRISB4_bit;  
sbit LCD_EN_Direction at TRISB5_bit;  
sbit LCD_D4_Direction at TRISB0_bit;  
sbit LCD_D5_Direction at TRISB1_bit;  
sbit LCD_D6_Direction at TRISB2_bit;  
sbit LCD_D7_Direction at TRISB3_bit;
```

```
void main( void )  
{  
    //Declaración de variables.  
    char Texto[30];  
    char Dato;  
    unsigned short N=0;  
    //Inicio de los módulos USART, y LCD.  
    Lcd_Init();  
    UART1_Init(9600);
```

Circuito 13-5

Para fines prácticos se puede usar el módulo de acople con un ordenador personal, antes citado con una sesión de Hyper terminal de Windows. Como parte complementaria se puede hacer un circuito real como se puede ver en la siguiente gráfica:



Circuito 13-6

13.4 Comunicación RS 485

El protocolo RS 485, es un formato de comunicación serial que usa como estrategia de transmisión, la diferencia de potencial entre dos terminales denominadas A, y B. Cuando la diferencia de potencial es positiva el protocolo representa un 1 lógico y cuando la diferencia es negativa representa un 0 lógico. La comunicación RS 485, implementa un par trenzado de cobre para la transmisión de datos, como características notables de este protocolo se tiene que puede establecer la comunicación en una topología en forma de bus, con un alcance máximo de 1000 metros sin la necesidad de implementar dispositivos de repetición. De igual forma es posible conectar al bus de datos un máximo de 32 nodos, con una conectividad half duplex, esto quiere decir que la comunicación es bidireccional pero no al mismo tiempo. El protocolo RS 485 permite establecer comunicaciones con una velocidad de hasta 2,5M bits por segundo. Para la implementación de este tipo de comunicación se usan convertidores integrados que permiten hacer un enlace de niveles TTL, a RS 485, el circuito integrado más popular para estas aplicaciones es el MAX485, que es un integrado de 8 pines. La distribución de pines y la apariencia física de este dispositivo es la siguiente:

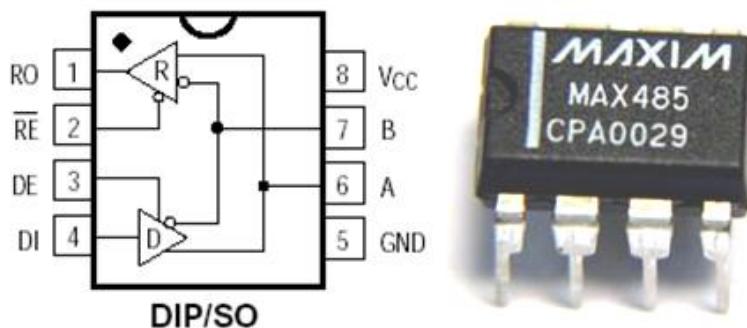


Figura 13-10

Los pines A, y B, son el bus de datos bidireccional, los pines DE, y RE, son los habilitadores de los búfer de transmisión y recepción respectivamente. Los pines DI, y RO, son respectivamente los terminales de transmisión y recepción. Como medida adicional se deben implementar un par de resistencias de 120Ω en paralelo a los pines A, y B, en el primer y en el último terminal de la red RS 485.

Para demostrar una comunicación con este tipo de redes se realizará en este capítulo una comunicación de half duplex entre dos microcontroladores, para este fin se usarán los módulos USART, y un control adicional con los pines de habilitación. Para contextualizar el uso de la comunicación RS 485, se puede observar y analizar el siguiente código de programa que permite enviar el estado de un potenciómetro para controlar la intensidad de luz de una lámpara en el terminal opuesto:

```
void main( void )
{
    //Declaración de variables.
    unsigned short DATO;
    //Configuración de puertos.
    TRISB = 0b11111110;
    PORTB = 0; //Se Configura la MAX485 como receptor.
```

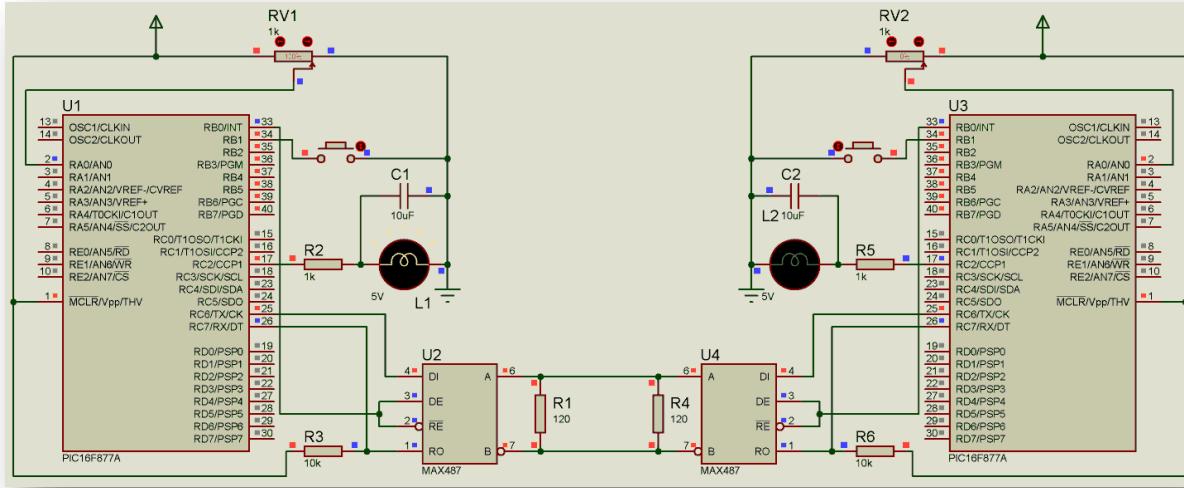
```

//Se activan las resistencias PULL-UP.
OPTION_REG = 0;
//Se configura el módulo USART a 250000 bps.
UART1_Init(250000);
//Se configura el módulo PWM con una frecuencia de 1K Hz.
PWM1_Init(1000);
//Se inicia el módulo PWM con 0%.
PWM1_Start();
PWM1_Set_Duty(0);
while(1) //Bucle infinito.
{
    //Se evalúa si un dato ha llegado.
    if( UART1_Data_Ready() )
    {
        DATO = UART1_Read();
        //Se evalúa si el dato de llegada vale 170.
        if( DATO==170 )
        {
            //Se espera el siguiente dato.
            while( !UART1_Data_Ready() );
            DATO = UART1_Read();
            //Se actualiza el ciclo útil del PWM.
            PWM1_Set_Duty(DATO);
        }
    }
    //Se evalúa si se ha pulsado el botón.
    if( Button(&PORTB, 1, 50, 0) )
    {
        PORTB = 1; //Se Configura la MAX485 como transmisor.
        //Se lee el canal análogo con el voltaje del
        //potenciómetro, y se guardan los 8 bits de mayor peso
        //en la variable DATO.
        DATO = (ADC_Read(0)>>2)&0xFF;
        //Se transmite la bandera 170, para sincronizar la comunicación.
        UART1_Write(170);
        //Se hace una pausa de 1m segundos entre datos.
        delay_ms(1);
        //Se transmite el valor de la variable DATO.
        UART1_Write(DATO);
        //Se espera a que se suelte el botón.
        while( Button(&PORTB, 1, 50, 0) );
        PORTB = 0; //Se Configura la MAX485 como receptor.
    }
}
}

```

El anterior programa es simétrico, es decir que los dos microcontroladores de la simulación se cargan con el mismo programa. El circuito electrónico recurre a una lámpara incandescente con el fin de mostrar la intensidad de la luz, la diferencia de potencial se obtiene del módulo PWM,

con la conversión digital análogo. Para fines ideales de la simulación las características por defecto de la lámpara son reconfiguradas cambiando la impedancia de 24Ω a $100K\Omega$ y su voltaje nominal de 12V a 5V. Para realizar la simulación se deben implementar los siguientes dispositivos: 16F877A, RES, MAX487, LAMP, BUTTON, CAP, y POT-HG. La teoría vista en este capítulo cita el dispositivo MAX485, sin embargo para fines de simulación se usa el MAX487, que tiene características y funcionamiento similar. El circuito a implementar es el siguiente:



Circuito 13-7

Cuando la simulación está corriendo, cada uno de los módulos MAX, están configurados como receptores, y en el momento que se pulsa un botón el microcontrolador hace la conversión análoga digital, y transmite esta información para configurar la intensidad de luz en el módulo opuesto.

13.5 Módulos inalámbricos infrarrojos

La comunicación infrarroja es una estrategia práctica para sistemas que no requieren una gran velocidad de transmisión, y que admiten línea de vista directa. El alcance de un enlace infrarrojo depende de la potencia del emisor. Sin embargo este estándar es utilizado en la comunicación de control de los electrodomésticos convencionales como: televisores, equipos de sonido, reproductores de video, aires acondicionados entre otros, tiene un alcance promedio de 10 metros.

La transmisión de datos se realiza por medio de diodos emisores de luz infrarroja, con una modulación digital por amplitud, ASK. Los estándares de comunicación para los controles infrarrojos implementan frecuencias portadoras de 36, 38, y 40 K Hz entre otras. La recepción de los datos se hace por medio de un demodulador ASK, que puede recibir datos hasta los 2,4K bits por segundo. Los demoduladores ASK, infrarrojos son dispositivos de carácter integrado que se pueden conseguir comercialmente. Estos poseen tres terminales que corresponden a dos de polarización, Vcc o poder, y referencia, y una salida demodulada en amplitud, con colector abierto, esto quiere decir que se requiere el uso de una resistencia PULL-UP, a Vcc, en este último terminal. Algunos fabricantes incorporan esta resistencia en la integración del dispositivo.

Para fines de diseño es importante tener presente que la salida del demodulador está invertida, es decir que cuando el dispositivo detecta la señal portadora la salida del demodulador se activa en bajo y se activa en alto cuando se detecta la ausencia de la misma. La apariencia física de estos dispositivos es la siguiente:



Figura 13-11

La distribución de pines puede variar de un fabricante a otro, sin embargo siempre serán las mismas tres, Vcc, Referencia, y Salida, para fines prácticos se hace imperativo consultar la ficha técnica del dispositivo que se use.

Para la simulación de este tipo de enlaces se puede usar en ISIS el módulo virtual: IRLINK, el cual tiene la siguiente apariencia en el simulador:

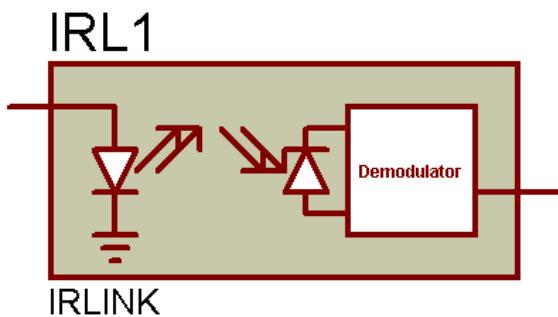


Figura 13-12

Este dispositivo virtual simula el enlace inalámbrico por medio del LED infrarrojo, y el dispositivo demodulador, de igual manera permite editar la frecuencia de la señal portadora ASK. Para contextualizar el desempeño de estos enlaces se debe observar y analizar el siguiente código fuente, que usa el módulo PWM, como fuente portadora, y el módulo USART, como señal moduladora, la información transmitida son datos ASCII, con textos fijos que se pueden ver en un hyper terminal virtual:

```
void main( void )
{
    //Configuración del módulo USART a 2400 bps.
    UART1_Init(2400);
    //Configuración de PWM para crear la frecuencia portadora a 38K Hz.
    PWM1_Init(38000);
    PWM1_Start();
```

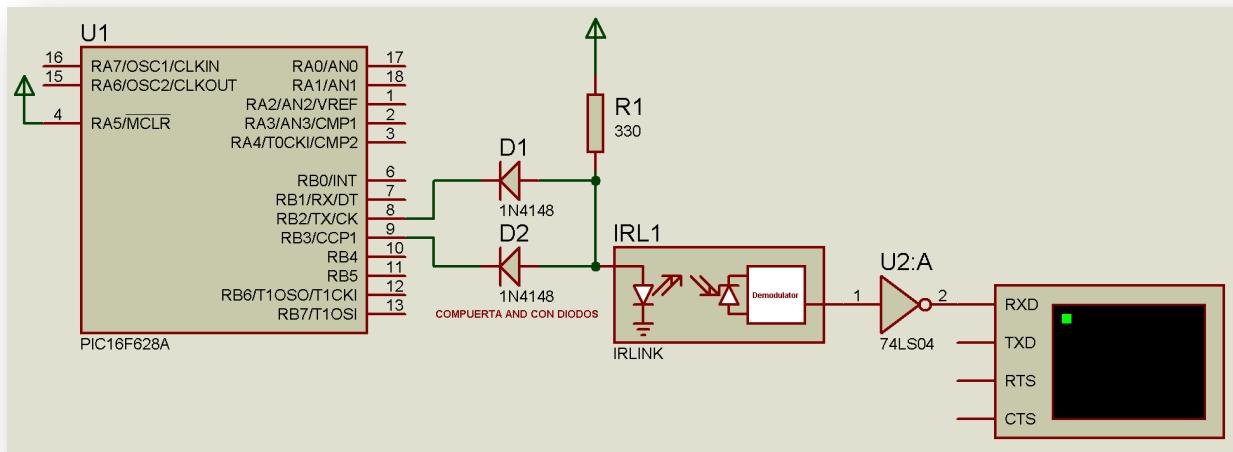
```

//Se configura el ciclo útil del PWM al 50%.
PWM1_Set_Duty(127);

while(1) //Bucle infinito.
{
    //Se transmite información en un texto fijo.
    UART1_Write_Text("Transmision de datos infrarrojos.");
    //Se envía los caracteres enter y retroceso del carro.
    UART1_Write(13);
    UART1_Write(10);
    //Se transmite información en un texto fijo.
    UART1_Write_Text("Está emision está modulada a 38K Hz.");
    UART1_Write(13);
    UART1_Write(10);
    //Se transmite información en un texto fijo.
    UART1_Write_Text("Con una velocidad de transmision de 2400 bps.");
    //Se envía los caracteres enter y retroceso del carro.
    UART1_Write(13);
    UART1_Write(10);
    //Se envía los caracteres enter y retroceso del carro.
    UART1_Write(13);
    UART1_Write(10);
    //Se realiza una pausa de 200m segundos.
    delay_ms(200);
}
}

```

Para la simulación de este ejemplo se implementan en ISIS los dispositivos: 16F628A, RES, 1N4148, IRLINK, 74LS04, y el VIRTUAL TERMINAL. El circuito electrónico debe ser ensamblado de la siguiente manera:



Circuito 13-8

La modulación se realiza por medio de la multiplicación de la señal moduladora, con la señal portadora, para esto se implementa una compuerta AND, diseñada con diodos rectificadores de switcheo rápido. En funcionamiento de la simulación, el hiperterminal virtual debe mostrar los datos de texto de la siguiente forma:

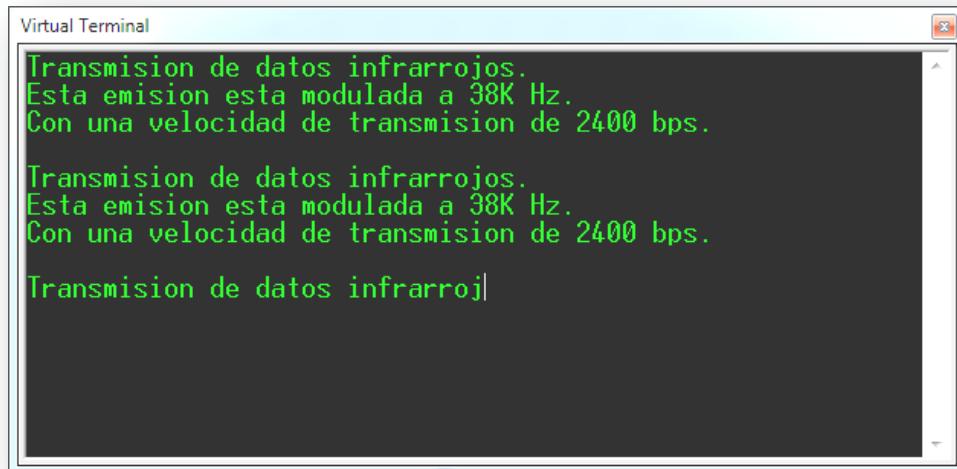


Figura 13-13

13.6 Comunicación con memorias SD

Las memorias de almacenamiento masivo SD, pueden ser manipuladas por los microcontroladores de alta gama como la familia 18F. Para leer o escribir sobre una memoria SD, se requiere trabajar con segmentos de memoria de 512 bytes. Para este fin es necesario contar con espacios de memoria RAM, de gran capacidad como los microcontroladores de la familia 18F. La comunicación de las memorias SD, requieren de un protocolo serial conocido como SPI, para este fin los microcontroladores cuentan con un módulo de comunicación serial de este tipo. La alimentación de voltaje de las memorias SD es de 3 voltios, para esto se requiere hacer un acople de voltajes con el microcontrolador por medio de un divisor de voltaje con resistencias. La apariencia física y la distribución de pines de este tipo de memorias son las siguientes:



Figura 13-14

La capacidad de las memorias SD actuales es del orden de los Giga bytes, sin embargo la máxima capacidad que soporta la librería para estas memorias en MikroC PRO, es de 1G bytes. Para la manipulación de este tipo de memorias el compilador MikroC PRO, cuenta con una librería

denominada: Mmc. La cual posee tres funciones para establecer de forma simple la comunicación y control de la memoria. La función: ***unsigned char Mmc_Init(void);***; está función inicializa la comunicación con la memoria SD, y retorna 0 si la comunicación se estableció exitosamente, o retorna 1 si algún error se presenta. La función: ***unsigned char Mmc_Read_Sector(unsigned long sector, char *dbuf);***; está permite leer un sector de 512 bytes, guardando la lectura en el apuntador *dbuf*, e iniciando en el sector definido por el parámetro *sector*. La función: ***unsigned char Mmc_Write_Sector(unsigned long sector, char *dbuf);***; está función trabaja de forma similar a la función de lectura, en las dos funciones se retorna el valor 0 si la operación culmina exitosamente o retornan 1 si un fallo se presenta. Para la comunicación física se requieren 4 conexiones que son: una línea de reloj, dos líneas de datos, entrada y salida, y un selector de pastilla.

Para contextualizar el funcionamiento de estos dispositivos se puede observar y analizar el siguiente código fuente:

```
//Pines de conexión para el selector de pastilla
sbit Mmc_Chip_Select      at RC2_bit;
sbit Mmc_Chip_Select_Direction at TRISC2_bit;

void main( void )
{
    //Declaración de variables.
    char SD;
    char Texto[50];
    long SEC=0;
    unsigned int Cont;
    unsigned short Menu=0;
    unsigned short NN;
    char Dato;
    unsigned short Bufer[512];
    //Inicio del puerto serial a 9600 bps.
    UART1_Init(9600);
    //Inicio del puerto SPI, para la memoria SD.
    SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV64, _SPI_DATA_SAMPLE_MIDDLE,
    _SPI_CLK_IDLE_HIGH, _SPI_LOW_2_HIGH);
    //Inicio de la memoria.

    if( Mmc_Init() == 0 )
    {
        //Confirmación exitosa de conexión.
        SD = 1;
        UART1_Write_Text("// La memoria fue iniciada exitosamente. //");
        UART1_Write(13);
        UART1_Write(10);
        UART1_Write(13);
        UART1_Write(10);
    }
}
```

```

}

else
{
    //Confirmación fallida de conexión.
    SD = 0;
    UART1_Write_Text("ERROR no se puede iniciar la memoria !!!");
    UART1_Write(13);
    UART1_Write(10);
    UART1_Write(13);
    UART1_Write(10);
}

while(1) //Bucle infinito.
{
    if( SD ) //Se ejecuta el Menú si la conexión fue exitosa.
    {
        //Se despliega el Menú.
        UART1_Write_Text( "1. Grabar un Sector." );
        UART1_Write(13);
        UART1_Write(10);
        UART1_Write_Text( "2. Leer un Sector." );
        UART1_Write(13);
        UART1_Write(10);
        UART1_Write_Text( "Digite una opcion: " );
        Menu = 0;
        //Bucle para esperar la opción.

        while( Menu==0 )
        {

            if( UART1_Data_Ready() )
            {
                Dato = UART1_Read();
                UART1_Write(Dato);
                UART1_Write(' ');
                switch( Dato )
                {
                    case '1': Menu=1; break;
                    case '2': Menu=2; break;
                    default: break;
                }
            }
        }

        //Envío del enter y retroceso del carro.
        UART1_Write(13);
        UART1_Write(10);
        //Captura del sector de lectura o escritura.
    }
}

```

```

UART1_Write_Text( "Digite el numero del sector de memoria, y pulse ENTER: " );
SEC=-1;
NN=0;
Texto[0]=0;
//Bucle para digital el sector.

while( SEC== -1 )
{
    if( UART1_Data_Ready() )
    {
        Dato = UART1_Read();
        UART1_Write(Dato);
        switch( Dato )
        {
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
            case '0': Texto[NN++]=Dato;
            Texto[NN]=0;
            break;
            case 13: SEC=atol(Texto);
            default : break;
        }
    }
    //Envío del enter y retroceso del carro.
    UART1_Write(13);
    UART1_Write(10);
    //Se graban o se leen los sectores.
    if( Menu==1 )
    {
        UART1_Write_Text( "Se grabara el sector: " );
        LongToStr( SEC, Texto );
        UART1_Write_Text( Texto );
        //Envío del enter y retroceso del carro.
        UART1_Write(13);
        UART1_Write(10);
        //Se borra el Búfer.
        for( Cont=0; Cont<512; Cont++ )
            Bufer[Cont]=0;
        //Bucle para capturar los datos a grabar.
    }
}

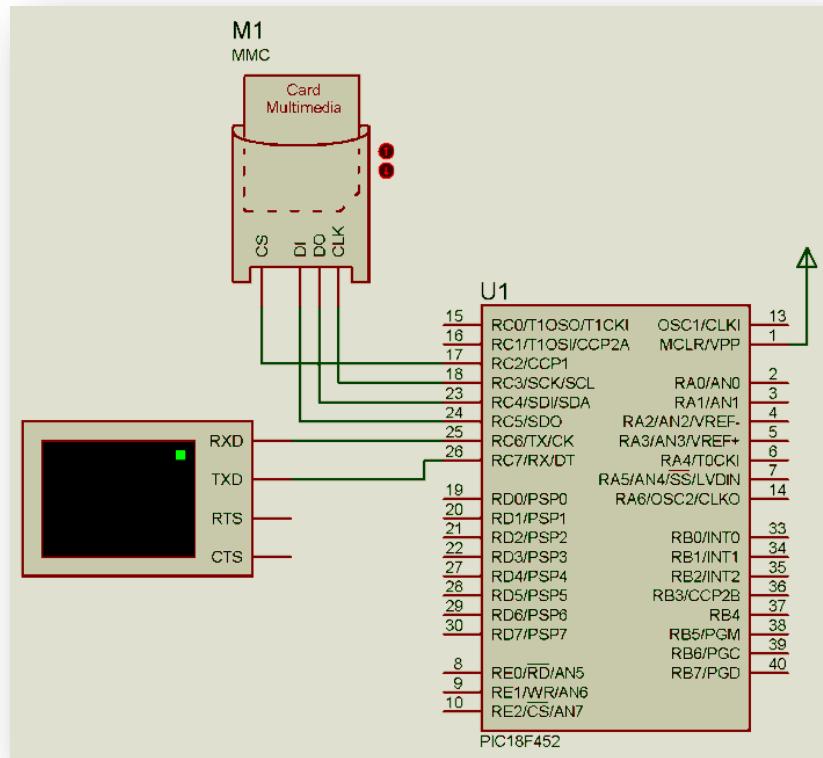
```

```

while( Menu )
{
    if( UART1_Data_Ready() )
    {
        Dato = UART1_Read();
        UART1_Write(Dato);
        switch( Dato )
        {
            case 13: Menu=0; break;
            default : Bufer[NN++]=Dato;
                        break;
        }
    }
    //Se graban los 512 datos del sector en la memoria.
    Mmc_Write_Sector( SEC, Bufer );
    //Envió del enter y retroceso del carro.
    UART1_Write(13);
    UART1_Write(10);
    UART1_Write_Text( "Fin de la grabacion." );
}
else
{
    UART1_Write_Text( "Se leera el sector: " );
    LongToStr( SEC, Texto );
    UART1_Write_Text( Texto );
    //Envió del enter y retroceso del carro.
    UART1_Write(13);
    UART1_Write(10);
    //Se leen 512 datos del sector.
    Mmc_Read_Sector( SEC, Bufer );
    //Bucle para imprimir los datos leídos de la memoria.
    for( Cont=0; Cont<512; Cont++ )
        UART1_Write( Bufer[Cont] );
    //Envió del enter y retroceso del carro.
    UART1_Write(13);
    UART1_Write(10);
    //Envió del enter y retroceso del carro.
    UART1_Write(13);
    UART1_Write(10);
}
}
}
}

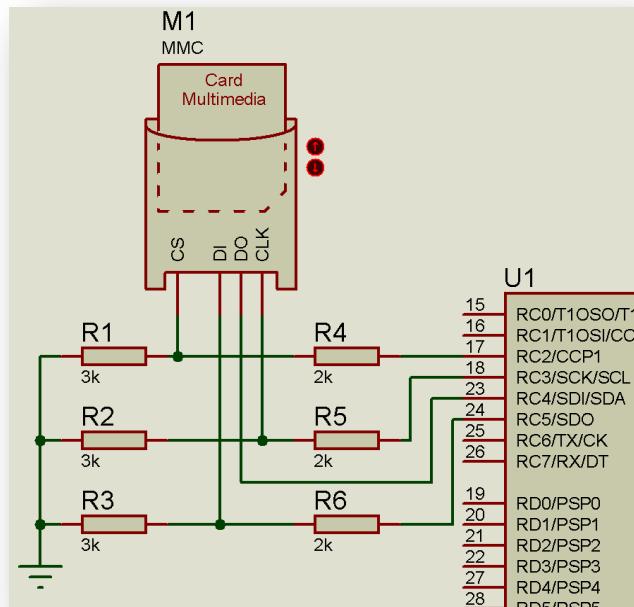
```

Como medida adicional se debe inicializar el puerto SPI, por medio de la función: *SPI1_Init_Advanced*. Para comprobar el funcionamiento de este programa se debe construir en ISIS, un circuito con los dispositivos: 18F452, MMC, y VIRTUAL TERMINAL, como se puede apreciar en la siguiente gráfica:



Círculo 13-9

Para fines de simulación las conexiones con la memoria se hacen directamente como se puede ver en el anterior circuito. Sin embargo se debe tener presente que para fines prácticos la memoria trabaja con 3 voltios, y se debe acoplar con un arreglo de resistencias como el siguiente:



Círculo 13-10

13.7 Reloj en tiempo real

Algunas aplicaciones requieren de un reloj en tiempo real, para tener dominio del tiempo de eventos, acciones, y demás. Este tipo de relojes son circuitos integrados con baterías propias, y cristales de cuarzo para definir las unidades de tiempo. Los relojes en tiempo real se comunican por medio del protocolo I²C. Una de las referencias comerciales más populares es el DS1307, que puede contar años, meses, días, hora, minutos, y segundos. La apariencia física y la vista en ISIS de estos dispositivos es la siguiente:

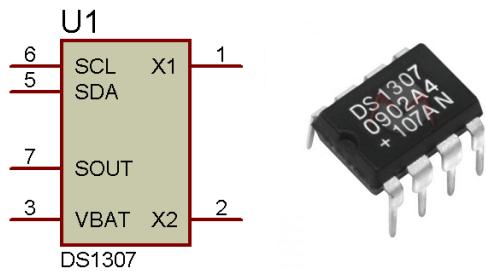


Figura 13-15

Este circuito integrado cuenta con dos pines de comunicación un pin de salida oscilante, y una entrada para la batería, además para fines prácticos cuenta con dos terminales para instalar un cristal de cuarzo de 32,768KHz. El siguiente código fuente en lenguaje C, muestra una serie de funciones que forman una librería que permiten manipular este tipo de relojes:

```
//Declaración de estructuras
typedef struct
{
    unsigned short Hor, Min, Seg;
}Hora;

typedef struct
{
    unsigned short Dia, Fec, Mes, Ano;
}Fecha;

//Función para definir dirección de memoria.
void DS1307SetDir( unsigned short dir )
{
    I2C1_Start();
    I2C1_Wr(0xD0);
    I2C1_Wr(dir);
}

//Función para convertir de código Bcd a Entero.
unsigned short BcdToShort( unsigned short bcd )
{
    unsigned short LV, HV;
    LV = bcd&0x0F;
    HV = (bcd>>4)&0x0F;
```

```

return LV + HV*10;
}

//Función para convertir de Entero a Bcd.
unsigned short ShortToBcd( unsigned short valor )
{
    unsigned short HV, LV;
    HV = valor/10;
    LV = valor - HV*10;
    return LV + HV*16;
}

//Función para inicializar el DS1307.
void DS1307Inicio( void )
{
    unsigned short VAL[7], HV, LV, DATO;
    I2C1_Init(100000); //Inicio del bus I2C.
    delay_ms(50); //Retardo.
    //Lectura de las primeras 7 direcciones.
    DS1307SetDir(0);
    I2C1_Repeated_Start();
    I2C1_Wr(0xD1);
    VAL[0] = I2C1_Rd(1);
    VAL[1] = I2C1_Rd(1);
    VAL[2] = I2C1_Rd(1);
    VAL[3] = I2C1_Rd(1);
    VAL[4] = I2C1_Rd(1);
    VAL[5] = I2C1_Rd(1);
    VAL[6] = I2C1_Rd(0);
    I2C1_Stop();
    delay_ms(50); //Retardo.
    //Validación y corrección de información,
    //como hora y fecha.
    DATO = BcdToShort( VAL[0] );
    if( DATO > 59 )VAL[0]=0;
    DATO = BcdToShort( VAL[1] );
    if( DATO>59 )VAL[1]=0;
    DATO = BcdToShort( VAL[2] );
    if( DATO>23 )VAL[2]=0;
    DATO = BcdToShort( VAL[3] );
    if( DATO>7 || DATO==0 )VAL[3]=1;
    DATO = BcdToShort( VAL[4] );
    if( DATO>31 || DATO==0 )VAL[4]=1;
    DATO = BcdToShort( VAL[5] );
    if( DATO>12 || DATO==0 )VAL[5]=1;
    DATO = BcdToShort( VAL[6] );
    if( DATO>99 )VAL[6]=0;
    //Grabación de las primeras 7 direcciones.
    DS1307SetDir(0);
}

```

```

I2C1_Wr(VAL[0]);
I2C1_Wr(VAL[1]);
I2C1_Wr(VAL[2]);
I2C1_Wr(VAL[3]);
I2C1_Wr(VAL[4]);
I2C1_Wr(VAL[5]);
I2C1_Wr(VAL[6]);
I2C1_Wr(0x10); //Se activa la salida oscilante 1Hz.
I2C1_Stop();
delay_ms(50); //Retardo.
}

//Función para grabar la hora minutos y segundos.
void DS1307SetHora( Hora h )
{
    DS1307SetDir(0);
    I2C1_Wr( ShortToBcd(h.Seg) );
    I2C1_Wr( ShortToBcd(h.Min) );
    I2C1_Wr( ShortToBcd(h.Hor) );
    I2C1_Stop();
}

//Función para grabar el día, fecha, mes, y año.
void DS1307SetFecha( Fecha f )
{
    DS1307SetDir(3);
    I2C1_Wr( ShortToBcd(f.Dia) );
    I2C1_Wr( ShortToBcd(f.Fec) );
    I2C1_Wr( ShortToBcd(f.Mes) );
    I2C1_Wr( ShortToBcd(f.Ano) );
    I2C1_Stop();
}

//Función para leer la hora minutos y segundos.
Hora DS1307GetHora( void )
{
    Hora H;
    unsigned short VAL[3];
    DS1307SetDir(0);
    I2C1_Repeated_Start();
    I2C1_Wr(0xD1);
    VAL[0] = I2C1_Rd(1);
    VAL[1] = I2C1_Rd(1);
    VAL[2] = I2C1_Rd(0);
    I2C1_Stop();
    H.Seg = BcdToShort( VAL[0] );
    H.Min = BcdToShort( VAL[1] );
    H.Hor = BcdToShort( VAL[2] );
    return H;
}

```

```

}

//Función para leer el día, fecha, mes, y año.
Fecha DS1307GetFecha( void )
{
    Fecha F;
    unsigned short VAL[4];
    DS1307SetDir(3);
    I2C1_Repeated_Start();
    I2C1_Wr(0xD1);
    VAL[0] = I2C1_Rd(1);
    VAL[1] = I2C1_Rd(1);
    VAL[2] = I2C1_Rd(1);
    VAL[3] = I2C1_Rd(0);
    I2C1_Stop();
    F.Dia = BcdToShort( VAL[0] );
    F.Fec = BcdToShort( VAL[1] );
    F.Mes = BcdToShort( VAL[2] );
    F.Ano = BcdToShort( VAL[3] );
    return F;
}

// Funciones para leer y grabar datos individuales //
//Función para leer las horas.
unsigned short DS1307GetHoras( void )
{
    Hora h;
    h=DS1307GetHora();
    return h.Hor;
}
//Función para leer los minutos.
unsigned short DS1307GetMinutos( void )
{
    Hora h;
    h=DS1307GetHora();
    return h.Min;
}
//Función para leer los segundos.
unsigned short DS1307GetSegundos( void )
{
    Hora h;
    h=DS1307GetHora();
    return h.Seg;
}
//Función para grabar las horas.
void DS1307SetHoras( unsigned short ho )
{
    Hora h;
    h=DS1307GetHora();

```

```

    h.Hor = ho;
    DS1307SetHora( h );
}
//Función para grabar los minutos.
void DS1307SetMinutos( unsigned short mi )
{
    Hora h;
    h=DS1307GetHora();
    h.Min = mi;
    DS1307SetHora( h );
}

//Función para grabar los segundos.
void DS1307SetSegundos( unsigned short se )
{
    Hora h;
    h=DS1307GetHora();
    h.Seg = se;
    DS1307SetHora( h );
}

//Función para leer el día de la semana.
unsigned short DS1307GetDias( void )
{
    Fecha f;
    f=DS1307GetFecha();
    return f.Dia;
}

//Función para leer la fecha del mes.
unsigned short DS1307GetFechas( void )
{
    Fecha f;
    f=DS1307GetFecha();
    return f.Fec;
}

//Función para leer el mes del año.
unsigned short DS1307GetMeses( void )
{
    Fecha f;
    f=DS1307GetFecha();
    return f.Mes;
}

//Función para leer el año.
unsigned short DS1307GetAnos( void )
{
    Fecha f;

```

```

f=DS1307GetFecha();
return f.Ano;
}

//Función para grabar el dia de la semana.
void DS1307SetDias( unsigned short di )
{
    Fecha f;
    f=DS1307GetFecha();
    f.Dia = di;
    DS1307SetFecha(f);
}

//Función para grabar la fecha del mes.
void DS1307SetFechas( unsigned short fe )
{
    Fecha f;
    f=DS1307GetFecha();
    f.Fec = fe;
    DS1307SetFecha(f);
}

//Función para grabar el mes del año.
void DS1307SetMeses( unsigned short me )
{
    Fecha f;
    f=DS1307GetFecha();
    f.Mes = me;
    DS1307SetFecha(f);
}

//Función para grabar el año.
void DS1307SetAnos( unsigned short an )
{
    Fecha f;
    f=DS1307GetFecha();
    f.Ano = an;
    DS1307SetFecha(f);
}

```

Para realizar la simulación de un ejemplo con este tipo de reloj, se puede implementar la anterior librería y el siguiente código fuente correspondiente a la función *main*:

```

void main( void )
{
    //Declaración de variables.
    char Text[50];
    unsigned short DATO;
    //Inicio y configuración del reloj.

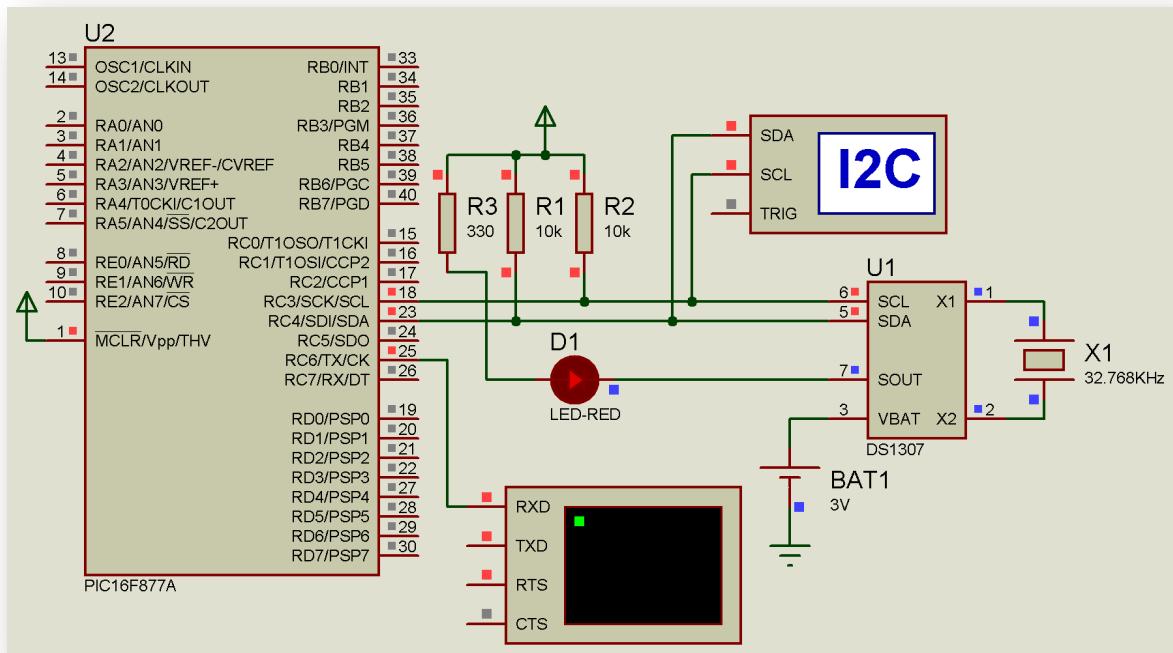
```

```

DS1307Inicio();
//Inicio y configuración de la USART a 9600 bps.
UART1_Init(9600);
//Se leen los segundos del reloj.
DATO = DS1307GetSegundos();
while(1) //Bucle infinito.
{
    //Se espera el cambio en los segundos.
    while( DATO == DS1307GetSegundos() )delay_ms(200);
    //Se leen la hora y se transmiten por la USART.
    DATO = DS1307GetHoras();
    ByteToStr( DATO, Text );
    UART1_Write_Text( Text ); UART1_Write_Text(":");
    //Se leen los minutos y se transmiten por la USART.
    DATO = DS1307GetMinutos();
    ByteToStr( DATO, Text );
    UART1_Write_Text( Text ); UART1_Write_Text(":");
    //Se leen los segundos y se transmiten por la USART.
    DATO = DS1307GetSegundos();
    ByteToStr( DATO, Text );
    UART1_Write_Text( Text ); UART1_Write(13); UART1_Write(10);
}
}

```

Para realizar la simulación pertinente en ISIS, se implementan los siguientes dispositivos: 16F877A, RES, LED-RED, DS1307, CRYSTAL, CELL, y los instrumentos virtuales, VIRTUAL TERMINAL, I2C DEBUGGER. El circuito electrónico correspondiente para simular es el siguiente:



Circuito 13-11

14 Tratamiento digital de señales

El tratamiento digital de señales es una línea de la tecnología demasiado extensa para ser tratada en un solo capítulo; sin embargo, este capítulo se centra en las nociones de adquisición, tratamiento y reconstrucción de señales. Cabe denotar que el alcance de los microcontroladores PICMicro de baja gama como los 16F y 18F sólo permiten realizar tratamientos sobre señales de baja frecuencia. De todos modos, muchas aplicaciones son de utilidad bajo estas características tales como sistemas de comunicación, generación de tonos DTMF y adquisición de señales bioeléctricas, entre otras. Si el desarrollador desea trabajar y tratar señales de mayor espectro, realizando tratamientos de mayor complejidad será necesario trabajar con unidades de procesamiento más robustas como la familia de microcontroladores PICMicro ds. Estos microcontroladores no pueden ser programados con el compilador MikroC PRO, para estos se debe utilizar el compilador MikroC PRO para dsPIC.

Un diagrama en bloques del procesamiento digital de una señal se puede apreciar en la siguiente figura:

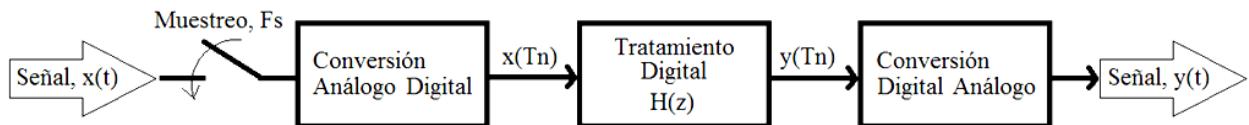


Figura 14-1

14.1 Muestreo

Como primera medida se requiere adquirir muestras periódicamente de la señal analógica, esto se logra haciendo una conversión analoga digital, con una frecuencia constante denominada frecuencia de muestreo F_s . El periodo de muestreo T , es el tiempo en segundos que existe entre muestra y muestra, y a su vez es el inverso de la frecuencia de muestreo $T = 1/F_s$. El muestreo permite convertir una señal continua en una señal discreta, es decir que pasa de $x(t)$ a $x(T_n)$, donde T es el periodo de muestreo, y n es el número de la muestra digitalizada. Para definir el periodo de muestreo T , o lo que es igual la frecuencia de muestreo F_s , se debe tener la siguiente consideración: La frecuencia de muestreo debe ser mayor que la máxima componente espectral contenida en la señal $x(t)$. Por ejemplo, si la máxima frecuencia contenida en la señal $x(t)$ es 500Hz, la frecuencia de muestreo debe ser como mínimo el doble, es decir $F_s = 2 \times 500\text{Hz} = 1000\text{Hz}$. Este concepto es conocido como teorema de muestreo de Nyquist. Sin embargo la frecuencia de muestreo puede ser mayor al doble del máximo en frecuencia de la señal $x(t)$, de hecho cuanto mayor sea la frecuencia de muestreo mayor será la fidelidad de la señal digitalizada, pero a su vez mayor serán los recursos requeridos en velocidad, y memoria de procesamiento.

$$F_s \geq 2F_{MAX} \quad \text{Ecuación 14-1}$$

14.2 Funciones de transferencia

Una función de transferencia es la relación que existe entre la entrada de una señal y su salida al pasar por el bloque de proceso. Las funciones de transferencia se estudian y se manipulan en términos de la frecuencia compleja y no en el dominio del tiempo continuo. Esto se debe a que al convertir las funciones a la frecuencia compleja las operaciones que implican manipulación de números complejos y sistemas integro diferenciales que se hacen lineales, y esto simplifica su tratamiento. Las funciones de transferencia para el caso de los sistemas de tiempo discreto se hacen en términos de la variable compleja z , para el tratamiento de estos sistemas se recurre a la transformación z , que representa la frecuencia compleja para el tiempo discreto. Las funciones de transferencia se denotan por excelencia con la letra H , para las funciones en términos de la variable z , y con h , para las funciones en términos del tiempo discreto T_n . A continuación se puede ver un ejemplo:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{A}{B} \quad \text{Ecuación 14-2}$$

De la ecuación anterior se puede deducir que:

$$H(z)X(z) = Y(z) \quad \text{Ecuación 14-3}$$

Como se puede ver en la anterior ecuación la salida Y , de un sistema discreto es igual a la entrada X , multiplicada por la función de transferencia H . Se debe recordar que el tratamiento real y la programación se realizan en términos del tiempo discreto, por esta razón siempre se deberá realizar la transformada inversa de z . La transformación inversa de la ecuación anterior da como resultado la siguiente ecuación en función del tiempo discreto T_n :

$$h(n) * x(n) = y(n) \quad \text{Ecuación 14-4}$$

De la anterior ecuación se puede observar que se omite la escritura del periodo T , dado que este es un valor constante, y no tiene relevancia escribirlo siempre. Por otro lado se puede observar que la función h y la señal x , no se multiplican como en la frecuencia compleja, en este caso al realizar la transformación su equivalencia es la convolución. Para el caso del tratamiento de señales, la longitud de $h(n)$, es finita, y el número máximo que asume n se denomina M , este a su vez determina el orden de la función de transferencia. Por otra parte la señal de entrada $x(n)$, es una serie de muestras de longitud indeterminada, igual que la salida $y(n)$.

De lo anterior se puede concluir que para realizar un tratamiento digital sobre una señal se debe realizar la operación de convolución continua entre los coeficiente de la función $h(n)$ y la señal $x(n)$. Sin embargo la operación de convolución puede ser de dos formas, una en la cual la convolución se hace con las muestras pasadas y actuales de la señal $x(n)$, esto se conoce como sistema sin memoria o sistemas FIR (Respuestá Finita al Impulso). Por otra parte cuando la convolución requiere información de las muestras pasadas de $x(n)$, y de $y(n)$, estos sistemas se denominan sistemas con memoria o IIR (Respuestá Infinita al Impulso). Los sistemas IIR son sistemáticamente más simples, en su implementación y requieren de campos de memoria pequeños, por otra parte realizar su diseño y lograr su estabilidad es más complicado y requiere de tediosos procesos matemáticos. Los sistemas FIR, ofrecen características contrarias a los sistemas IIR, los sistemas FIR, son de fácil implementación y diseño, pero consumen recursos de

memoria y procesamiento mayor, una de las virtudes de estos sistemas es que siempre son estables. Los análisis y ejemplos de este capítulo se concentrarán en los sistemas FIR e IIR.

14.3 Convolución

Para iniciar es importante conocer la estructura de una convolución continua en forma matemática y por medio de un segmento de código en lenguaje C:

$$y(n) = \sum_{k=-\infty}^{+\infty} [h(k)x(n-k)] \quad \text{Ecuación 14-5}$$

La ecuación anterior describe la forma general de la convolución, sin embargo se debe recordar que la longitud de la función $h(n)$, es finita y su máximo es M , por lo tanto la ecuación se puede reescribir de la siguiente forma:

$$y(n) = \sum_{k=0}^{M} [h(k)x(n-k)] \quad \text{Ecuación 14-6}$$

Cada vez que una muestra de la salida $y(n)$, es calculada por medio de la convolución, se requieren M , muestras de la señal $x(n)$, incluida la muestra actual, esto quiere decir que para hacer la convolución se debe tener presente la necesidad de un campo de memoria igual a M para guardar las últimas muestras durante el proceso.

A continuación se puede observar un ejemplo de cómo se implementar una convolución, en lenguaje C:

```
//Orden del sistema FIR de orden 17, M=17.
```

```
#define M 17
```

```
float x[M];
```

```
float h[M];
```

```
//Rutina para hacer la convolución.
```

```
float yn=0.0;
short k;
for( k=M-1; k>=1; k-- )x[n]=x[n-1];
x[0]=x0;
for( k=0; k<M; k++ )yn += h[k]*x[k];
```

14.4 Filtros FIR

El diseño de los filtros FIR, es relativamente simple y utiliza estructuras ya definidas para cada tipo de filtro. Los filtros pueden ser de cinco naturalezas: Pasa bajas, pasa altas, pasa bandas, rechaza banda, y multi banda. Para el tratamiento de los filtros se debe tener presente las siguientes consideraciones:

Los cálculos de los filtros se hacen en radianes y no en hercios.

La frecuencia de muestreo en hercios F_s , es equivalente a una frecuencia en radianes/segundo de 2π .

La máxima frecuencia tratada por los filtros es $F_s/2$, o π Radianes.

La relación que existe entre radianes y hercios es: $W = 2\pi F$, y $F = W/2\pi$.

Las frecuencias de corte de los filtros se denotan como W_c , y F_c .

La frecuencia de corte se calcula como $W_c = 2 \pi F_c / F_s$.

La banda de transición del filtro se denota como dW en radianes/segundo y dF en hercios.

El orden de los sistemas FIR se denota como M .

Se recomienda usar un número impar para M .

A continuación se estudiará la forma de diseño para cada uno de estos filtros.

14.4.1 Filtro Pasa bajas

Los filtros pasa bajas se caracterizan por tener una frecuencia de corte a partir de la cual las frecuencias inferiores son permitidas, y las frecuencias superiores son atenuadas. La función de transferencia $h(n)$, para este filtro es:

$$h(n) = \begin{cases} \frac{\sin(w_c n)}{\pi n}, & n \neq 0 \\ \frac{w_c}{\pi}, & n = 0 \end{cases}$$

Ecuación 14-7

$$\frac{-M}{2} < n < \frac{M}{2}$$

La respuesta espectral de este filtro de orden 7, en escala lineal es la siguiente:

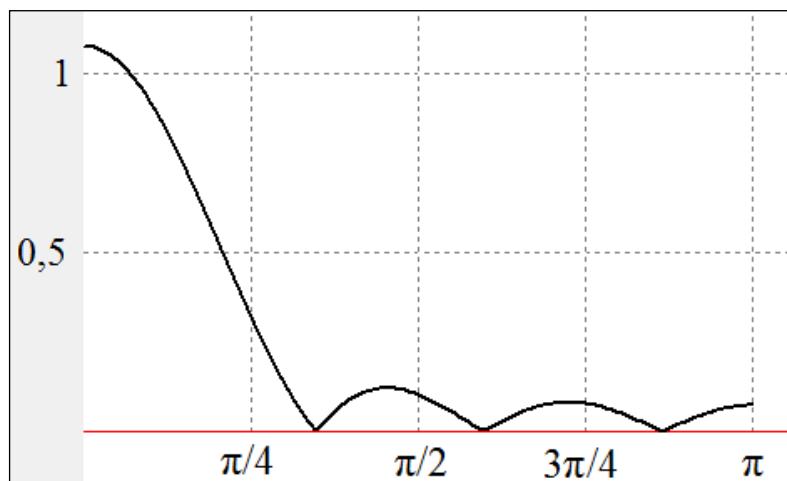


Figura 14-2

14.4.2 Filtros Pasa Altas

Los filtros pasa altos permiten el paso de las frecuencias superiores a la frecuencia de corte, y suprimen las inferiores a la misma. El cálculo de la función de transferencia $h(n)$, para estos filtros está definida por la siguiente ecuación:

$$h(n) = \begin{cases} \frac{-\operatorname{Sen}(w_c n)}{\pi n}, & n \neq 0 \\ 1 - \frac{w_c}{\pi}, & n = 0 \end{cases} \quad Ecuación 14-8$$

$$\frac{-M}{2} < n < \frac{M}{2}$$

La respuesta espectral en escala lineal de este filtro en orden 17, es la siguiente:

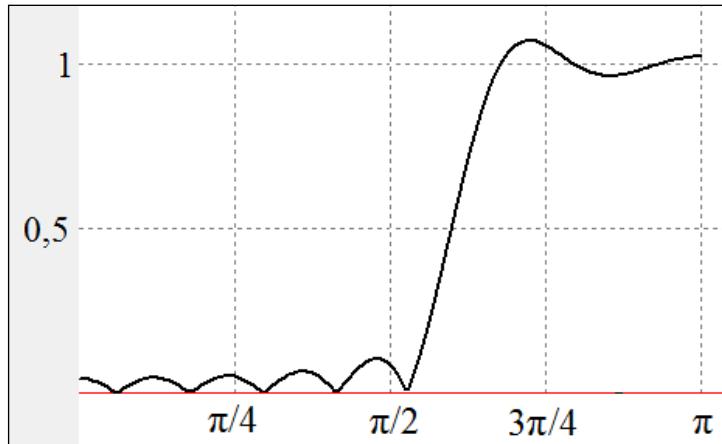


Figura 14-3

14.4.3 Filtro Pasa Banda

Los filtros pasa banda permiten el paso de una porción de frecuencias entre W_{c1} y W_{c2} , y el resto de frecuencias son eliminadas. La función de transferencia para este tipo de filtros se calcula por medio de la siguiente ecuación:

$$h(n) = \begin{cases} \frac{\operatorname{Sen}(w_{c2}n) - \operatorname{Sen}(w_{c1}n)}{\pi n}, & n \neq 0 \\ \frac{w_{c2} - w_{c1}}{\pi}, & n = 0 \end{cases} \quad Ecuación 14-9$$

$$\frac{-M}{2} < n < \frac{M}{2}$$

La respuesta espectral en escala lineal de este filtro en orden 17, es la siguiente:

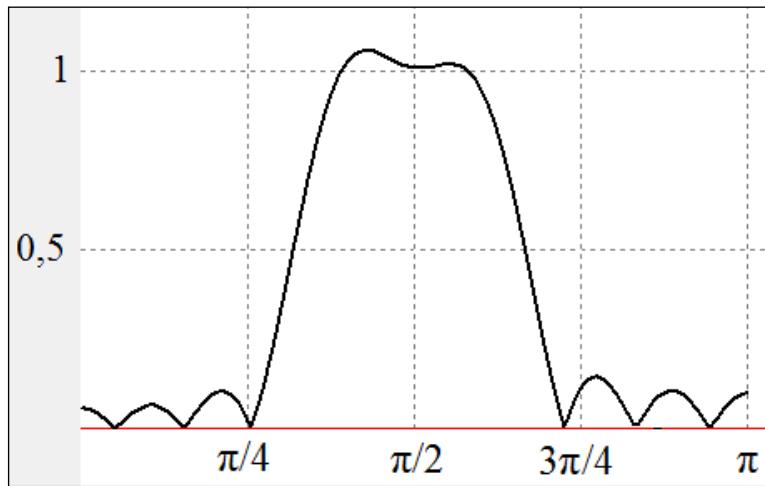


Figura 14-4

14.4.4 Filtro Rechaza Banda

Los filtros rechaza banda tienen un comportamiento similar a los filtros pasa banda, y su proceder es inverso a los filtros pasa banda, la función que caracteriza los coeficientes de la función de transferencia es la siguiente:

$$h(n) = \begin{cases} \frac{\operatorname{Sen}(w_{c1}n) - \operatorname{Sen}(w_{c2}n)}{\pi n}, & n \neq 0 \\ 1 - \frac{w_{c2} - w_{c1}}{\pi}, & n = 0 \end{cases}$$

Ecuación 14-10

$$\frac{-M}{2} < n < \frac{M}{2}$$

La respuesta espectral en escala lineal de este filtro en orden 17, es la siguiente:

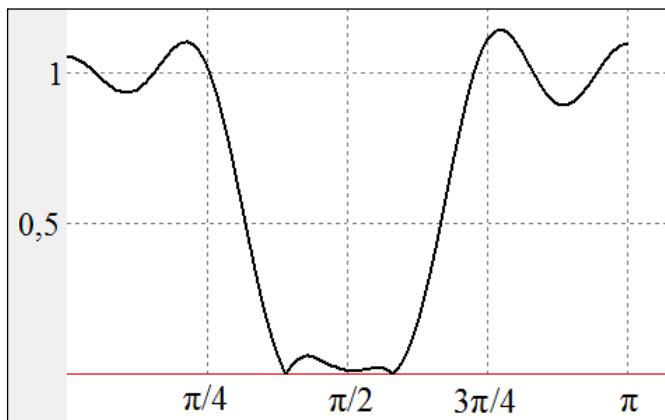


Figura 14-5

14.4.5 Filtro Multi Band

Los filtros multi banda son arreglos que integran diferentes filtros en uno solo, estos pueden dar ganancias arbitrarias en un rango de frecuencias. La siguiente ecuación integra las ganancias A del filtro, y las frecuencias de corte w:

$$h(n) = \begin{cases} \sum_{l=1}^L \left[(A_l - A_{l+1}) \frac{\sin(w_l n)}{\pi n} \right], & n \neq 0 \\ \sum_{l=1}^L \left[(A_l - A_{l+1}) \frac{w_l}{\pi} \right], & n = 0 \end{cases}$$

Ecuación 14-11

$$\frac{-M}{2} < n < \frac{M}{2}$$

La siguiente gráfica muestra la respuesta de un filtro multi banda de orden 65, con cuatro bandas diferentes:

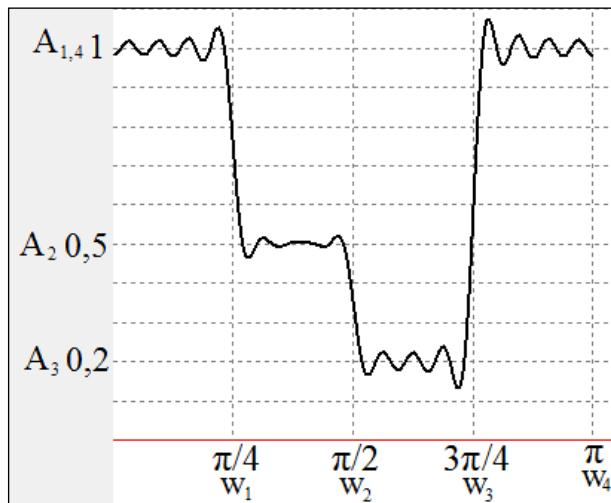


Figura 14-6

Para fines de diseño se debe tener presente que la ultima frecuencia calculada debe ser igual a π . Este tipo de filtros son ideales para el diseño de sistemas que requieran efectuar ecualización, sobre una señal.

14.4.6 Ventanas fijas

Las ventanas se aplican a las funciones de transferencia $h(n)$, el objetivo de las ventanas es mejorar y suavizar la respuesta espectral de los filtros FIR. Las ventanas de mayor uso son las siguientes:

- Rectangular
- Hamming
- Hanning
- Blackman

Los filtros que se demostraron anteriormente por defecto son filtros con ventana rectangular. Estos filtros tienen la menor transición en la frecuencia de corte, lo que los hace más cercanos a los filtros ideales, sin embargo esta propiedad produce en los filtros sobresaltos y oscilaciones en el espectro. Este efecto es conocido como: fenómeno Gibbs. Este efecto puede apreciarse en la siguiente gráfica:

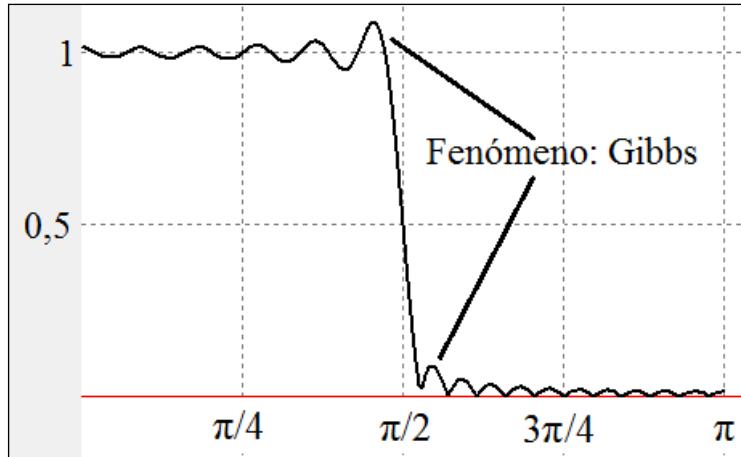


Figura 14-7

La ventana Rectangular ofrece una banda de transición igual a la siguiente relación: $dW = 1,8\pi/M$

14.4.7 Ventanas Rectangulares

Una ventana rectangular ofrece un patrón lineal, y constante. Se debe recordar que al realizar el cálculo de un filtro FIR, por defecto ya se encuentra con una ventana de este tipo.

La aplicación de las ventanas involucra la creación de una nueva función de factores $w(n)$, que posteriormente debe ser multiplicada término a término con la función de transferencia $h(n)$, para crear la función $h(n)$ definitiva.

Una ventana Rectangular se representa por medio de la siguiente función $w(n)$:

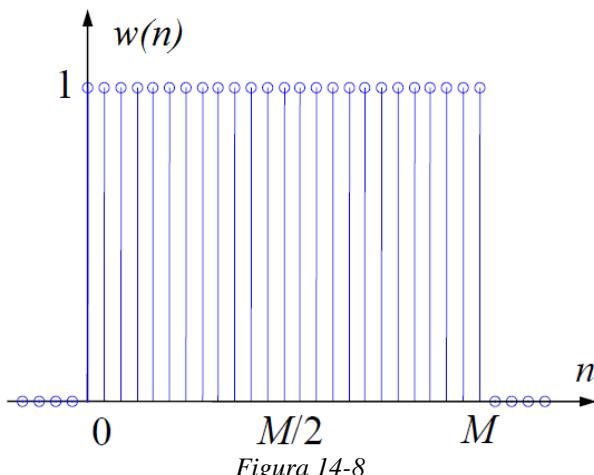


Figura 14-8

La respuesta espectral de la ventana Rectangular en escala logarítmica es la siguiente:

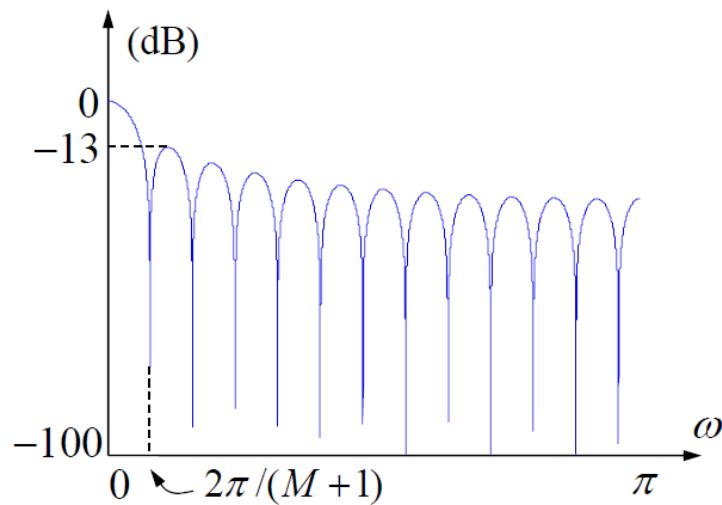


Figura 14-9

La ventana Rectangular ofrece una atenuación del fenómeno Gibbs de -13dB.

14.4.8 Ventanas Hamming

Para el caso de la ventana Hamming, los factores $w(n)$, se calculan de la siguiente forma:

$$w(n) = \frac{1 - \cos(2\pi n / M)}{2}, 0 \leq n \leq M \quad \text{Ecuación 14-12}$$

La misma respuesta espectral aplicando una ventana Hamming, en la figura (14.7) del fenómeno Gibbs se ve de la siguiente forma:

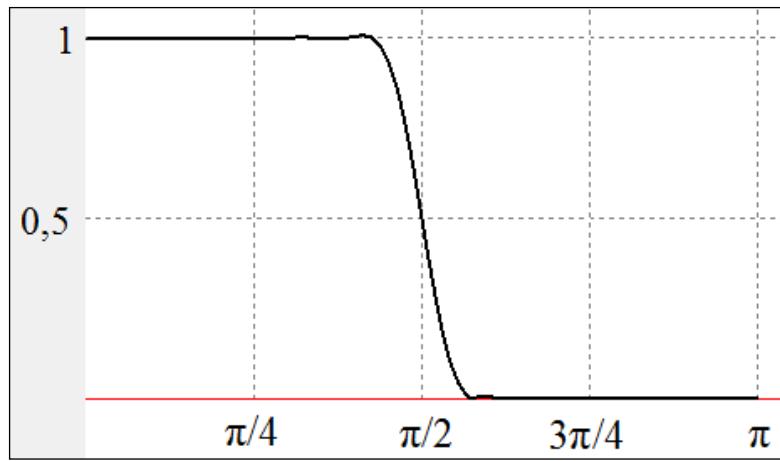


Figura 14-10

La ventana Hamming ofrece una banda de transición igual a la siguiente relación: $dW = 6,2\pi/M$

Una ventana Hamming se representa por medio de la siguiente función $w(n)$:

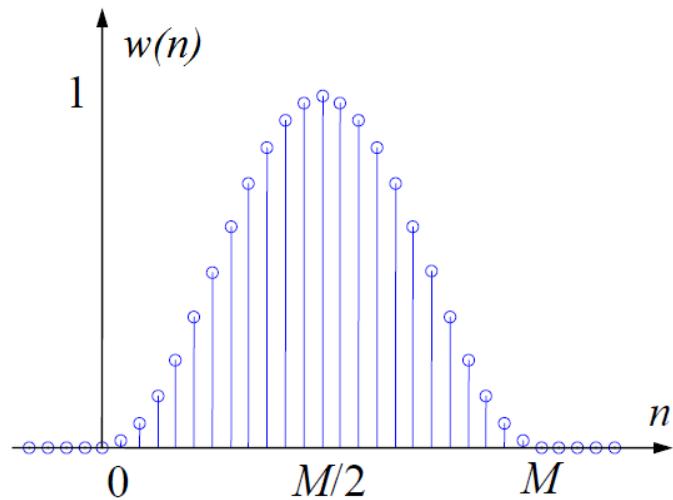


Figura 14-11

La respuesta espectral de la ventana Hamming en escala logarítmica es la siguiente:

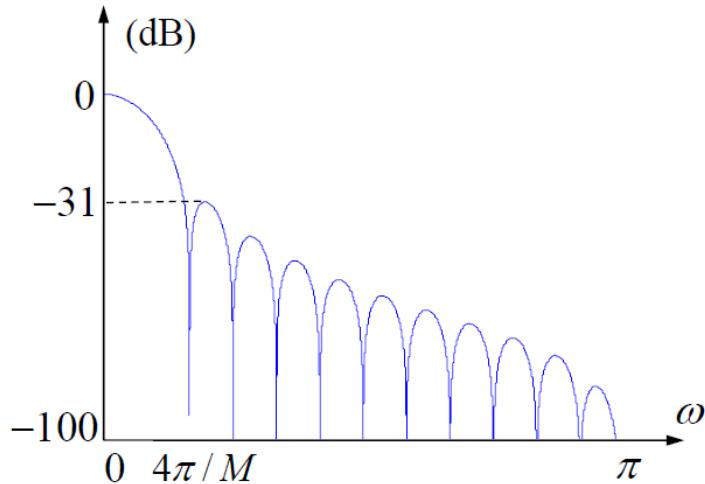


Figura 14-12

En la ventana Hamming se ofrece una atenuación del fenómeno Gibbs de -31dB.

14.4.9 Ventanas Hanning

Para el cálculo de la ventana Hanning, los factores $w(n)$, se deducen de la siguiente forma:

$$w(n) = \frac{27 - 23\cos(2\pi n/M)}{50}, 0 \leq n \leq M \quad \text{Ecuación 14-13}$$

La misma respuesta espectral aplicando una ventana Hanning, en la figura (14.7) del fenómeno Gibbs se ve de la siguiente forma:

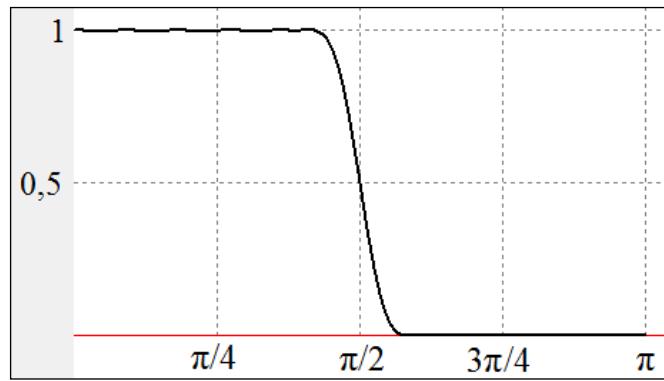


Figura 14-13

La ventana Hanning ofrece una banda de transición igual a la siguiente relación: $dW = 6,6\pi/M$

Una ventana Hanning se representa por medio de la siguiente función $w(n)$:

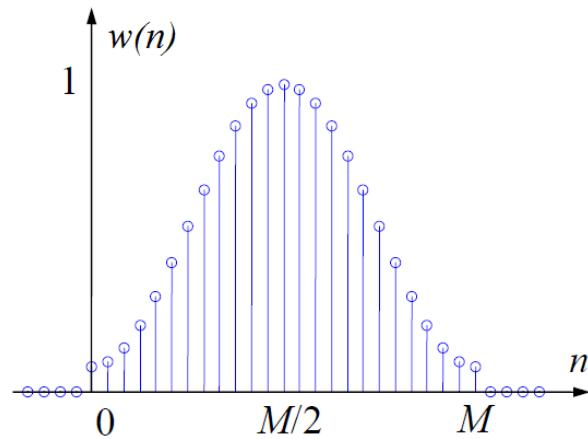


Figura 14-14

La respuesta espectral de la ventana Hanning en escala logarítmica es la siguiente:

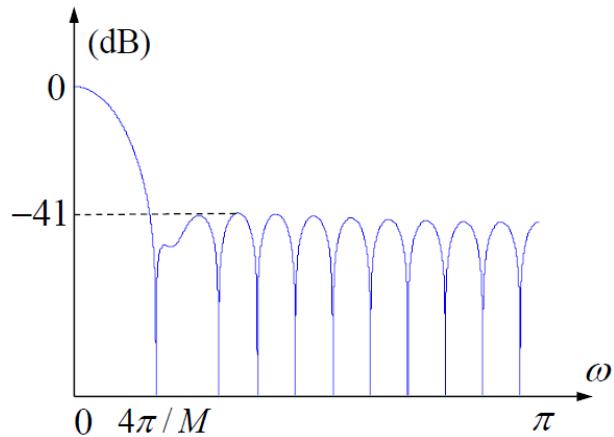


Figura 14-15

En la ventana Hanning se ofrece una atenuación del fenómeno Gibbs de -41dB.

14.4.10 Ventanas Blackman

Para el diseño de una ventana Blackman, los factores $w(n)$, se calculan de la siguiente forma:

$$w(n) = \frac{21 - 25\cos(2\pi n/M) + 4\cos(4\pi n/M)}{50}, 0 \leq n \leq M \quad \text{Ecuación 14-14}$$

La misma respuestapectral aplicando una ventana Blackman, en la figura (14.7) del fenómeno Gibbs se ve de la siguiente forma:

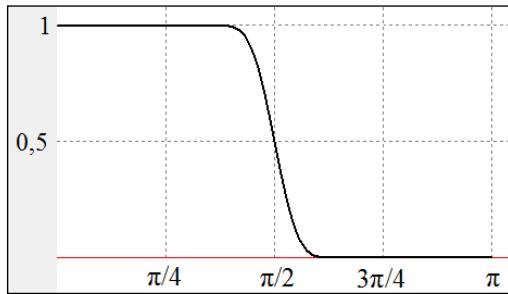


Figura 14-16

La ventana Blackman ofrece una banda de transición igual a la siguiente relación: $dW = 11\pi/M$

Una ventana Blackman se representa por medio de la siguiente función $w(n)$:

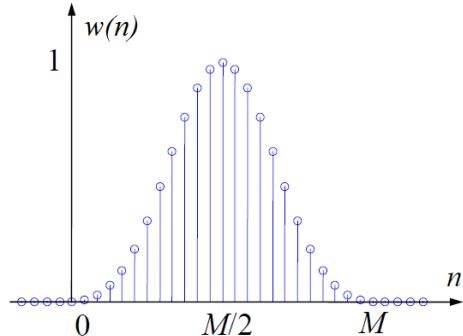


Figura 14-17

La respuestapectral de la ventana Blackman en escala logarítmica es la siguiente:

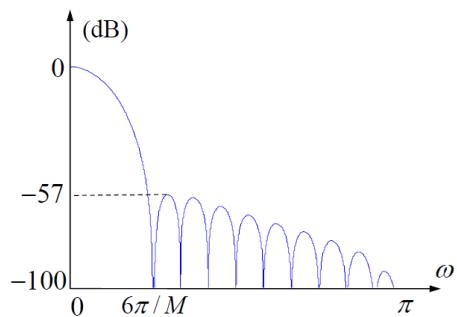


Figura 14-18

La ventana Blackman ofrece una atenuación del fenómeno Gibbs de -57dB.

14.5 Ejemplos de diseño para filtros FIR

En la siguiente sección se mostrarán ejemplos de diseño para filtros FIR, implementando un microcontrolador 18F452, con una fuente de 40MHz como reloj. Para fines prácticos reales la frecuencia de 40MHz, se logra utilizando un cristal de 10MHz, y activando la fuente de reloj HS-PLL en el PIC. Esta opción implementa internamente en el PIC, un PLL que multiplica la frecuencia externa por un factor de cuatro, y el resultado es usado como fuente de reloj para el procesador del microcontrolador. Para todos los ejemplos FIR que se mostrarán se deben simular con el mismo arreglo en ISIS, para este fin se implementa un circuito con los dispositivos 18F452, RES, OSCILLOSCOPE, Generador virtual Seno.

Para usar los generadores de señal virtual en ISIS, se debe picar en la barra de herramientas de la izquierda el icono de Generator Mode, este tiene la siguiente apariencia visual: , dentro de este se escoge la opción SINE, y se pega dentro del área de trabajo. Este generador tiene por defecto una frecuencia de 1Hz, y una amplitud de 1 voltio. La adquisición de señales se hace en el microcontrolador por medio del módulo AD, los niveles de tensión que las entradas análogas admiten, no pueden salirse de los confines de la polarización del microcontrolador. En otras palabras los niveles de las entradas análogas no pueden ser superiores a 5 voltios, ni voltajes negativos. Para evitar las circunstancias antes nombradas, se debe manipular la configuración del generador virtual de señal, para este propósito, se hace doble clic, sobre el generador y se editan los parámetros: Offset, y Amplitude. El parámetro offset se configura a 2,5 voltios, y el parámetro Amplitude a 2 voltios. Esta acción evita que las señales generadas se salgan de los parámetros de PIC. Para fines de simulación el parámetro Frequency, puede ser alterado al gusto del desarrollador.

Una vista de esta edición se puede apreciar en la siguiente figura:

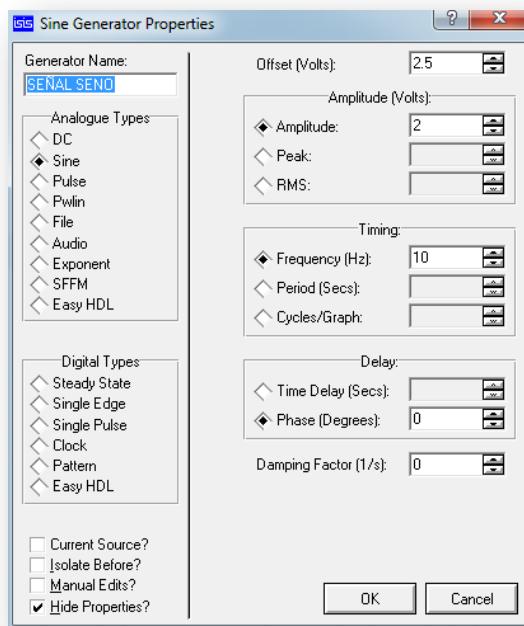
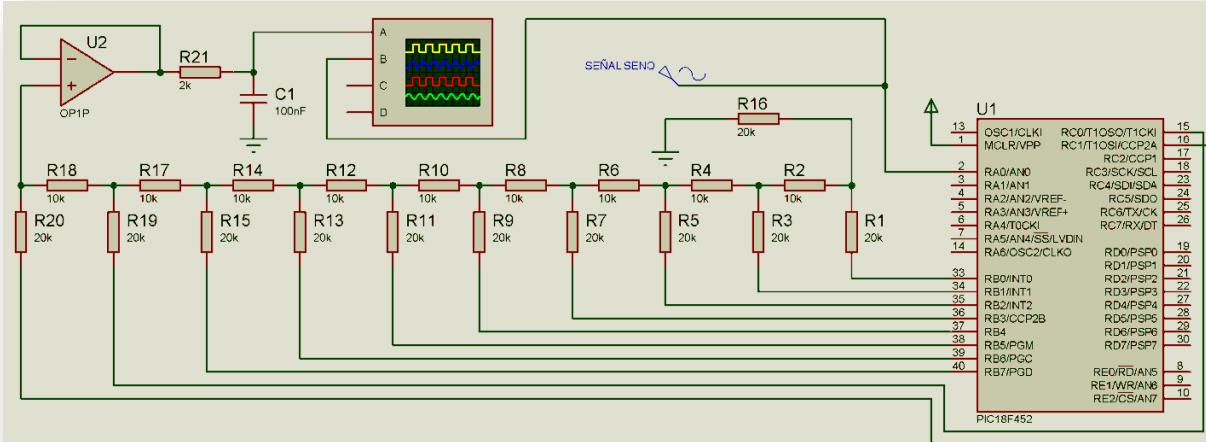


Figura 14-19

Indiferente al ejemplo que se simule en esta sección, se puede implementar el siguiente circuito electrónico en ISIS:



Circuito 14-1

Para la reconstrucción de la señal procesada, se configuran 10 bits de salida, para hacer un convertidor DA, por medio de un arreglo R-2R.

Para todos los ejemplos usados en este apartado, se usará la interrupción por TIMER 0, para crear el periodo de muestreo, y por defecto la frecuencia de muestreo.

El siguiente código fuente muestra un ejemplo del muestreo de la señal por medio del TIMER 0:

```
//Declaración de variables.
float x0, y0;
unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        //Adquisición de una muestra de 10 bits en, x[0].
        x0 = (float)(ADC_Read(0)-512.0);
        //
        //Espacio para procesar la señal.
        //
        //Reconstrucción de la señal: y en 10 bits.
        YY = (unsigned int)(x0+512);
        PORTC = (YY>>8)&3;
        PORTB = YY&255;
```

```

    INTCON.F2=0;
}
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

14.5.1 Ejemplo de diseño para filtro pasa bajas

Fs = 1291,32KHz.

Fc = 150Hz.

Ventana Rectangular.

Se determina la frecuencia de corte digital:

$$W_c = 2 \pi F_c / F_s = 2 \pi 150 / 1291,32 = 0,72985.$$

Usando la formula (14.7), se designan los coeficientes de la función h(n):

```

h(-8)=-0.0171035387965417
h(-7)=-0.0419431579233366
h(-6)=-0.0501329294124475
h(-5)=-0.0309497847516785
h(-4)=0.0175345019583181
h(-3)=0.0864308262744764
h(-2)=0.158173992108178
h(-1)=0.212237065988464
h(0)=0.232320416318186
h(1)=0.212237065988464
h(2)=0.158173992108178
h(3)=0.0864308262744764
h(4)=0.0175345019583181
h(5)=-0.0309497847516785
h(6)=-0.0501329294124475
h(7)=-0.0419431579233366
h(8)=-0.0171035387965417

```

Se debe tener presente que para fines de programación los valores de n no pueden ser negativos, por esta razón se debe iniciar el vector en 0, y para usarlo en el código en lenguaje C, se implementa de la siguiente manera:

```
const float h[]=
{
-0.0171035387965417, //h(0)
-0.0419431579233366, //h(1)
-0.0501329294124475, //h(2)
-0.0309497847516785, //h(3)
0.0175345019583181, //h(4)
0.0864308262744764, //h(5)
0.158173992108178, //h(6)
0.212237065988464, //h(7)
0.232320416318186, //h(8)
0.212237065988464, //h(9)
0.158173992108178, //h(10)
0.0864308262744764, //h(11)
0.0175345019583181, //h(12)
-0.0309497847516785, //h(13)
-0.0501329294124475, //h(14)
-0.0419431579233366, //h(15)
-0.0171035387965417 //h(16)
};
```

El programa definitivo con la función de transferencia h(n), será de la siguiente forma:

```
#define M 17
//Función de trasferencia h[n]
const float h[]=
{
-0.0171035387965417, //h(0)
-0.0419431579233366, //h(1)
-0.0501329294124475, //h(2)
-0.0309497847516785, //h(3)
0.0175345019583181, //h(4)
0.0864308262744764, //h(5)
0.158173992108178, //h(6)
0.212237065988464, //h(7)
0.232320416318186, //h(8)
0.212237065988464, //h(9)
0.158173992108178, //h(10)
0.0864308262744764, //h(11)
0.0175345019583181, //h(12)
-0.0309497847516785, //h(13)
-0.0501329294124475, //h(14)
-0.0419431579233366, //h(15)
-0.0171035387965417 //h(16)
};
```

```

//Declaración de variables.
float x0, y0;
float x[M];
unsigned int YY;
unsigned short i;
//Declaración de la función de interrupciones.
void interrupt (void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        PORTC.F7=1;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        //Corrimiento continuo de la señal x[n]
        for( i=M-1; i!=0; i-- ) x[i]=x[i-1];
        //Adquisición de una muestra de 10 bits en, x[0].
        x[0] = (float)(ADC_Read(0)-512.0);
        //Convolución continua.
        y0 = 0.0; for( i=0; i<M; i++ ) y0 += h[i]*x[i];
        //Reconstrucción de la señal: y en 10 bits.
        YY = (unsigned int)(y0+512.0);
        PORTC = (YY>>8)&3;
        PORTB = YY&255;
        PORTC.F7=0;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

Por limitaciones de velocidad en este microcontrolador, y teniendo presente la frecuencia de muestreo de 1291,32Hz no es posible usar un orden del filtro superior a 17. Sin embargo es posible usar órdenes menores como; 15, 13, 11, 9, 7, 5, o 3. Para fines prácticos y didácticos en los próximos ejemplos se implementarán filtros de orden 17 igual a este.

14.5.2 Ejemplo de diseño para filtro pasa altas

Fs = 1291,32KHz.

Fc = 200Hz.

Ventana Hamming.

Se determina la frecuencia de corte digital:

$$W_c = 2 \pi F_c / F_s = 2 \pi 200 / 1291,32 = 0,97314.$$

Usando la formula (14.8), se definen los coeficientes de la función $h(n)$ multiplicados por la ventana Hamming $w(n)$ de la ecuación (14.12), dando como resultado la siguiente función $h(n)$:

h(-8)=0
h(-7)=-0.000774581987451374
h(-6)=0.00297591407324525
h(-5)=0.0174357425540551
h(-4)=0.0246447931670207
h(-3)=-0.0148886974624375
h(-2)=-0.118648252092459
h(-1)=-0.243426834227261
h(0)=0.684363125797732
h(1)=-0.260893089082499
h(2)=-0.136977589378571
h(3)=-0.0187342667800403
h(4)=0.0345797805794857
h(5)=0.028555061237806
h(6)=0.00631989035694063
h(7)=-0.00299371636029323
h(8)=-0.00134023946307802

La implementación del código fuente es igual al filtro pasa bajas, sustituyendo el arreglo $h[]$, por el siguiente:

```
const float h[]=  
{  
    0, //h(0)  
    -0.000774581987451374, //h(1)  
    0.00297591407324525, //h(2)  
    0.0174357425540551, //h(3)  
    0.0246447931670207, //h(4)  
    -0.0148886974624375, //h(5)  
    -0.118648252092459, //h(6)  
    -0.243426834227261, //h(7)  
    0.684363125797732, //h(8)  
    -0.260893089082499, //h(9)  
    -0.136977589378571, //h(10)  
    -0.0187342667800403, //h(11)  
    0.0345797805794857, //h(12)  
    0.028555061237806, //h(13)
```

```

0.00631989035694063, //h(14)
-0.00299371636029323, //h(15)
-0.00134023946307802 //h(16)
};

```

14.5.3 Ejemplo de diseño para filtro pasa banda

Fs = 1291,32KHz.

Fc1 = 150Hz.

Fc2 = 400Hz.

Ventana Hanning.

Se determinan las frecuencias de corte digital:

$$W_{c1} = 2 \pi F_c / F_s = 2 \pi 150 / 1291,32 = 0,72985.$$

$$W_{c2} = 2 \pi F_c / F_s = 2 \pi 400 / 1291,32 = 1,94628.$$

Usando la formula (14.9), se definen los coeficientes de la función h(n) multiplicados por la ventana Hanning w(n) de la ecuación (14.13), dando como resultado la siguiente función h(n):

```

h(-8)=0.0018052104538582
h(-7)=0.00905810215732946
h(-6)=0.00179096961917994
h(-5)=0.00393014193876244
h(-4)=0.0307760790492056
h(-3)=-0.0879236273273945
h(-2)=-0.218010454414228
h(-1)=0.078115497545518
h(0)=0.384167993159943
h(1)=0.0832388331308223
h(2)=-0.248392736747743
h(3)=-0.107904878578323
h(4)=0.0411879382357919
h(5)=0.00583790841302624
h(6)=0.00299868100666665
h(7)=0.0163162389068139
h(8)=0.00250614601893693

```

La función h[], para implementar en el código fuente en lenguaje C es:

```

const float h[]=
{
    0.0018052104538582, //h(0)
    0.00905810215732946, //h(1)
    0.00179096961917994, //h(2)
    0.00393014193876244, //h(3)
    0.0307760790492056, //h(4)
    -0.0879236273273945, //h(5)
    -0.218010454414228, //h(6)
    0.078115497545518, //h(7)
}

```

```

0.384167993159943, //h(8)
0.0832388331308223, //h(9)
-0.248392736747743, //h(10)
-0.107904878578323, //h(11)
0.0411879382357919, //h(12)
0.00583790841302624, //h(13)
0.00299868100666665, //h(14)
0.0163162389068139, //h(15)
0.00250614601893693 //h(16)
};

```

14.5.4 Ejemplo de diseño para filtro rechaza banda

Fs = 1291,32KHz.

Fc1 = 150Hz.

Fc2 = 400Hz.

Ventana Blackman.

Se determinan las frecuencias de corte digital:

$$W_{c1} = 2 \pi F_c / F_s = 2 \pi 150 / 1291,32 = 0,72985.$$

$$W_{c2} = 2 \pi F_c / F_s = 2 \pi 400 / 1291,32 = 1,94628.$$

Usando la formula (14.10), se definen los coeficientes de la función h(n) multiplicados por la ventana Blackman w(n) de la ecuación (14.14), dando como resultado la siguiente función h(n):

```

h(-8)=3.13154095338657E-19
h(-7)=-0.00105084724932717
h(-6)=-0.000518135020348656
h(-5)=-0.00174730211401576
h(-4)=-0.0182611591144528
h(-3)=0.0645431455464317
h(-2)=0.186587834540544
h(-1)=-0.0738928265716166
h(0)=0.604271792109402
h(1)=-0.0827284691705043
h(2)=0.234965429330525
h(3)=0.0923521657912548
h(4)=-0.0302353209611255
h(5)=-0.00346395569934133
h(6)=-0.00133318382487159
h(7)=-0.00472035632958784
h(8)=-0.000290742652784183

```

La función h[], para implementar en el código fuente en lenguaje C es:

```

const float h[]=
{
    3.13154095338657E-19, //h(0)
    -0.00105084724932717, //h(1)

```

```

-0.000518135020348656, //h(2)
-0.00174730211401576, //h(3)
-0.0182611591144528, //h(4)
0.0645431455464317, //h(5)
0.186587834540544, //h(6)
-0.0738928265716166, //h(7)
0.604271792109402, //h(8)
-0.0827284691705043, //h(9)
0.234965429330525, //h(10)
0.0923521657912548, //h(11)
-0.0302353209611255, //h(12)
-0.00346395569934133, //h(13)
-0.00133318382487159, //h(14)
-0.00472035632958784, //h(15)
-0.000290742652784183 //h(16)
};


```

14.5.5 Ejemplo de diseño para filtro multi banda

Fs = 1291,32KHz.

Fc1 = 161Hz.

Fc2 = 322Hz.

Fc3 = 484Hz.

Fc4 = 645,66Hz.

A1 = 1.

A2 = 0,5.

A3 = 0,2.

A4 = 1.

Ventana Rectangular.

Se determinan las frecuencias de corte digital:

$$W_{c1} = 2 \pi F_{c1} / F_s = 2 \pi 161 / 1291,32 = 0,78337.$$

$$W_{c2} = 2 \pi F_{c2} / F_s = 2 \pi 322 / 1291,32 = 1,56675.$$

$$W_{c3} = 2 \pi F_{c3} / F_s = 2 \pi 484 / 1291,32 = 2,355.$$

$$W_{c4} = 2 \pi F_{c4} / F_s = 2 \pi 645,66 / 1291,32 = \pi.$$

Usando la formula (14.11), se definen los coeficientes de la función h(n):

```

h(-8)=-0.000403386658426763
h(-7)=-0.00443133542115946
h(-6)=-0.0685784954060509
h(-5)=0.0330418473673128
h(-4)=-0.000367826230314909
h(-3)=-0.0538949660023408
h(-2)=0.207286062892996
h(-1)=0.0275264614524777
h(0)=0.674596536876994
h(1)=0.0275264614524777
h(2)=0.207286062892996

```

```
h(3)=-0.0538949660023408  
h(4)=-0.000367826230314909  
h(5)=0.0330418473673128  
h(6)=-0.0685784954060509  
h(7)=-0.00443133542115946  
h(8)=-0.000403386658426763
```

La función h[], para implementar en el código fuente en lenguaje C es la siguiente:

```
const float h[]=  
{  
    -0.000403386658426763, //h(0)  
    -0.00443133542115946, //h(1)  
    -0.0685784954060509, //h(2)  
    0.0330418473673128, //h(3)  
    -0.000367826230314909, //h(4)  
    -0.0538949660023408, //h(5)  
    0.207286062892996, //h(6)  
    0.0275264614524777, //h(7)  
    0.674596536876994, //h(8)  
    0.0275264614524777, //h(9)  
    0.207286062892996, //h(10)  
    -0.0538949660023408, //h(11)  
    -0.000367826230314909, //h(12)  
    0.0330418473673128, //h(13)  
    -0.0685784954060509, //h(14)  
    -0.00443133542115946, //h(15)  
    -0.000403386658426763, //h(16)  
};
```

14.6 Filtros IIR

Los filtros IIR, son de implementación computacional más simple, pero su diseño y sus cálculos son de mayor complejidad. Para diseñar un filtro IIR, existen diferentes técnicas, sin embargo este capítulo se centrará en el diseño por medio de la transformación bilineal. La transformación bilineal es una relación que existe entre la variable compleja s , y z . Esto quiere decir que para realizar un filtro IIR, se requiere una función de transferencia en términos de la variable s . En síntesis para iniciar el diseño de un filtro IIR, se requiere como punto de partida, la función de transferencia de un filtro análogo. Para fines de estudio en este capítulo se mostrarán a continuación las funciones de transferencia de los filtros básicos de segundo orden:

Filtro Pasa Bajas:

$$H(s) = \frac{\Omega_c^2}{s^2 + 2\zeta\Omega_c s + \Omega_c^2} \quad \text{Ecuación 14-15}$$

Filtro Pasa Altas:

$$H(s) = \frac{s^2}{s^2 + 2\zeta\Omega_c s + \Omega_c^2} \quad \text{Ecuación 14-16}$$

Filtro Pasa Banda:

$$H(s) = \frac{s\Omega_B}{s^2 + \Omega_B s + \Omega_0^2} \quad \text{Ecuación 14-17}$$

Filtro Rechaza Banda:

$$H(s) = \frac{s^2 + \Omega_0^2}{s^2 + \Omega_B s + \Omega_0^2} \quad \text{Ecuación 14-18}$$

Las frecuencias Ω están definidas en radianes/segundo. Las frecuencias Ω_c representan la frecuencia de corte en los filtros pasa bajas, y altas. Las frecuencias Ω_0 representan la frecuencia de resonancia en los filtros pasa banda y rechaza banda. Las frecuencias Ω_B representan el ancho de banda de los filtros pasa banda y rechaza banda. El coeficiente ζ , representa el amortiguamiento del filtro, el valor que este coeficiente debe tomar es mayor que 0, cuando el coeficiente de amortiguamiento vale 0, el filtro se convierte en resonante, y tendría un comportamiento similar a los filtros pasa banda o rechaza banda. Por otra parte el coeficiente de amortiguamiento no podrá ser negativo, esto generaría una función de transferencia inestable. Cabe denotar que la implementación de filtros IIR, produce en la respuesta de la señal distorsiones mayores a las que se generan con los filtros FIR.

14.6.1 Transformación bilineal

La transformación bilineal es una relación matemática entre las variables complejas s, y z. Esta relación se rige por las siguientes ecuaciones:

$$s = \frac{2(1-z^{-1})}{T_s(1+z^{-1})} = \frac{2F_s(1-z^{-1})}{(1+z^{-1})} \quad \text{Ecuación 14-19}$$

En la ecuación (14.19), se puede observar la relación entre las dos variables, donde Ts, y Fs, son respectivamente el periodo de muestreo y frecuencia de muestreo implementado. De la misma forma existe una relación entre las frecuencias del filtro análogo y el filtro digital, esta relación se puede apreciar en la siguiente ecuación:

$$\Omega = \frac{2 \tan\left(\frac{\omega_d}{2}\right)}{T_s} = 2F_s \tan\left(\frac{\omega_d}{2}\right) \quad \text{Ecuación 14-20}$$

Donde la frecuencia digital es: Wd, la frecuencia análoga es: Ω, y Ts, Fs, son respectivamente el periodo de muestreo y la frecuencia de muestreo.

De la misma forma que en los filtros FIR, la relación entre la frecuencia digital y la frecuencia en hertzios es la siguiente:

$$\omega_d = \frac{2\pi F_x}{F_s} \Rightarrow \Omega_x = \frac{2 \tan\left(\frac{\pi F_x}{F_s}\right)}{T_s} = 2F_s \tan\left(\frac{\pi F_x}{F_s}\right) \quad \text{Ecuación 14-21}$$

Para contextualizar este proceso, se mostrará a continuación la transformación bilineal de los cuatro filtros básicos en segundo orden.

14.6.2 Filtro Pasa Bajas IIR

Como primera medida, se realiza el reemplazo de variables en la ecuación (14.15), para obtener la función h(z):

$$\begin{aligned} H(z) &= \frac{\Omega_c^2}{\left[\frac{2F_s(1-z^{-1})}{(1+z^{-1})} \right]^2 + 2\zeta\Omega_c \left[\frac{2F_s(1-z^{-1})}{(1+z^{-1})} \right] + \Omega_c^2} \\ H(z) &= \frac{\Omega_c^2(1+z^{-1})^2}{4F_s^2(1-z^{-1})^2 + 4\zeta\Omega_c F_s(1-z^{-1})(1+z^{-1}) + \Omega_c^2(1+z^{-1})^2} \\ H(z) &= \frac{\Omega_c^2(1+2z^{-1}+z^{-2})}{4F_s^2(1-2z^{-1}+z^{-2}) + 4\zeta\Omega_c F_s(1-z^{-2}) + \Omega_c^2(1+2z^{-1}+z^{-2})} \\ H(z) &= \frac{\Omega_c^2 + 2\Omega_c^2 z^{-1} + \Omega_c^2 z^{-2}}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2] + [2\Omega_c^2 - 8F_s^2]z^{-1} + [4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2]z^{-2}} \end{aligned}$$

En este punto se tiene una función de transferencia con polinomios ordenados en los polos y los ceros, de la forma $H(z) = A(z)/B(z)$. Para los polinomios A, B, se tiene en este caso expresiones de segundo orden con la forma: $A(z) = a_1z^0 + a_2z^{-1} + a_3z^{-2}$, $B(z) = b_1z^0 + b_2z^{-1} + b_3z^{-2}$.

Después de tener la función de transferencia de esta forma se debe implementar una ecuación en diferencias para la ejecución en la programación, para deducir la ecuación en diferencias es necesario forzar el coeficiente del denominador b_1 a valer 1. Para lograr este objetivo todos los coeficientes a y b se dividen en b_1 . Esta condición aplicada en la última ecuación del filtro pasa bajas, da como resultado lo siguiente:

$$H(z) = \frac{\frac{\Omega_c^2}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]} + \frac{2\Omega_c^2}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]} z^{-1} + \frac{\Omega_c^2}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]} z^{-2}}{1 + \frac{[2\Omega_c^2 - 8F_s^2]}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]} z^{-1} + \frac{[4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2]}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2]} z^{-2}}$$

La misma ecuación en términos de los coeficientes, es de la siguiente forma:

$$H(z) = \frac{a_1z^0 + a_2z^{-1} + a_3z^{-2}}{b_1z^0 + b_2z^{-1} + b_3z^{-2}},$$

$$a_1 = \frac{\Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

$$a_2 = 2a_1$$

$$a_3 = a_1$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

$$b_3 = \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

Para realizar la ecuación en diferencias se hace la siguiente transformación de z a n.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{A(z)}{B(z)} \Rightarrow Y(z)B(z) = X(z)A(z) \rightarrow y(n) * b(n) = x(n) * a(n)$$

Realizando la transformación inversa de z, se tiene una doble convolución, que genera una ecuación de la siguiente forma:

$$b_1y(n) + b_2y(n-1) + b_3y(n-2) = a_1x(n) + a_2x(n-1) + a_3x(n-2)$$

En conclusión la salida y(n) queda de la siguiente manera:

$$y(n) = a_1x(n) + a_2x(n-1) + a_3x(n-2) - b_2y(n-1) - b_3y(n-2)$$

14.6.3 Ejemplo de diseño para filtro pasa bajas

Fs = 1291.32Hz.

Fc = 100Hz.

$$\zeta = \frac{\sqrt{2}}{2}.$$

Para el caso particular de los filtros pasa bajas, y pasa altas de segundo orden el valor de ζ debe ser $\sqrt{2}/2$ para que la frecuencia de corte no sufra desplazamientos con respecto a las condiciones establecidas para los filtros en general.

Para deducir la frecuencia de corte análoga en función de las frecuencias reales se usa la ecuación (14.21), dando como resultado la siguiente frecuencia de corte:

$$\Omega_c = 2F_s \tan\left(\frac{\pi F_c}{F_s}\right) = 2(1291,32) \tan\left(\frac{100\pi}{1291,32}\right) = 641.0154$$

El paso siguiente es remplazar las constantes en las relaciones que determinan los coeficientes a, y b. Este paso se puede apreciar a continuación:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{\Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{(641,0154)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(641,0154)(1291,32) + (641,0154)^2} = 0,0436286852600961$$

$$a_2 = 2a_1 = 0,0872573705201923$$

$$a_3 = a_1 = 0,0436286852600961$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{2(641,0154)^2 - 8(1291,32)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(641,0154)(1291,32) + (641,0154)^2} = -1,32843480532316$$

$$b_3 = \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{4(1291,32)^2 - 4\left(\frac{\sqrt{2}}{2}\right)(641,0154)(1291,32) + (641,0154)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(641,0154)(1291,32) + (641,0154)^2} = 0,502949546363549$$

La implementación de un filtro IIR, de orden N, requiere un búfer de datos igual a: 2(N+1). En el caso de un filtro de segundo orden se requieren 3 campos para guardar los últimos valores de la entrada x, y 3 campos más para los 3 últimos valores de la salida y. Para ilustrar la implementación de este proceso se usará la misma arquitectura de software y hardware implementada en los filtros FIR. A continuación se puede observar el siguiente ejemplo en lenguaje C:

```
//Declaración de variables.
float x0=0, x1=0, x2=0, y0=0, y1=0, y2=0;
```

```

unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        //Fs = 1291,32 Hz.
        //Adquisición de una muestra de 10 bits en, x[0].
        x0 = (float)(ADC_Read(0)-512.0);
        //Implementación de la ecuación en diferencias.
        y0 = (x0+x2)*0.0436286852600961 + x1*0.0872573705201923 + y1*1.32843480532316
        -y2*0.502949546363549;
        //Corrimiento de los valores x(n), y y(n).
        y2 = y1;
        y1 = y0;
        x2 = x1;
        x1 = x0;
        //Reconstrucción de la señal: y en 10 bits.
        YY = (unsigned int)(y0+512.0);
        PORTC = (YY>>8)&3;
        PORTB = YY&255;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

14.6.4 Filtro Pasa Altas IIR

De la misma forma que se hizo en el filtro pasa bajas, se realiza el reemplazo de variable en la ecuación (14.16), para obtener la función h(z):

$$H(z) = \frac{\left[\frac{2F_s(1-z^{-1})}{(1+z^{-1})} \right]^2}{\left[\frac{2F_s(1-z^{-1})}{(1+z^{-1})} \right]^2 + 2\zeta\Omega_c \left[\frac{2F_s(1-z^{-1})}{(1+z^{-1})} \right] + \Omega_c^2}$$

$$H(z) = \frac{4F_s^2(1-z^{-1})^2}{4F_s^2(1-z^{-1})^2 + 4\zeta\Omega_c F_s(1+z^{-1})(1-z^{-1}) + \Omega_c^2(1+z^{-1})^2}$$

$$H(z) = \frac{4F_s^2(1-2z^{-1}+z^{-2})}{4F_s^2(1-2z^{-1}+z^{-2}) + 4\zeta\Omega_c F_s(1-z^{-2}) + \Omega_c^2(1+2z^{-1}+z^{-2})}$$

$$H(z) = \frac{4F_s^2 - 8F_s^2 z^{-1} + 4F_s^2 z^{-2}}{[4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2] + [2\Omega_c^2 - 8F_s^2]z^{-1} + [4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2]z^{-2}}$$

Realizando la normalización de los coeficientes se obtiene lo siguiente:

$$H(z) = \frac{\frac{4F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} - \frac{8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}z^{-1} + \frac{4F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}z^{-2}}{1 + \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}z^{-1} + \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}z^{-2}}$$

De esta forma los coeficientes a, y b quedan definidos así:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{4F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

$$a_2 = -2a_1$$

$$a_3 = a_1$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

$$b_3 = \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2}$$

14.6.5 Ejemplo de diseño para filtro pasa altas

Fs = 1291.32Hz.

Fc = 200Hz.

$$\zeta = \frac{\sqrt{2}}{2}.$$

Se determina la frecuencia de corte análoga en términos de la frecuencia digital:

$$\Omega_c = 2F_s \tan\left(\frac{\pi F_c}{F_s}\right) = 2(1291,32) \tan\left(\frac{200\pi}{1291,32}\right) = 1366,19404$$

Seguidamente se determinan los coeficientes del filtro:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{4F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{4(1291,32)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(1366,19404)(1291,32) + (1366,19404)^2} = 0,493018837784645$$

$$a_2 = -2a_1 = -0,986037675569289$$

$$a_3 = a_1 = 0,493018837784645$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_c^2 - 8F_s^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{2(1366,19404)^2 - 8(1291,32)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(1366,19404)(1291,32) + (1366,19404)^2} = -0,709951943923452$$

$$b_3 = \frac{4F_s^2 - 4\zeta\Omega_c F_s + \Omega_c^2}{4F_s^2 + 4\zeta\Omega_c F_s + \Omega_c^2} = \frac{4(1291,32)^2 - 4\left(\frac{\sqrt{2}}{2}\right)(1366,19404)(1291,32) + (1366,19404)^2}{4(1291,32)^2 + 4\left(\frac{\sqrt{2}}{2}\right)(1366,19404)(1291,32) + (1366,19404)^2} = 0,262123407215126$$

La programación de este filtro pasa altas, bajo las mismas condiciones del filtro pasa bajas es el siguiente:

```
//Declaración de variables.
float x0=0, x1=0, x2=0, y0=0, y1=0, y2=0;
unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        //Fs = 1291,32 Hz.
        //Adquisición de una muestra de 10 bits en, x[0].
        x0 = (float)(ADC_Read(0)-512.0);
        //Implementación de la ecuación en diferencias.
        y0 = (x0+x2)*0.493018837784645 - x1*0.986037675569289 +
        y1*0.709951943923452 - y2*0.262123407215126;
        //Corrimiento de los valores x(n), y y(n).
```

```

y2 = y1;
y1 = y0;
x2 = x1;
x1 = x0;
//Reconstrucción de la señal: y en 10 bits.
YY = (unsigned int)(y0+512.0);
PORTC = (YY>>8)&3;
PORTB = YY&255;
INTCON.F2=0;
}
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

14.6.6 Filtro Pasa Banda IIR

Para la función de transferencia $h(z)$ en el filtro pasa banda se realiza el cambio de variable bilineal, tomando como base la ecuación (14.17):

$$H(z) = \frac{\left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})} \right] \Omega_B}{\left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})} \right]^2 + \Omega_B^2 \left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})} \right] + \Omega_0^2}$$

$$H(z) = \frac{2F_s \Omega_B (1-z^{-1})(1+z^{-1})}{4F_s^2 (1-z^{-1})^2 + 2F_s \Omega_B (1-z^{-1})(1+z^{-1}) + \Omega_0^2 (1+z^{-1})^2}$$

$$H(z) = \frac{2F_s \Omega_B (1-z^{-2})}{4F_s^2 (1-2z^{-1}+z^{-2}) + 2F_s \Omega_B (1-z^{-2}) + \Omega_0^2 (1+2z^{-1}+z^{-2})}$$

$$H(z) = \frac{2F_s \Omega_B - 2F_s \Omega_B z^{-2}}{[4F_s^2 + 2F_s \Omega_B + \Omega_0^2] + [2\Omega_0^2 - 2F_s \Omega_B] z^{-1} + [4F_s^2 - 2F_s \Omega_B + \Omega_0^2] z^{-2}}$$

Normalizando la ecuación se obtiene la siguiente relación:

$$H(z) = \frac{\frac{2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} - \frac{2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}z^{-2}}{1 + \frac{2\Omega_0^2 - 2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}z^{-1} + \frac{4F_s^2 - 2F_s\Omega_B + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}z^{-2}}$$

De la anterior ecuación, se pueden establecer las siguientes constantes:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$a_2 = 0$$

$$a_3 = -a_1$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_0^2 - 2F_s\Omega_B}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$b_3 = \frac{4F_s^2 - 2F_s\Omega_B + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

14.6.7 Ejemplo de diseño para filtro pasa banda

$$F_s = 1291,32 \text{ Hz.}$$

$$F_o = 300 \text{ Hz.}$$

$$F_b = 10 \text{ Hz.}$$

A continuación se determinan las frecuencias análogas para el diseño del filtro pasa banda:

$$\Omega_0 = 2F_s \tan\left(\frac{\pi F_0}{F_s}\right) = 2(1291,32) \tan\left(\frac{300\pi}{1291,32}\right) = 2310,58$$

$$\Omega_B = 2F_s \tan\left(\frac{\pi F_b}{F_s}\right) = 2(1291,32) \tan\left(\frac{10\pi}{1291,32}\right) = 62,8442$$

Posteriormente se determinan los coeficientes del filtro:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{2F_s \Omega_B}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} = \frac{2(1291,32)(62,8442)}{4(1291,32)^2 + 2(1291,32)(62,8442) + (2310,58)^2} = 0.0133351841372129$$

$$a_2 = 0$$

$$a_3 = -a_1 = -0.0133351841372129$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_0^2 - 2F_s \Omega_B}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} = \frac{2(2310,58)^2 - 2(1291,32)(62,8442)}{4(1291,32)^2 + 2(1291,32)(62,8442) + (2310,58)^2} = -0.218755004098491$$

$$b_3 = \frac{4F_s^2 - 2F_s \Omega_B + \Omega_0^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} = \frac{4(1291,32)^2 - 2(1291,32)(62,8442) + (2310,58)^2}{4(1291,32)^2 + 2(1291,32)(62,8442) + (2310,58)^2} = 0.973329631725574$$

14.6.8 Filtro Rechaza Banda IIR

Para la función de transferencia $h(z)$ en el filtro pasa banda se realiza el cambio de variable bilineal, tomando como base la ecuación (14.17):

$$H(z) = \frac{\left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})}\right]^2 + \Omega_0^2}{\left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})}\right]^2 + \Omega_B^2 \left[2F_s \frac{(1-z^{-1})}{(1+z^{-1})}\right] + \Omega_0^2}$$

$$H(z) = \frac{4F_s^2(1-z^{-1})^2 + \Omega_0^2(1+z^{-1})^2}{4F_s^2(1-z^{-1})^2 + 2F_s \Omega_B (1-z^{-1})(1+z^{-1}) + \Omega_0^2(1+z^{-1})^2}$$

$$H(z) = \frac{4F_s^2(1-2z^{-1}+z^{-2}) + \Omega_0^2(1+2z^{-1}+z^{-2})}{4F_s^2(1-2z^{-1}+z^{-2}) + 2F_s \Omega_B (1-z^{-2}) + \Omega_0^2(1+2z^{-1}+z^{-2})}$$

$$H(z) = \frac{[4F_s^2 + \Omega_0^2] + [2\Omega_0^2 - 8F_s^2]z^{-1} + [4F_s^2 + \Omega_0^2]z^{-2}}{[4F_s^2 + 2F_s \Omega_B + \Omega_0^2] + [2\Omega_0^2 - 8F_s^2]z^{-1} + [4F_s^2 - 2F_s \Omega_B + \Omega_0^2]z^{-2}}$$

Normalizando la ecuación se obtiene la siguiente relación:

$$H(z) = \frac{\frac{4F_s^2 + \Omega_0^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2} + \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2}z^{-1} + \frac{4F_s^2 + \Omega_0^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2}z^{-2}}{1 + \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2}z^{-1} + \frac{4F_s^2 - 2F_s \Omega_B + \Omega_0^2}{4F_s^2 + 2F_s \Omega_B + \Omega_0^2}z^{-2}}$$

De la anterior ecuación, se pueden establecer las siguientes constantes:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{4F_s^2 + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$a_2 = \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$a_3 = a_1$$

$$b_1 = 1$$

$$b_2 = \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

$$b_3 = \frac{4F_s^2 - 2F_s\Omega_B + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2}$$

14.6.9 Ejemplo de diseño filtro rechaza banda

Fs = 1291,32Hz.

Fo = 100Hz.

Fb = 10Hz.

A continuación se determinan las frecuencias análogas para el diseño del filtro rechaza banda:

$$\Omega_0 = 2F_s \tan\left(\frac{\pi F_0}{F_s}\right) = 2(1291,32) \tan\left(\frac{100\pi}{1291,32}\right) = 641,015$$

$$\Omega_B = 2F_s \tan\left(\frac{\pi F_b}{F_s}\right) = 2(1291,32) \tan\left(\frac{10\pi}{1291,32}\right) = 62,8442$$

Posteriormente se determinan los coeficientes del filtro:

$$H(z) = \frac{a_1 z^0 + a_2 z^{-1} + a_3 z^{-2}}{b_1 z^0 + b_2 z^{-1} + b_3 z^{-2}},$$

$$a_1 = \frac{4F_s^2 + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} = \frac{4(1291,32)^2 + (641,015)^2}{4(1291,32)^2 + 2(1291,32)(62,8442) + (641,015)^2} = 0,977592319507735$$

$$a_2 = \frac{2\Omega_0^2 - 8F_s^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} = \frac{2(641,015)^2 - 8(1291,32)^2}{4(1291,32)^2 + 2(1291,32)(62,8442) + (641,015)^2} = -1.72826896958352$$

$$a_3 = a_1 = 0,977592319507735$$

$$b_1 = 1$$

$$b_2 = a_2 = -1.72826896958352$$

$$b_3 = \frac{4F_s^2 - 2F_s\Omega_B + \Omega_0^2}{4F_s^2 + 2F_s\Omega_B + \Omega_0^2} = \frac{4(1291,32)^2 - 2(1291,32)(62,8442) + (641,015)^2}{4(1291,32)^2 + 2(1291,32)(62,8442) + (641,015)^2} = 0.95518463901547$$

El código fuente en lenguaje C, correspondiente a este diseño es el siguiente:

```

//Declaración de variables.
float x0=0, x1=0, x2=0, y0=0, y1=0, y2=0;
unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt (void)
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        //Adquisición de una muestra de 10 bits en, x[0].
        x0 = (float)(ADC_Read(0)-512.0);
        //Implementación de la ecuación en diferencias.
        y0 = (x0+x2)*0.977592319507735 + (y1-x1)*1.72826896958352 - y2*0.95518463901547;
        //Corrimiento de los valores x(n), y y(n).
        y2 = y1;
        y1 = y0;
        x2 = x1;
        x1 = x0;
        //Reconstrucción de la señal: y en 10 bits.
        YY = (unsigned int)(y0+512.0);
        PORTC = (YY>>8)&3;
        PORTB = YY&255;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
}

```

```

PORTB = 0;
TRISC = 0;
PORTC = 0;
//Se configura el TIMER 0, su interrupción.
INTCON = 0b10100000;
T0CON = 0b11000101;
while(1)//Bucle infinito.
{
}
}

```

14.7 Osciladores digitales

Generar una señal seno de forma sintética, implica usar una matriz de rotación que incluye dos valores de salida de la señal y dos valores pasados, de la misma. La forma general de esta matriz se puede apreciar a continuación:

$$\begin{bmatrix} y(n) \\ x(n) \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} y(n-1) \\ x(n-1) \end{bmatrix}$$

Los cálculos de la matriz de rotación, permiten crear osciladores de 2 salidas que pueden tener amplitud equitativa o no y salida en cuadratura o no, así como el número de multiplicaciones requeridas por iteración. Estas condiciones se resumen en la siguiente tabla:

| Propiedades de los osciladores recursivos | | | | | |
|---|--------------------------------|---------------------|----------------------|-------------------|---|
| Oscilador | Multiplicaciones por iteración | Amplitud equitativa | Salida en cuadratura | Factores | Matriz de rotación |
| Doble cuadrado | 2 | Si | No | $k=2\cos(\theta)$ | $\begin{bmatrix} k & -1 \\ 1 & 0 \end{bmatrix}$ |
| De acople en cuadratura estándar | 4 | Si | Si | $k=\sin(\theta)$ | $\begin{bmatrix} \sqrt{1-k^2} & k \\ -k & \sqrt{1-k^2} \end{bmatrix}$ |

Tabla 14-1

Para iniciar el diseño de un oscilador o generador digital se debe definir el valor de la constante k, y por defecto, el valor del ángulo θ , este se define por medio de la siguiente ecuación:

$$\theta = \frac{2\pi F_g}{F_s} \quad \text{Ecuación 14-22}$$

Donde F_g es la frecuencia en hertzios de la señal que se desea generar, y F_s es la frecuencia de muestreo en hertzios. La implementación de estos generadores siempre usa la misma forma de operaciones matemáticas, que es la siguiente:

$$y(n) = ay(n-1) + bx(n-1)$$

$$x(n) = cy(n-1) + dx(n-1)$$

Para que el oscilador arranque correctamente es importante dar valores iniciales a las salidas y_1 , y y_2 . Por obvias razones los osciladores de mayor velocidad son los que solo requieren de una multiplicación por iteración.

14.7.1 Ejemplo de diseño oscilador doble cuadrado

$F_s = 1291,32\text{Hz}$.

$F_g = 100\text{Hz}$.

El valor del ángulo θ , es el que se calcula a continuación:

$$\theta = \frac{2\pi F_g}{F_s} = \frac{2\pi 100}{1291,32} = 0,4865707421$$

Con el valor θ se puede calcular el valor de k , que es el siguiente:

$$k=2\cos(0,4865707421) = 1,76788313$$

De esta manera se tiene la siguiente matriz de rotación:

$$\begin{bmatrix} 1,76788313 & -1 \\ 1 & 0 \end{bmatrix}$$

Las ecuaciones de implementación son las siguientes:

$$y(n) = 1,76788313y(n-1) + x(n-1)$$

$$x(n) = y(n-1)$$

Las mismas funciones en términos de solo la salida y es igual a la siguiente ecuación:

$$y(n) = 1,76788313y(n-1) + y(n-2)$$

La condición para el funcionamiento de este oscilador es que la magnitud de la señal debe ser 1. Otra condición para el funcionamiento adecuado de este oscilador es dar valores iniciales a las salidas $y(n-1)$, y $y(n-2)$, las cuales se pueden iniciar de la siguiente forma:

$$y(n-1) = \cos(-\theta) = \cos(-0,4865707421).$$

$$y(n-2) = \cos(-2\theta) = \cos(-2(0,4865707421)).$$

Para manipular la amplitud se debe multiplicar la salida $y(n)$, por el factor deseado.

Para fines de simulación, se implementarán las mismas condiciones del hardware usado en los filtros FIR, e IIR, omitiendo el generador de onda seno que se conecta a la entrada análoga. Teniendo presente las anteriores observaciones se puede implementar el siguiente código fuente en lenguaje C, para simular este ejemplo:

```

//Declaración de variables inicializadas.
float y0, y1=1.0, y2=0.883941565;
unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt (void)
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        // Fs = 1291,32 Hz.
        //Implementación de la ecuación del oscilador.
        y0 = 1.76788313*y1 - y2;
        //Se hace el corrimiento de la salida.
        y2 = y1;
        y1 = y0;
        //Reconstrucción de la señal: y en 10 bits, amplificada 500 veces.
        YY = (unsigned int)(500*y0+512.0);
        PORTC = (YY>>8)&3;
        PORTB = YY&255;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}

```

14.7.2 Ejemplo de diseño para oscilador de acople en cuadratura

Fs = 1291,32Hz.

Fg = 150Hz.

A = 450.

Se calcula el ángulo de paso θ :

$$\theta = \frac{2\pi F_s}{F_s} = \frac{2\pi 150}{1291,32} = 0,7298561132$$

Posteriormente se calculan las constantes de la matriz:

$$a = d = \sqrt{1 - [\operatorname{sen}(\theta)]^2} = \sqrt{1 - [\operatorname{sen}(0,7298561132)]^2} = 0,7452703484.$$

$$b = -c = \operatorname{sen}(\theta) = 0,6667624073.$$

El patrón de la matriz de rotación es el siguiente:

$$\begin{bmatrix} 0,7452703484 & 0,6667624073 \\ -0,6667624073 & 0,7452703484 \end{bmatrix}$$

El juego de ecuaciones que rigen las iteraciones del oscilador son las siguientes:

$$y(n) = 0,7452703484 y(n-1) + 0,6667624073 x(n-1)$$

$$x(n) = -0,6667624073 y(n-1) + 0,7452703484 x(n-1)$$

Para dar inicio correcto al oscilador se deben dar valores iniciales a las salidas retardadas $y(n-1)$, y $x(n-1)$, para este fin se debe tener presente que este oscilador tiene las dos salidas en cuadratura, es decir que entre las dos salidas siempre existe un desfase de 90 grados. Bajo este concepto se puede concluir que las dos señales en cuadratura tendrán la misma magnitud a $3\pi/4$, de esta manera se asignan los siguientes valores iniciales:

$$y(n-1) = x(n-1) = A \operatorname{Sen}(3\pi/4),$$

$$y(n-1) = x(n-1) = 450 \operatorname{Sen}(3\pi/4) = 318,1980515.$$

Finalmente se implementa, el código fuente en lenguaje C, para este oscilador:

```
//Declaración de variables inicializadas.
float y0, y1=318.1980515, x0, x1=318.1980515;
unsigned int YY;
//Declaración de la función de interrupciones.
void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        //Fs = 1291,32 Hz.
        //Implementación de las ecuaciones del oscilador.
        y0 = 0.7452703484*y1 + 0.6667624073*x1;
        x0 = -0.6667624073*y1 + 0.7452703484*x1;
        //Se hace el corrimiento de la salida.
```

```

y1 = y0;
x1 = x0;
//Reconstrucción de la señal: y en 10 bits.
YY = (unsigned int)(y0+512.0);
PORTC = (YY>>8)&3;
PORTB = YY&255;
INTCON.F2=0;
}
}

void main( void )
{
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    TRISC = 0;
    PORTC = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
    }
}
}

```

14.8 Transformada discreta de Fourier DFT

La transformación de Fourier es un procedimiento matemático que permite analizar las componentes espectrales y la intensidad de potencia de una señal análoga, o discreta. Las aplicaciones de este procedimiento son numerosas en las que se pueden nombrar: Analizadores de espectro, reconocimiento de patrones, en señales, de comunicaciones, biomédicas, y audio, entre otras. La transformada de Fourier, es una temática sumamente densa, sin embargo este capítulo busca mostrar de forma simple la implementación de esta herramienta. La transformada discreta de Fourier o DTF, está definida por la siguiente ecuación:

$$X(jw) = \sum_{n=0}^{L-1} x(n)e^{-jwn} \quad \text{Ecuación 14-23}$$

Como se puede apreciar en la anterior ecuación la potencia de una frecuencia w, se puede evaluar en un fragmento de señal de longitud L, de igual manera se puede notar que los valores resultantes de la función X(jw), son números complejos, por esta razón los análisis de Fourier se hacen por excelencia en términos de la magnitud de estos números complejos. La ecuación (15.23) en términos de un número complejo rectangular es la siguiente:

$$X(jw) = \sum_{n=0}^{L-1} x(n)[\cos(wn) - jsen(wn)] \quad \text{Ecuación 14-24}$$

De esta forma la relación X(jw), puede ser pasada a la magnitud de sí misma como |X(jw)|, y su relación matemática es la siguiente:

$$| X(jw) | = \sqrt{\left[\sum_{n=0}^{L-1} x(n) \cos(wn) \right]^2 + \left[\sum_{n=0}^{L-1} x(n) \sin(wn) \right]^2} \quad Ecuación\ 14-25$$

Esta ecuación matemática permite evaluar la potencia, o la intensidad de la componente w, dentro de la muestra x(n), de longitud L.

Un fragmento de código en lenguaje C, que realice esta operación se puede implementar con una función como la siguiente:

```
//Función para determinar la intensidad de potencia
//de la componente w, en la señal x_n.
float TDF(float *x_n, float w, unsigned int L )
{
    //Declaracion de variables.
    unsigned int n;
    float R=0.0, I=0.0;
    //Bucle for para realizar las sumatorias.
    for( n=0; n<L; n++ )
    {
        //Cálculo y sumatoria de los componentes
        //reales e imaginarios.
        R += cos( w*n );
        I += sin( w*n );
    }
    //Se retorna el valor de la magnitud del
    //número complejo.
    return sqrt( R*R + I*I );
}
```

El siguiente ejemplo implementa un programa que permite detectar la presencia de tres frecuencias diferentes en una señal analógica, por medio de la transformada de Fourier. Como primera medida se definen las tres frecuencias analógicas y sus respectivas frecuencias digitales, F1=100Hz, F2=200Hz, y F3=300Hz, la frecuencia de muestreo se define de 1291,32Hz, con el fin de implementar la misma estructura del microcontrolador 18F452, que se usó en los filtros FIR, e IIR.

$$W_1 = \frac{2\pi 100}{1291.32} = 0,48657707421$$

$$W_2 = \frac{2\pi 200}{1291.32} = 0,9731414842$$

$$W_3 = \frac{2\pi 300}{1291.32} = 1,459712226$$

Para realizar la transformada de Fourier, se usará la función antes citada. Una salida del microcontrolador se activa cuando se detecta la potencia suficiente de la componente espectral. La señal es adquirida por una entrada analógica del PIC, la cual es la suma de tres fuentes diferentes. El programa correspondiente para este ejercicio es el siguiente:

```

//Declaración de variables inicializadas.
float x[64];
unsigned short i, OK=1;
//Declaración de la función de interrupciones.

//Función para determinar la intensidad de potencia
//de la componente w, en la señal x_n.
float TDF(float *x_n, float w, unsigned int L )
{
    //Declaración de variables.
    unsigned short n;
    float R=0.0, I=0.0;
    //Bucle for para realizar las sumatorias.
    for( n=0; n<L; n++)
    {
        //Cálculo y sumatoria de los componentes
        //reales e imaginarios.
        R += x_n[n]*cos( w*n );
        I += x_n[n]*sin( w*n );
    }
    //Se retorna el valor de la magnitud del
//número complejo.
    return sqrt( R*R + I*I );
}

void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        //Fs = 1291,32 Hz.
        if( OK ) //Se evalúa la adquisición de muestras.
        {
            OK++; //Se cuentan las muestras tomadas.
            //Se corren las últimas 64 muestras en el bufer x.
            for( i=63; i!=0; i-- )x[i]=x[i-1];
            //Se guarda la última muestra.
            x[0] = ((float)ADC_Read(0)-512.0);
        }
        INTCON.F2=0;
    }
}

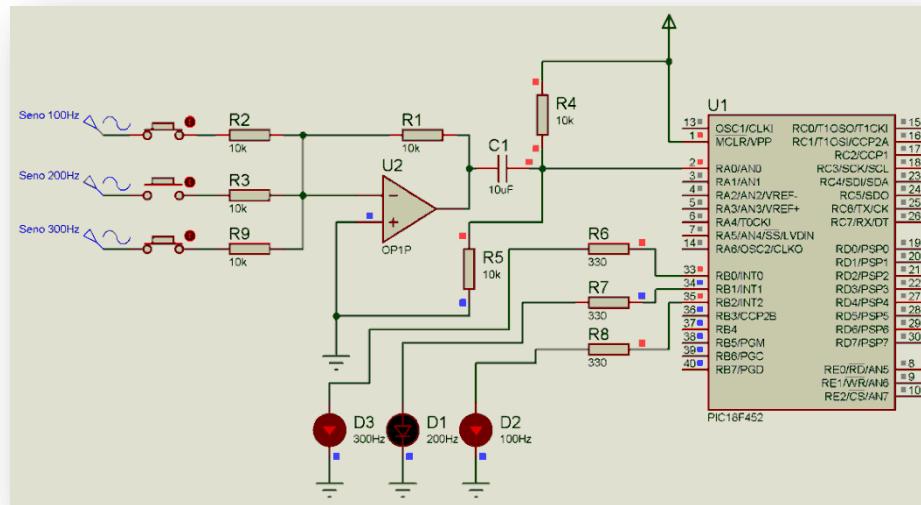
```

```

void main( void )
{
    float RES1, RES2, RES3;
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    while(1)//Bucle infinito.
    {
        while( OK<65 );//Se espera a recibir 64 muestras.
        OK =0; //Se suspenden la adquisición de muestras.
        //Se hace la transformada de Fourier, de la componente de 100Hz.
        RES1 = TDF( x, 0.4865707421, 64 );
        //Se hace la transformada de Fourier, de la componente de 200Hz.
        RES2 = TDF( x, 0.9731414842, 64 );
        //Se hace la transformada de Fourier, de la componente de 300Hz.
        RES3 = TDF( x, 1.459712226, 64 );
        OK=1; //Se activa la adquisición de muestras.
        //Se evalúa la potencia de la señal de 100Hz.
        if( RES1>800 )PORTB.F2=1; else PORTB.F2=0;
        //Se evalua la potencia de la señal de 200Hz.
        if( RES2>800 )PORTB.F1=1; else PORTB.F1=0;
        //Se evalúa la potencia de la señal de 300Hz.
        if( RES3>800 )PORTB.F0=1; else PORTB.F0=0;
    }
}

```

Para realizar la simulación de este ejercicio se implementa en ISIS, los dispositivos: 18F452, RES, CAP, OP1P, BUTTON, LED-RED, y generador virtual. El circuito correspondiente para esta simulación es el siguiente:



Circuito 14-2

La transformada de Fourier, implica múltiples cálculos matemáticos, como sumas, multiplicaciones y evaluaciones trigonométricas. Estos procesos hacen lento el rendimiento de la máquina de proceso. Por esta razón la ciencia de tratamiento de señales, se vio en la necesidad de buscar técnicas de realizar la transformada de Fourier de forma mucho más eficiente, de este esfuerzo, sale como resultado notable el algoritmo de transformación rápida de Fourier, o FFT. Este algoritmo se fundamenta en la división en segmentos cortos para realizar la evaluación de la transformada de Fourier.

14.8.1 Transformada rápida de Fourier FFT

La transformada rápida de Fourier, organiza en una matriz las muestras de entrada y realiza trasformadas más pequeñas en las filas y posteriormente en las columnas. Las muestras digitalizadas de la señal análoga se pueden representar de la siguiente forma:

$$x(0), x(1), x(2), x(3), x(4), x(5), \dots, x(N)$$

Donde N es el número máximo de muestras que se desean analizar.

Las muestras son organizadas en una matriz de L filas, con M columnas. En la siguiente matriz se puede apreciar el almacenamiento por columnas:

$$x(l, m) = \begin{bmatrix} x(0) & x(4) & x(8) & x(12) \\ x(1) & x(5) & x(9) & x(13) \\ x(2) & x(6) & x(10) & x(14) \\ x(3) & x(7) & x(11) & x(15) \end{bmatrix}$$

Seguidamente se debe realizar la transformación por filas, haciendo la transformada de M muestras. Esta operación se realiza en función de la siguiente ecuación:

$$\begin{aligned} F(l, q) &= \sum_{m=0}^{M-1} [x(l, m) W_M^{mq}] \\ 0 \leq l &\leq (L-1) \\ 0 \leq q &\leq (M-1) \end{aligned} \quad \text{Ecuación 14-26}$$

$$W_M^{mq} = e^{-j\frac{2\pi mq}{M}} = \cos\left(\frac{2\pi mq}{M}\right) - j\sin\left(\frac{2\pi mq}{M}\right)$$

La matriz F(l,q), es un arreglo de valores complejos correspondiente a las transformadas de M, puntos sobre cada una de las filas. Seguidamente la matriz F(l,q), se multiplica por los factores de fase Wn, está operación se define en la siguiente ecuación:

$$\begin{aligned} G(l, q) &= W_N^{lq} F(l, q) \\ 0 \leq l &\leq (L-1) \\ 0 \leq q &\leq (M-1) \end{aligned} \quad \text{Ecuación 14-27}$$

$$W_N^{lq} = e^{-j\frac{2\pi lq}{N}} = \cos\left(\frac{2\pi lq}{N}\right) - j\sin\left(\frac{2\pi lq}{N}\right)$$

Por último se realiza la trasformada por columnas de L muestras. Esta operación se resume en la siguiente ecuación:

$$X(p, q) = \sum_{l=0}^{L-1} [G(l, q) W_L^{lp}]$$

$$0 \leq p \leq (L-1)$$

$$0 \leq q \leq (M-1)$$

$$W_L^{lp} = e^{-j\frac{2\pi lp}{L}} = \cos\left(\frac{2\pi lp}{L}\right) - j\sin\left(\frac{2\pi lp}{L}\right)$$

Ecuación 14-28

La matriz $X(p,q)$, es la transformada final de las N muestras iniciales. Esta matriz contiene en cada uno de sus términos un número complejo que representa la magnitud y la fase de las componentes espectrales. La matriz $X(p,q)$, se debe leer por filas, como se aprecia en la siguiente matriz:

$$X(p, q) = \begin{bmatrix} X(0) & X(1) & X(2) & X(3) \\ X(4) & X(5) & X(6) & X(7) \\ X(8) & X(9) & X(10) & X(11) \\ X(12) & X(13) & X(14) & X(15) \end{bmatrix}$$

Cabe denotar que entre más muestras se ingresen para hacer la transformación, mayor será la resolución en el espectro, pero a su vez mayor será la cantidad de recursos de procesamiento y de memoria requeridos. A continuación se resume el proceso antes citado:

- Se archiva la señal de $x(n)$, por columnas en la matriz $x(l,m)$.
- Se calcula la transformada de M muestras de cada una de las filas.
- Se multiplican las anteriores transformadas por el factor de fase W_n .
- Se calcula la transformada de L muestras de cada una de las columnas.
- Se leen los datos de la matriz $X(p,q)$, por filas.

El resultado de una transformada de Fourier, entrega un espectro evaluado de 0 a 2π , lo que significa que los valores superiores a π , son un espejo de las componentes inferiores causadas por los armónicos de las frecuencias inferiores a π . En conclusión la lectura de la matriz solo se debe hacer hasta la mitad, por ejemplo si la cantidad de muestras analizadas son 128, solo se deben leer 64. De la misma manera la resolución de la transformación en hercios está definida por la siguiente ecuación:

$$R_{Hz} = \frac{F_s}{N}$$

Ecuación 14-29

Donde R, es la resolución en hercios, o el mínimo ancho de banda que se puede analizar con la FFT. Fs es la frecuencia de muestreo, y N es el número de muestras ingresadas a la FFT.

También cabe denotar que la primera muestra de la FFT, hace alusión a la componente espectral más baja incluida la frecuencia 0, o en otras palabras los niveles DC. Estás componentes son las más intensas, y para fines prácticos esta primera muestra puede ser ignorada.

Para realizar un ejemplo con la FFT, se implementará el PIC 18F4585, dada su amplia memoria RAM de 3328 Bytes, útil para los procesos de la FFT. Sin embargo está capacidad de memoria solo permite hacer una FFT, con máximo 169 muestras.

A continuación se presentará un ejemplo en lenguaje C, para realizar la transformación de una señal de $N = ML$, muestras:

```
#define M 16
#define L 8
#define N L*M
#define PI 3.141592654

typedef struct
{
    float R, I;
}Complejo;

//Declaración de variables inicializadas.

unsigned short i, OK=0;
unsigned short Fi=0, Co=0;
//Declaración de la función de interrupciones.

Complejo X[L][M];
Complejo Y[L][M];

//Función para realizar una multiplicación compleja.
Complejo Producto( Complejo A, Complejo B )
{
    Complejo Res;
    Res.R = A.R*B.R - A.I*B.I;
    Res.I = A.R*B.I + A.I*B.R;
    return Res;
}

//Función para realizar una suma compleja.
Complejo Adicion( Complejo A, Complejo B )
{
    Complejo Res;
    Res.R = A.R + B.R;
    Res.I = A.I + B.I;
    return Res;
}

//Función para determinar la magnitud de un número complejo.
float Magnitud( Complejo A )
{
    return sqrt( A.R*A.R + A.I*A.I );
}
```

```

//Cálculo rápido de Fourier, de N muestras.
void FFT( void )
{
    unsigned short l,q,m,p;
    Complejo WW;

    //Trío de bucles for, para ejecutar la
    //ecuación (15.26), la matriz Y[][][], representa F(l,q).
    for( l=0; l<L; l++ )
    {
        for( q=0; q<M; q++ )
        {
            Y[l][q].R = 0.0;
            Y[l][q].I = 0.0;
            for( m=0; m<M; m++ )
            {
                WW.R = cos( 2*PI*m*q / M );
                WW.I = -sin( 2*PI*m*q / M );
                WW = Producto( X[l][m], WW );
                Y[l][q] = Adicion( Y[l][q], WW );
            }
        }
    }

    //Dupla se bucles for, para realizar la multiplicación
    //de factores Wn (15.27), Y[][] representa la matriz F(l,q), y
    //X[][] representa la matriz G(l,q).
    for( l=0; l<L; l++ )
    {
        for( q=0; q<M; q++ )
        {
            WW.R = cos( 2*PI*l*q / N );
            WW.I = -sin( 2*PI*l*q / N );
            X[l][q] = Producto( Y[l][q], WW );
        }
    }

    //Trío de bucles for, para ejecutar la
    //ecuación (15.28), la matriz Y[][][], representa X(p,q),
    //y X[][] representa la matriz G(l,q).
    for( p=0; p<L; p++ )
    {
        for( q=0; q<M; q++ )
        {
            Y[p][q].R=0.0;
            Y[p][q].I=0.0;
            for( l=0; l<L; l++ )
            {

```

```

WW.R = cos( 2*PI*l*p / L );
WW.I = -sin( 2*PI*l*p / L );
WW = Producto( X[l][q], WW );
Y[p][q] = Adicion( Y[p][q], WW );
}
}
}

//Doble for anidado para determinar,
//la magnitud de la transformada,
//Este resultado queda en los términos reales de X[][]
for( l=0; l<L; l++ )
for( m=0; m<M; m++ )
{
    X[l][m].R = Magnitud( Y[l][m] );
    X[l][m].I = 0.0;
}

}

void interrupt ( void )
{
    if( INTCON.F2 )
    {
        TMR0L=135;
        //Timer0 con periodo de 774,4u segundo.
        //Fs = 1291,32 Hz.
        if( OK ) //Se evalúa la adquisición de muestras.
        {
            //Se hace la adquisición de las muestras,
            //y se archivan por columnas.
            X[Fi][Co].R = ((float)ADC_Read(0)-127.0);
            X[Fi][Co].I = 0.0;
            Fi++;
            if( Fi==L )
            {
                Fi = 0;
                Co++;
                if( Co==M )
                {
                    Co=0;
                    OK=0;
                }
            }
        }
        INTCON.F2=0;
    }
}

```

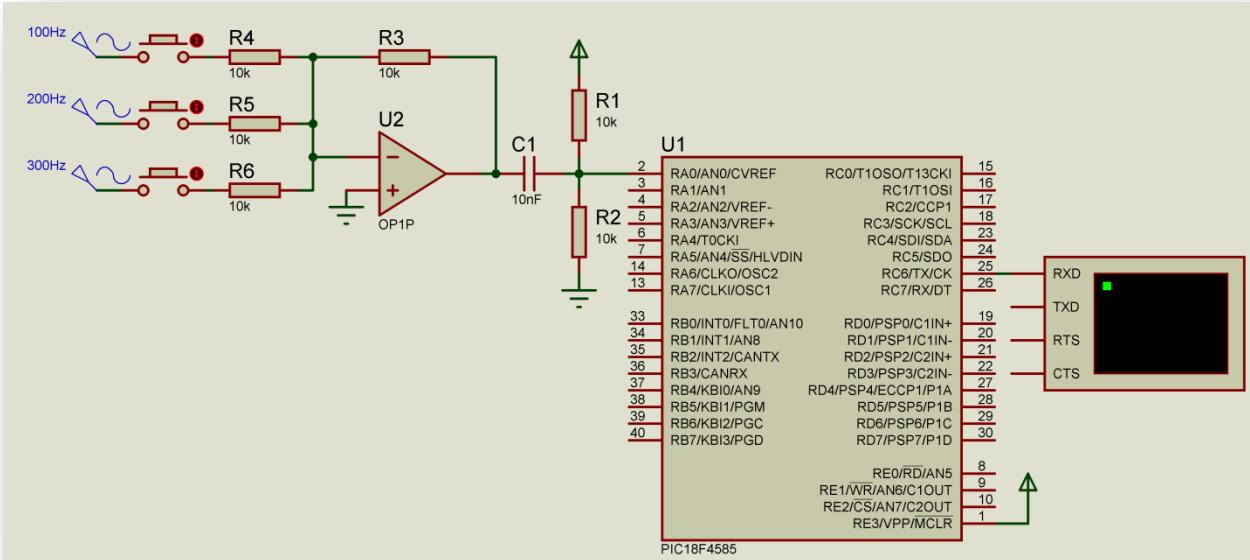
```

void main( void )
{
    char Text[30];
    unsigned short ff, cc, cont;
    //Inicio del puerto B como salida.
    TRISB = 0;
    PORTB = 0;
    //Se configura el TIMER 0, su interrupción.
    INTCON = 0b10100000;
    T0CON = 0b11000101;
    UART1_Init(9600);

    while(1)//Bucle infinito.
    {
        OK = 1; //Se habilita la lectura de muestras.
        UART1_Write_Text("// Adquisicion y FFT de:");
        IntToStr( N, Text );
        UART1_Write_Text(Text);
        UART1_Write_Text(" Muestras //");
        UART1_Write(13); UART1_Write(10);
        while( OK ); //Se espera la adquisicion de las muestras.
        FFT(); //Cálculo de la FFT.
        UART1_Write_Text("// Inicio de las muestras //");
        UART1_Write(13); UART1_Write(10);
        //La primera componente espectral se ignora y se hace igual a cero.
        X[0][0].R = 0.0;
        cont = 0;
        //Se envían la magnitud de la FFT, hasta N/2
        //por medio del puerto serial.
        for( ff=0; ff<L; ff++ )
        {
            for( cc=0; cc<M; cc++ )
            {
                FloatToStr( X[ff][cc].R, Text );
                UART1_Write_Text(Text);
                UART1_Write(13); UART1_Write(10);
                cont++;
                if( cont==N )
                {
                    cc = M;
                    ff = L;
                }
            }
        }
        UART1_Write_Text("// Fin de las muestras //");
        UART1_Write(13); UART1_Write(10);
    }
}

```

Para simular este ejercicio en ISIS, se implementa un circuito similar al ejercicio anterior de transformada discreta de Fourier. Sustituyendo el PIC, por un 18F4585, y anexando un VIRTUAL TERMINAL. El circuito para simular es el siguiente:



Circuito 14-3

14.9 Control digital PID

El en ámbito del control automático la forma de control más popular es el PID, que significa proporcional, integral, derivativo. El control es una rama de la física y la electrónica que permite manipular una variable física para permanecer en un valor deseado arbitrariamente. Todo sistema de control cuenta con las siguientes características:

- Punto de control
- Error
- Controlador (PID)
- Corrección
- Planta
- Variable a controlar
- Sensor o transductor

Estás características se pueden apreciar en el siguiente diagrama en bloques:

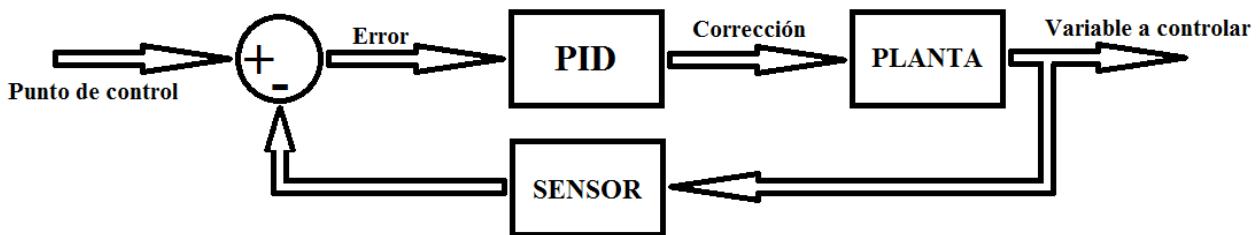


Figura 14-20

Para el caso del microcontrolador su función está sujeta a la entrada del punto de control, la lectura del sensor, y la salida de corrección. El punto de partida de un controlador es el controlador PID análogo que tiene una función de transferencia $PID(s)$, esta función tiene una relación similar a un conjunto de filtros y su tratamiento es equivalente a un filtro IIR. La función de transferencia análoga es la siguiente:

$$PID(s) = K_p + \frac{Ki}{s} + sKd \quad Ecuación\ 14-30$$

De la misma forma que se hace con los filtros IIR, se realiza la transformación bilineal, y este desarrollo da el siguiente análisis:

$$PID(z) = K_p + \frac{\frac{Ki}{2Fs(1-z^{-1})}}{(1+z^{-1})} + \frac{\frac{2FsKd(1-z^{-1})}{(1+z^{-1})}}{(1+z^{-1})}$$

$$PID(z) = K_p + \frac{\frac{Ki(1+z^{-1})}{2Fs(1-z^{-1})}}{(1+z^{-1})} + \frac{\frac{2FsKd(1-z^{-1})}{(1+z^{-1})}}{(1+z^{-1})}$$

$$PID(z) = K_p + \frac{\frac{Ki}{2Fs}(1+z^{-1})}{(1-z^{-1})} + \frac{\frac{2FsKd(1-z^{-1})}{(1+z^{-1})}}{(1+z^{-1})}$$

Del anterior análisis se pueden deducir tres funciones de transferencia para el controlador donde la entrada, es la señal de error para cada una de ellas. Las respectivas operaciones matemáticas para su implementación son las siguientes:

$$y_p(n) = [K_p]e(n)$$

$$y_i(n) = \frac{Ki}{2Fs}[e(n) + e(n-1)] + y_i(n-1) \quad Ecuación\ 14-31$$

$$y_d(n) = 2FsKd[e(n) - e(n-1)] - y_d(n-1)$$

$$y_{pid}(n) = y_p(n) + y_i(n) + y_d(n)$$

Para el caso particular del siguiente ejemplo se emplea la técnica de conversión digital análogo por medio de una salida PWM, y como estrategia para simular el comportamiento de la planta se usa un circuito RC, de segundo orden. El siguiente código fuente muestra un control PID correspondiente a la sintonización del control PID, cuando su K_p crítico es de 1,8 y un periodo crítico de 41,2m segundos.

//Declaración de constantes y variables.

`const float Kp=1.08, Ki=52.42718447, Kd=0.005562, Fs = 152.5878906;`

//Declaración de coeficientes de integración y derivación.

`const float Fi=ki/(2.0*Fs), Fd=2.0*Fs*Kd;`

```

float SENSOR, PUNTO_DE_CONTROL=127.0, YN=0.0;
int ADQUI;
unsigned short SALIDA;

float e0=0.0, e1=0.0, yi0=0.0, yi1=0.0, yd0=0.0, yd1=0.0, ypid=0.0;

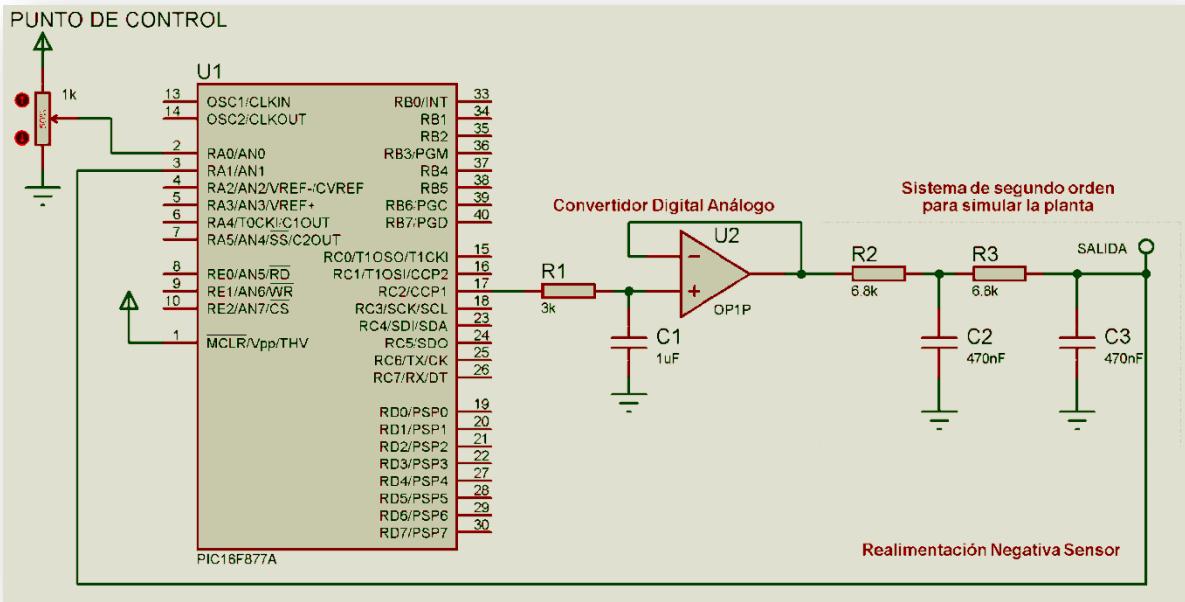
//Función de interrupciones para el Timer 0.
void interrupt()
{
    if( INTCON.F2 )// 6,5536ms :: 152,5878906 Hz
    {
        //Adquisición de la variable controlada.
        SENSOR = (float)((ADC_Read(1)>>2)&0xFF);
        //Adquisición del punto de control.
        PUNTO_DE_CONTROL = (float)((ADC_Read(0)>>2)&0xFF);
        //Calculo del nivel de error.
        e0 = PUNTO_DE_CONTROL - SENSOR;
        //Ecuación en diferencias.
        //Ecuación integral.
        yi0=Fi*(e0+e1)+yi1;
        //Ecuación derivativa.
        yd0=Fd*(e0-e1)-yd1;
        //Resultado PID.
        ypid=Kp*e0+yi0+yd0;
        //Ajuste y corrección de la SALIDA Y(n)
        //delimitada por los límites 0 y 255.
        YN += ypid;
        if(YN>255.0)YN=255.0;
        if(YN<0.0)YN=0.0;
        SALIDA = (unsigned short)(YN);
        PWM1_Set_Duty(SALIDA);
        //Actualización de muestras.
        e1=e0;
        yi1=yi0;
        yd1=yd0;
        INTCON.F2=0;
    }
}

void main( void )
{
    //Configuración del modulo PWM.
    PWM1_Init(10000);
    PWM1_Start();
    //Configuración de la interrupción de Timer 0.
    //a 6,5536ms
    OPTION_REG = 0b000000110;
    INTCON = 0b10100000;
    while( 1 )//Bucle infinito.
}

```

```
{
}
```

El circuito para simular implementa los dispositivos: 16F877A, POT-HG, RES, CAP, OP1P La simulación trabaja con reloj de 20MHz. El esquemático del circuito es el siguiente:



Circuito 14-4

Para la sintonía del controlador, y establecer las constantes K_p , K_i , y K_d , se puede usar la siguiente estrategia:

Primero se establece el controlador con las constantes K_i , y K_d igual a 0. Posteriormente se incrementa el valor de K_p , gradualmente hasta que la salida oscile de forma sostenida. Cuando esto sucede se mide el periodo de la oscilación sostenida y se determina el valor de K_p . Estos dos parámetros se denominan periodo crítico P_c , y K_p crítico K_{pc} . Posteriormente estos valores se remplazan en la siguiente tabla, para determinar los valores definitivos de K_p , K_i , y K_d :

| <i>Tipo de control</i> | <i>Kp</i> | <i>Ti</i> | <i>Td</i> |
|------------------------|--------------|---------------------------|------------|
| <i>P</i> | $0,5K_{pc}$ | <i>Infinito</i> , $K_i=0$ | 0 |
| <i>PI</i> | $0,45K_{pc}$ | $P_c/1,2$ | 0 |
| <i>PID</i> | $0,6K_{pc}$ | $0,5P_c$ | $0,125P_c$ |

Tabla 14-2

Donde, Ti es el periodo de integración y Td el periodo derivativo. La relación de estos periodos con las constantes K_i , y K_d se especifica en las siguientes ecuaciones:

$$Ki = \frac{Kp}{Ti}$$
$$Kd = TdKp$$

Ecuación 14-32

15 Transmisión de datos

La transmisión de datos es una línea de las comunicaciones electrónicas que se encarga de garantizar el transporte de información de forma digital. La información enviada puede ser bloques de texto, de audio, video, o cualquier otro tipo de información. Para realizar la transmisión de información se pueden seguir diversos formatos y protocolos, sin embargo este capítulo se centra en técnicas básicas de transmisión de datos como son, el control de flujo de la información, la transparencia de datos y el control de errores. Las técnicas de transmisión de datos permiten optimizar y garantizar el estado correcto de los datos durante una transmisión de datos. Este es un aspecto importante dado que un medio de transmisión está expuesto a producir errores por razones de ruido o falta de continuidad en el medio de transmisión de datos. El siguiente grafico ilustra de forma simple el tráfico de información entre dos puntos que gestionan una comunicación.

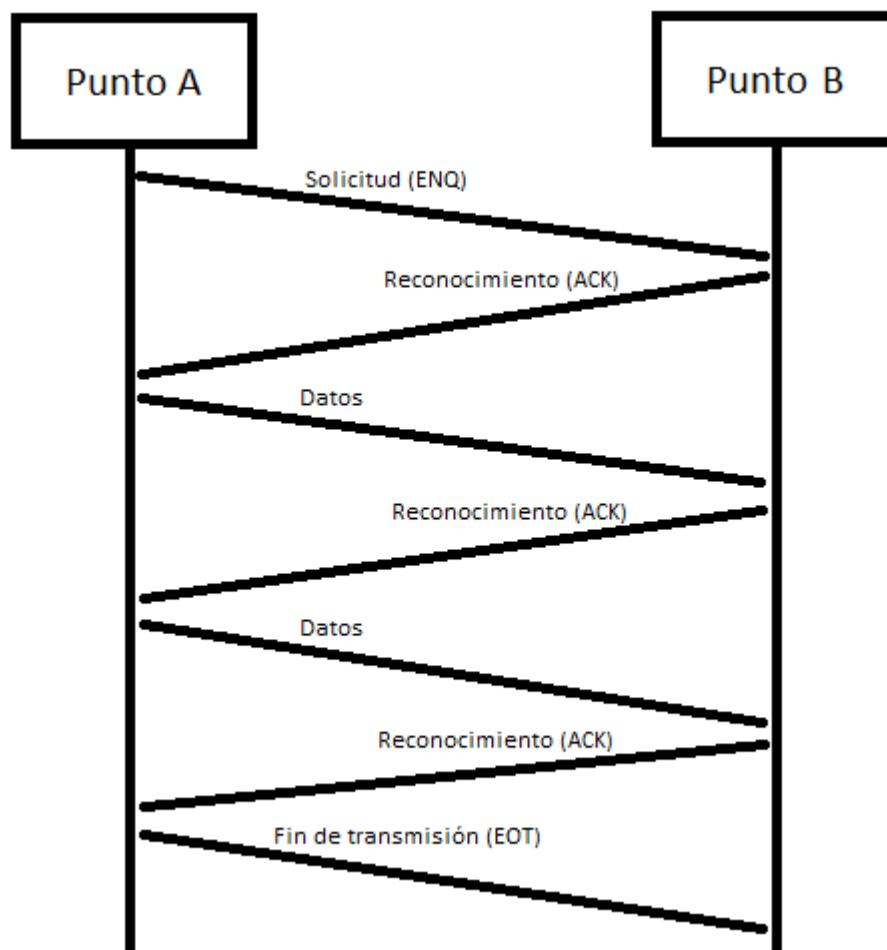


Figura 15-1

15.1 Control de flujo

El control de flujo permite gestionar el tráfico de información de forma ordenada. Cuando el medio de transmisión es altamente confiable, se puede implementar solamente el control de flujo. Por otro lado es importante tener presente que los ejemplos mostrados en este capítulo trican con la información en código ASCII, y el carácter de reconocimiento positivo ACK, se representa con el valor 0x06, el carácter de reconocimiento negativo NAK, se representa con el código 0x15, los finales de cadenas de texto o de caracteres se representan con el carácter nulo NULL que equivale al código 0x00. El siguiente ejemplo muestra una comunicación simple con el respectivo control de flujo, implementado con un microcontrolador PIC 16F628A, con oscilador de 4MHz.

```
// Declaración de constantes del LCD
sbit LCD_RS at RB0_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_RS_Direction at TRISB0_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D7_Direction at TRISB7_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB4_bit;

//Declaración de variables de trabajo.
char BUFER[20], Dato, Bandera=0;
unsigned short Estado=0, N=0;

//Declaración de constantes.
const char ACK='A', NAK='N', ENQ='&', EOT='%', NULL=0x00;

//Vector de interrupciones.
void interrupt()
{
    //Se evalúa la interrupción por recepción serial.
    if( PIR1.F5 )
    {
        //Se lee el Dato de entrada.
        Dato = UART1_Read();
        //Se evalúa el estado del tráfico de información.
        switch( Estado )
        {
            //En espera...
            case 0: //Se evalúa si llega una solicitud..
                if( Dato == ENQ )
                {
                    //Se cambia al siguiente Estado y se
```

```

    //envía el reconocimiento positivo.
    Estado = 1;
    N = 0;
    UART1_Write(ACK);
}
else
{
    //Se envía reconocimiento negativo.
    UART1_Write(NAK);
}
break;
//Recibiendo...
case 1: // Se evalúa si lo que llega es el final de transmisión EOT
    if( Dato == EOT )
    {
        //Llega el fin de la información y se vuelve
        //al Estado inicial y se activa la Bandera de datos
        //de llegada.
        BUFER[N] = NULL;
        Estado = 0;
        Bandera = 1;
    }
    else
    {
        //Se archiva el Dato recibido y se incrementa
        //el indicador N
        BUFER[N] = Dato;
        N++;
    }
}

default: break;
}
}
}

void main( void )
{
    //Se activan las interrupciones periféricas.
    INTCON = 0b11000000;
    //Se activa la interrupción por recepción serial.
    PIE1 = 0b00100000;
    //Configuración del modulo USART a 9600 bps.
    UART1_Init(9600);
    //Configuración del LCD.
    Lcd_Init();

    Lcd_Out(1,1,"Control de Flujo");

    while(1) //Bucle infinito.
}

```

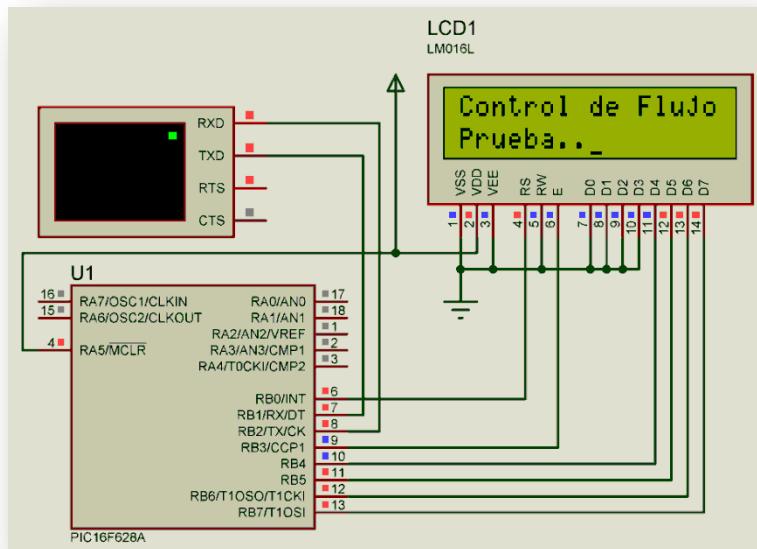
```

{
    //Se evalúa si la Bandera esta activa.
    if( Bandera == 1 )
    {
        //Se imprime la información en el LCD.
        Lcd_Out(2,1,"      ");
        Lcd_Out(2,1,BUFER);
        Bandera = 0;
    }
}
}

```

Es importante notar que en la declaración de constantes se usan como caracteres de ACK, NAK, ENQ y EOT, los caracteres A, N, &, y %, esto solo se realiza en este ejemplo de forma didáctica dado que este tipo de caracteres tienen valores y equivalencias que no se pueden ver ni digitar en el VIRTUAL TERMINAL.

Para realizar la respectiva simulación de este ejemplo se implementa en ISIS los dispositivos PIC16F628A, VIRTUAL TERMINAL, y LM016L, en un circuito como el siguiente:



Circuito 15-1

La transmisión de información obedece a la siguiente estructura:

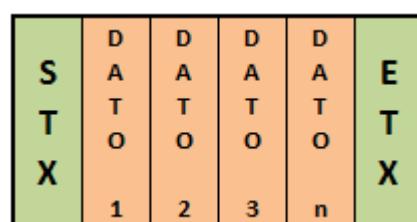


Figura 15-2

15.2 Transparencia de datos

La transparencia de datos es la capacidad de empaquetar información con un bloque de inicio o encabezado y un bloque final del paquete de información, el éxito de la transparencia esta en no repetir los datos de encabezado o final dentro de la información. De esta manera se puede ver de forma transparente la información dentro de los datos de control. Para encabezar la información se implementa el carácter DLE o marcador de transparencia equivalente al código 0x10, STX comienzo de texto que equivale al código 0x02, y para finalizar el bloque de datos se usa el carácter ETX que es equivalente a 0x03.

Para demostrar el funcionamiento de la estructura de transparencia se implementan a continuación dos programas uno que transmite información y otro que la recibe, los programas esta fundamentados en un microcontrolador PIC 16F628A, y un PIC 16F877A ambos con oscilador de 4MHz, a continuación se puede apreciar el código fuente del modulo receptor que es similar al ejemplo del control de flujo:

```
// Declaración de constantes del LCD
sbit LCD_RS at RB0_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_RS_Direction at TRISB0_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D7_Direction at TRISB7_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB4_bit;

//Declaración de variables de trabajo.
char BUFER[20], Dato, Bandera=0;
unsigned short Estado=0, N=0;

//Declaración de constantes.
const char ACK=0x06, NAK=0x15, ENQ=0x05, EOT=0x04, STX=0x02, ETX=0x03,
DLE=0x10,
NULL=0x00;

//Vector de interrupciones.
void interrupt()
{
    //Se evalúa la interrupción por recepción serial.
    if( PIR1.F5 )
    {
        //Se lee el Dato de entrada.
        Dato = UART1_Read();
        //Se evalúa el estado del tráfico de información.
        switch( Estado )

```

```

{
    //En espera...
    case 0: //Se evalúa si llega una solicitud..
        if( Dato == ENQ )
        {
            //Se cambia al siguiente Estado y se
            //envía el reconocimiento positivo.
            Estado = 1;
            UARTI_Write(ACK);
        }
        else
        {
            //Se envía reconocimiento negativo.
            UARTI_Write(NAK);
        }
        break;
    //Delimitador de transparencia...
    case 1: // Se evalúa si lo que llega es DLE.
        if( Dato == DLE )
        {
            //Se cambia al siguiente Estado...
            Estado = 2;
        }
        else
        {
            //Se identifica un error de orden en los datos
            //y se transmite un reconocimiento negativo y se
            //reanuda al Estado inicial.
            //Se reinicia de nuevo al Estado inicial.
            Estado = 0;
            UARTI_Write(NAK);
        }
        break;
    // Recibiendo información...
    case 2: // Se evalúa si lo que llega es STX.
        if( Dato == STX )
        {
            //Se inicia la cadena de información y se
            //pasa al Estado 3
            N=0;
            BUFER[N]=NULL;
            Estado = 3;
        }
        else
        {
            //Se identifica un error de orden en los datos
            //y se transmite un reconocimiento negativo y se
            //reanuda al Estado inicial.
            //Se reinicia de nuevo al Estado inicial.
        }
}

```

```

    Estado = 0;
    UART1_Write(NAK);
}
break;
// Esperando datos y delimitador de transparencia.
case 3: //Se evalúa si el Dato que llega es un DLE.
    if( Dato == DLE )
    {
        //Se cambia al Estado siguiente.
        Estado = 4;
        //Se finaliza el BUFER de datos;
        BUFER[N]=NULL;
    }
else
{
    //Se anexa un nuevo Dato al BUFER.
    BUFER[N]=Dato;
    //Se incrementa el índice de los datos.
    N++;
}
break;
// Esperando fin de cadena...
case 4: // Se evalúa si llega un ETX
    if( Dato == ETX )
    {
        //Se cambia al Estado siguiente.
        Estado = 5;
    }
else
{
    //Se cambia de nuevo al Estado inicial.
    Estado = 0;
    //Se transmite reconocimiento negativo.
    UART1_Write(NAK);
}
break;
//Esperando fin de transmisión...
case 5: //Se evalúa si llega el EOT...
    if( Dato == EOT )
    {
        //Se activa la Bandera de llegada de información.
        Bandera = 1;
        //Se transmite reconocimiento positivo.
        UART1_Write(ACK);
    }
else
{
    //Se transmite reconocimiento negativo.
    UART1_Write(NAK);
}

```

```

        }
        Estado = 0;
        break;

    default: break;
}
}

void main( void )
{
    //Se activan las interrupciones periféricas.
    INTCON = 0b11000000;
    //Se activa la interrupción por recepción serial.
    PIE1 = 0b00100000;
    //Configuración del modulo USART a 9600 bps.
    UART1_Init(9600);
    //Configuración del LCD.
    Lcd_Init();
    //Texto de inicio.
    Lcd_Out(1,1,"Transparencia");
    while(1) //Bucle infinito.
    {
        //Se evalúa si la Bandera esta activa.
        if( Bandera == 1 )
        {
            //Se imprime la información en el LCD.
            Lcd_Out(2,1,"          ");
            Lcd_Out(2,1,BUFER);
            Bandera = 0;
        }
    }
}

```

El siguiente programa está diseñado para un microcontrolador PIC 16F877A, el cual toma la lectura de un sensor LM35, y la transmite con el respectivo control de flujo y transparencia:

```

//Declaración de constantes.
const char ACK=0x06, NAK=0x15, ENQ=0x05, EOT=0x04, STX=0x02, ETX=0x03,
DLE=0x10, NULL=0x00;

void main( void )
{
    char BUFER[10];
    int adc;
    float Temp;
    //Se configura el modulo ADC, con el pin
    //AN3 como voltaje de referencia.
    ADCON1 = 0b11000001;

```

```

//Se inicia el modulo USART a 9600 bps.
UART1_Init(9600);
delay_ms(500);
while(1)//Bucle infinito.
{
    OTRO:
    //Lectura del canal 0
    adc = ADC_Read(0);
    //Se calcula el valor de la temperatura.
    Temp = 0.244*adc;
    //Se escribe la temperatura en el BUFER.
    FloatToStr( Temp, BUFER );
    //Retardo de lectura de 400m segundos
    delay_ms(100);
    //Se transmite el ENQ...
    UART1_Write(ENQ);
    //Se espera el ACK...
    while( !UART1_Data_Ready() );
    //Se verifica si llega un ACK...
    if( UART1_Read() != ACK )goto OTRO;
    //Se transmite un DLE...
    UART1_Write(DLE);
    //Se transmite un STX...
    UART1_Write(STX);
    //Se transmite la información del BUFER...
    UART1_Write_Text(BUFER);
    //Se transmite un DLE...
    UART1_Write(DLE);
    //Se transmite el ETX...
    UART1_Write(ETX);
    //Se transmite el fin de transmisión...
    UART1_Write(EOT);
    //Se espera el ACK
    while( !UART1_Data_Ready() );
    //Se verifica si llega un ACK...
    if( UART1_Read() != ACK )goto OTRO;
}
}

```

La estructura para la transparencia de datos obedece a la siguiente forma:

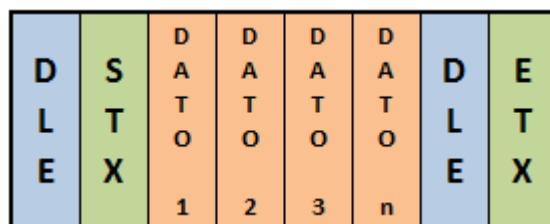
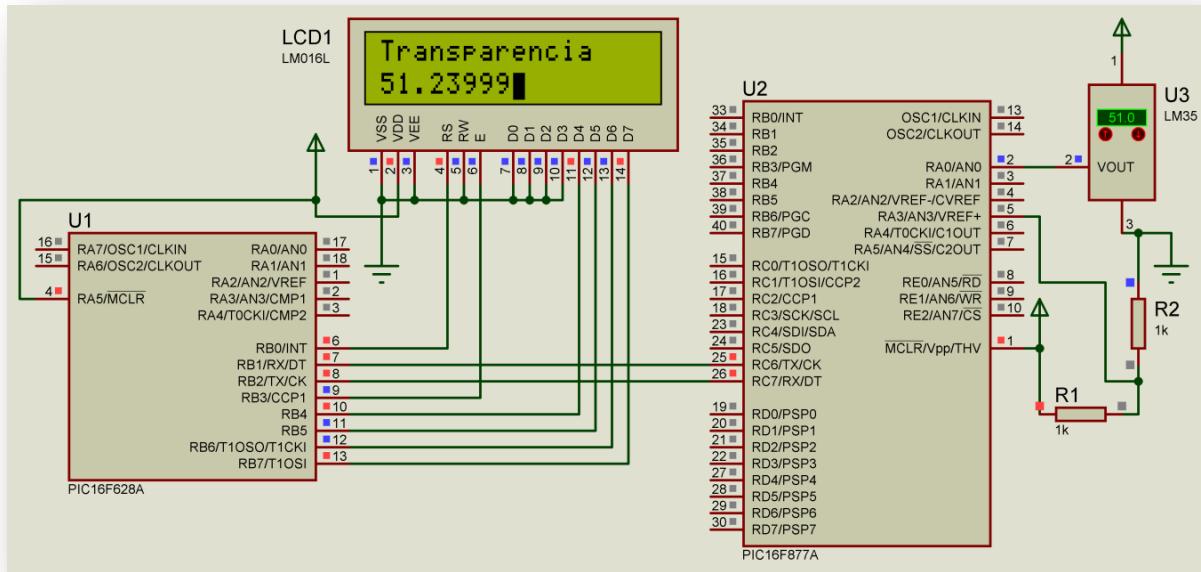


Figura 15-3

Para realizar la respectiva simulación se realiza el siguiente circuito que tiene los dispositivos, PIC 16F628A, PIC 16F877A, RES, LM016L, LM35:



Circuito 15-2

15.3 Control de errores CRC

El control de errores en una transmisión es de suma importancia y más cuando la calidad del medio de transmisión no es confiable. Para detectar los errores de transmisión existen diferentes técnicas entre las cuales se pueden citar, VRC o verificación de redundancia vertical, esta estrategia anexa un bit más a cada paquete de información el cual representa el número par o impar de unos en el paquete. LRC o Verificación de redundancia longitudinal la cual verifica el número de bits pares o impares en un grupo de bytes, para este fin se realiza la operación lógica o exclusiva, entre ellos. Por último se puede hablar del CRC, o verificación de redundancia cíclica, que es la estrategia de mayor efectividad, y que es la técnica en la cual se centra el ejemplo de este capítulo. Para llegar al resultado del CRC, se efectúa una división binaria entre el paquete de datos que se desean transmitir y un polinomio fijo que debe respetar algunas condiciones. Después de hacer la división el residuo de la misma es anexado al paquete de información y este es el CRC. El ente receptor realiza la misma división con el mismo polinomio y despues de culminarla el resultado consignado en el residuo debe ser 0 si los datos han llegado sin ningún error, de lo contrario los datos del paquete están deteriorados.

La estructura de información con el CRC incorporado se implementa en la forma que se puede apreciar en la siguiente figura:

| | | | | | | | | |
|---|---|-----------------------|-----------------------|-----------------------|-----------------------|-------------|-------------|-------------|
| D | S | D A T O 1 | D A T O 2 | D A T O 3 | D A T O n | C R C | D L E | E T X |
|---|---|-----------------------|-----------------------|-----------------------|-----------------------|-------------|-------------|-------------|

Figura 15-4

Los polinomios que se implementan en el cálculo del CRC se denotan en términos de X, en el cual cada X, es un 1 en el polinomio, para ilustrar esta situación se puede apreciar el siguiente ejemplo:

$$X^7 + X^5 + X^2 + X + 1$$

Este polinomio equivale al siguiente número binario:

$$10100111$$

Para que los polinomios funciones de forma adecuada deben cumplir con las siguientes condiciones:

- Debe empezar y terminar con un 1.
- No debe ser divisible por X, es decir por 10.
- Debe ser divisible por X+1, es decir por 11.

Los siguientes son polinomios estándar para diferente número de bits:

CRC-4

$$X^4 + X + 1$$

CRC-8

$$\begin{aligned} &X^8 + X^2 + X + 1 \\ &X^8 + X^5 + X^4 + 1 \end{aligned}$$

CRC-12

$$X^{12} + X^{11} + X^3 + X + 1$$

CRC-16

$$\begin{aligned} &X^{16} + X^{15} + X^2 + 1 \\ &X^{16} + X^{12} + X^5 + 1 \end{aligned}$$

CRC-32

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

El siguiente ejemplo aritmético muestra una división para generar el CRC, y su respectiva verificación:

Información 1 0 1 1 0 1 0 1 1
 Polinomio 1 0 0 1 1

Se cambia el polinomio por 0 →
 dado que el bit mas significativo es cero

$$\begin{array}{r}
 \text{Calculo del CRC} \\
 \begin{array}{r}
 1 0 1 1 0 1 0 1 1 \quad 0 0 0 0 0 \\
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 0 1 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 1 0 1 1 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 0 1 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 1 0 1 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 0 1 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 1 1 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 0 0 1 1 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 0 1 1 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \end{array}
 \end{array}$$

← CRC

$$\begin{array}{r}
 \text{Chequeo del CRC} \\
 \begin{array}{r}
 1 0 1 1 0 1 0 1 1 \quad 1 1 0 0 \\
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 0 1 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 1 0 1 1 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 0 1 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 1 0 1 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 0 1 0 0 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 1 0 0 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 1 0 0 1 1 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \hline
 0 0 0 0 0 \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \end{array}
 \end{array}$$

Chequeo → 0 0 0 0

La implementación de una división de este modo es demasiado compleja. Cuanto más larga sea la trama más difícil es realizarla, por eso es más simple realizar una operación lógica con registros de desplazamiento como estrategia para determinar el residuo de la división. El siguiente diagrama muestra un circuito para el polinomio:

$$X^{16} + X^{15} + X^2 + 1$$

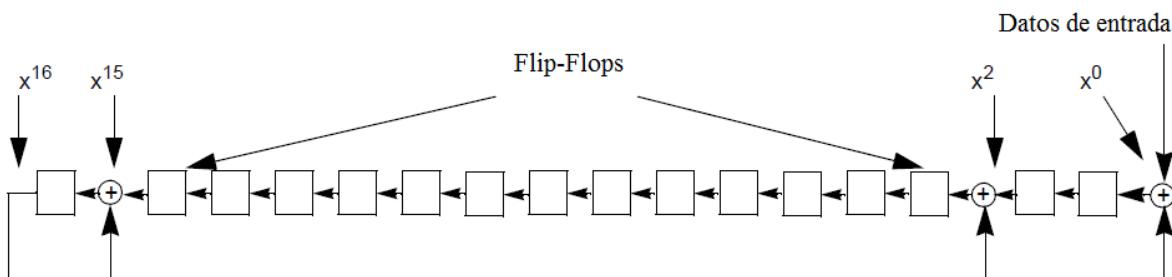


Figura 15-5

Usando como estrategia de programación el corrimiento de bits como lo muestra la figura se puede hacer un conjunto de funciones que calculen el CRC, sobre una trama de longitud indefinida teniendo como base mínima un byte y polinomios con orden múltiplo de 8, tales como 8, 16, 24, 32. Para comprender con claridad este concepto se puede observar el siguiente ejemplo que usa un polinomio de orden 8:

Polinomio:

$$X^8 + X^5 + X^4 + 1$$

100110001

```

unsigned short CRC=0;

//Función para insertar in bit en el cálculo del CRC.
void CorrerBitCRC( char _bit )
{
    unsigned short CRC2=0;
    CRC2 = CRC2 / (CRC<<1)&128; //Set X7
    CRC2 = CRC2 / (CRC<<1)&64; //Set X6
    CRC2 = CRC2 / ((CRC<<1)^((CRC>>2))&32; //Set X5
    CRC2 = CRC2 / ((CRC<<1)^((CRC>>3))&16; //Set X4
    CRC2 = CRC2 / (CRC<<1)&8; //Set X3
    CRC2 = CRC2 / (CRC<<1)&4; //Set X2
    CRC2 = CRC2 / (CRC<<1)&2; //Set X1
    CRC2 = CRC2 / ((CRC>>7)^_bit)&1; //Set X0
    CRC = CRC2;
}

//Función para insertar un Byte al cálculo del CRC.
void CorrerByteCRC( char Byte )
{
    short B;
    for( B=7; B>=0; B-- )
        CorrerBitCRC( (Byte>>B)&1 );
}

//Función para calcular el CRC de una trama de texto.
unsigned short CalcularRCR( char *trama, unsigned char FINAL )
{
    unsigned short N=0;
    CRC = 0;
    while( trama[N]!=0 )
    {
        CorrerByteCRC( trama[N] );
        N++;
    }
    CorrerByteCRC( FINAL );
    return CRC;
}

```

Para realizar la implementación de este ejemplo, se usa el mismo hardware del ejemplo de transparencia, el código fuente para el microcontrolador transmisor implementando el CRC es el siguiente:

```

//Función para insertar un Byte al cálculo del CRC.
void CorrerByteCRC( char Byte )
{
    short B;
    for( B=7; B>=0; B-- )
        CorrerBitCRC( (Byte>>B)&1 );
}

```

```

//Función para calcular el CRC de una trama de texto.
unsigned short CalcularRCR( char *trama, unsigned char FINAL )
{
    unsigned short N=0;
    CRC = 0;
    while( trama[N]!=0 )
    {
        CorrerByteCRC( trama[N] );
        N++;
    }
    CorrerByteCRC( FINAL );
    return CRC;
}

void main( void )
{
    char BUFER[10];
    int adc;
    float Temp;
    //Se configura el modulo ADC, con el pin
    //AN3 como voltaje de referencia.
    ADCON1 = 0b11000001;
    //Se inicia el modulo USART a 9600 bps.
    UART1_Init(9600);
    delay_ms(500);
    while(1)//Bucle infinito.
    {
        OTRO:
        //Lectura del canal 0
        adc = ADC_Read(0);
        //Se calcula el valor de la temperatura.
        Temp = 0.244*adc;
        //Se escribe la temperatura en el BUFER.
        FloatToStr( Temp, BUFER );
        //Calculo del CRC.
        CalcularRCR( BUFER, 0 );
        //Retardo de lectura de 400m segundos
        delay_ms(100);
        //Se transmite el ENQ...
        UART1_Write(ENQ);
        //Se espera el ACK...
        while( !UART1_Data_Ready() );
        //Se verifica si llega un ACK...
        if( UART1_Read() != ACK )goto OTRO;
        //Se transmite un DLE...
        UART1_Write(DLE);
        //Se transmite un STX...
        UART1_Write(STX);
        //Se transmite la información del BUFER...
}

```

```

UART1_Write_Text(BUFER);
//Se transmite el CRC...
UART1_Write(CRC);
//Se transmite un DLE...
UART1_Write(DLE);
//Se transmite el ETX...
UART1_Write(ETX);
//Se transmite el fin de transmisión...
UART1_Write(EOT);
//Se espera el ACK...
while( !UART1_Data_Ready() );
//Se verifica si llega un ACK...
if( UART1_Read() != ACK )goto OTRO;
}
}

```

El código fuente para el microcontrolador receptor con la implementación del CRC es el siguiente:

```

// Declaración de constantes del LCD
sbit LCD_RS at RB0_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_RS_Direction at TRISB0_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D7_Direction at TRISB7_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB4_bit;

//Declaración de variables de trabajo.
char BUFER[20], Dato, Bandera=0;
unsigned short Estado=0, N=0;

//Declaración de constantes.
const char ACK=0x06, NAK=0x15, ENQ=0x05, EOT=0x04, STX=0x02, ETX=0x03,
DLE=0x10,
NULL=0x00;

unsigned short CRC=0, CRC_2, Lon;

//Función para insertar un bit en el cálculo del CRC.
void CorrerBitCRC( char _bit )
{
    unsigned short CRC2=0;

```

```

CRC2 = CRC2 / (CRC<<1)&128; //Set X7
CRC2 = CRC2 / (CRC<<1)&64; //Set X6
CRC2 = CRC2 / ((CRC<<1)^((CRC>>2))&32; //Set X5
CRC2 = CRC2 / ((CRC<<1)^((CRC>>3))&16; //Set X4
CRC2 = CRC2 / (CRC<<1)&8; //Set X3
CRC2 = CRC2 / (CRC<<1)&4; //Set X2
CRC2 = CRC2 / (CRC<<1)&2; //Set X1
CRC2 = CRC2 / ((CRC>>7)^_bit)&1; //Set X0
CRC = CRC2;
}

//Función para insertar un Byte al cálculo del CRC.
void CorrerByteCRC( char Byte )
{
    short B;
    for( B=7; B>=0; B-- )
        CorrerBitCRC( (Byte>>B)&1 );
}

//Función para calcular el CRC de una trama de texto.
unsigned short CalcularRCR( char *trama, unsigned char FINAL )
{
    unsigned short N=0;
    CRC = 0;
    while( trama[N]!=0 )
    {
        CorrerByteCRC( trama[N] );
        N++;
    }
    CorrerByteCRC( FINAL );
    return CRC;
}

//Vector de interrupciones.
void interrupt()
{
    //Se evalúa la interrupción por recepción serial.
    if( PIR1.F5 )
    {
        //Se lee el Dato de entrada.
        Dato = UART1_Read();
        //Se evalúa el estado del tráfico de información.
        switch( Estado )
        {
            //En espera...
            case 0: //Se evalúa si llega una solicitud..
                if( Dato == ENQ )
                {
                    //Se cambia al siguiente Estado y se
}

```

```

    //envía el reconocimiento positivo.
    Estado = 1;
    UART1_Write(ACK);
}
else
{
    //Se envía reconocimiento negativo.
    UART1_Write(NAK);
}
break;

//Delimitador de transparencia...
case 1: // Se evalúa si lo que llega es DLE.
if( Dato == DLE )
{
    //Se cambia al siguiente Estado...
    Estado = 2;
}
else
{
    //Se identifica un error de orden en los datos
    //y se transmite un reconocimiento negativo y se
    //reanuda al Estado inicial.
    //Se reinicia de nuevo al Estado inicial.
    Estado = 0;
    UART1_Write(NAK);
}
break;

// Recibiendo información...
case 2: // Se evalúa si lo que llega es STX.
if( Dato == STX )
{
    //Se inicia la cadena de información y se
    //pasa al Estado 3
    N=0;
    BUFER[N]=NULL;
    Estado = 3;
}
else
{
    //Se identifica un error de orden en los datos
    //y se transmite un reconocimiento negativo y se
    //reanuda al Estado inicial.
    //Se reinicia de nuevo al Estado inicial.
    Estado = 0;
    UART1_Write(NAK);
}
break;

// Esperando datos y delimitador de transparencia.
case 3: //Se evalúa si el Dato que llega es un DLE.

```

```

if( Dato == DLE )
{
    //Se cambia al Estado siguiente.
    Estado = 4;
    //Se finaliza el BUFER de datos;
    BUFER[N]=NULL;
}
else
{
    //Se anexa un nuevo Dato al BUFER.
    BUFER[N]=Dato;
    //Se incrementa el índice de los datos.
    N++;
}
break;
//Esperando fin de cadena...
case 4: // Se evalua si llega un ETX
if( Dato == ETX )
{
    //Se cambia al Estado siguiente.
    Estado = 5;
}
else
{
    //Se cambia de nuevo al Estado inicial.
    Estado = 0;
    //Se transmite reconocimiento negativo.
    UART1_Write(NAK);
}
break;
//Esperando fin de transmisión...
case 5: //Se evalúa si llega el EOT...
if( Dato == EOT )
{
    //Se mide la longitud de la trama.
    Lon = strlen(BUFER);
    //Se filtra el CRC, que está en la última posición.
    CRC_2 = BUFER[Lon-1];
    BUFER[Lon-1]=0;
    //Se calcula el CRC...
    CalcularRCR( BUFER, CRC_2 );
    // Se verifica que el chequeo sea cero...
    if( CRC==0 )
    {
        //Se activa la Bandera de llegada de información.
        Bandera = 1;
        //Se transmite reconocimiento positivo.
        UART1_Write(ACK);
    }
}

```

```

    else
    {
        //Se transmite reconocimiento negativo.
        UART1_Write(NAK);
    }
}
else
{
    //Se transmite reconocimiento negativo.
    UART1_Write(NAK);
}
Estado = 0;
break;
default: break;
}
}

void main( void )
{
    //Se activan las interrupciones periféricas.
    INTCON = 0b11000000;
    //Se activa la interrupción por recepción serial.
    PIE1 = 0b00100000;
    //Configuración del modulo USART a 9600 bps.
    UART1_Init(9600);
    //Configuración del LCD.
    Lcd_Init();
    //Texto de inicio.
    Lcd_Out(1,1,"Transparencia");
    while(1) //Bucle infinito.
    {
        //Se evalúa si la Bandera esta activa.
        if( Bandera == 1 )
        {
            //Se imprime la información en el LCD.
            Lcd_Out(2,1,"      ");
            Lcd_Out(2,1,BUFER);
            Bandera = 0;
        }
    }
}

```

16 Actuadores y potencia

Los actuadores son dispositivos eléctricos o electrónicos que permiten crear un cambio físico sobre una variable. Están pueden ser temperatura, velocidad, presión, luminosidad, humedad, posición angular o lineal, entre otras. Los actuadores pueden ser de accionamiento AC, o DC. Los actuadores DC, son aquellos como motores, servomotores, lámparas incandescentes, solenoides, relevadores, entre otros. Los actuadores AC, son aquellos como motores AC, lámparas incandescentes, resistencias eléctricas, entre otros. Por otra parte la potencia es la cantidad energía eléctrica requerida para realizar un trabajo determinado. En el caso puntual de la electricidad la potencia está definida como:

$$P_e = VI = \frac{V^2}{R} = I^2 R \text{ Ecuación 16-1}$$

Donde P_e es la potencia eléctrica en vatios, V es la diferencia de potencial eléctrico sobre una carga eléctrica con resistencia R , por la cual circula una corriente I . En el caso de un microcontrolador el uso de un arreglo de potencia se hace indispensable cuando un actuador requiere más voltaje o corriente, que la que puede entregar un pin del PIC. La tensión que entregan las salidas del PIC siempre es de 5 voltios y la corriente que puede suministrar solo puede ser del orden de pocos miliamperios.

16.1 Actuadores DC

Los actuadores DC, requieren de corriente siempre directa es decir que siempre circula en la misma dirección. Para el uso de estos actuadores se implementan esencialmente dos tipos de dispositivos, los transistores de potencia o los relevadores. Los transistores pueden ser BJT, o MOSFET. Un transistor BJT, se caracteriza por controlar una corriente por medio de otra corriente. Los MOSFET, permiten controlar un flujo de corriente por medio de una diferencia de potencial. Los transistores funcionan en tres zonas, conocidas como corte, saturación y activa. La zona activa es común en los sistemas de amplificación de señal. Sin embargo para el caso de los actuadores se requieren en la mayoría de los casos ser activados y desactivados, o tener un comportamiento ON, OFF. En otras palabras los transistores se usan en corte y saturación. Bajo estos términos los transistores trabajan como interruptores de corriente, en las zonas de corte y saturación. La vista de los transistores en ISIS es la siguiente:

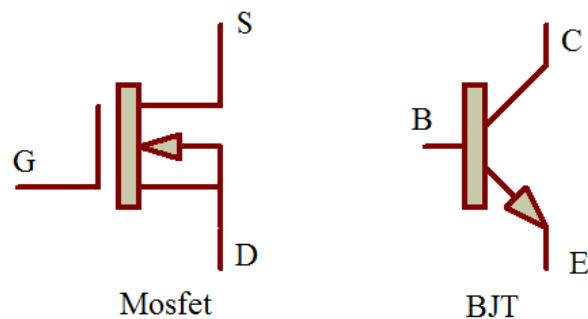
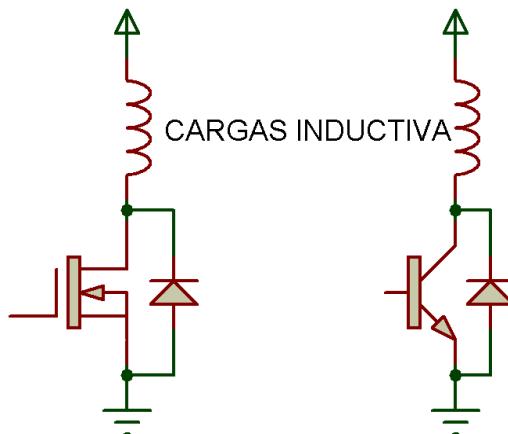


Figura 16-1

Los transistores MOSFET, controlan la corriente que circula entre la fuente S, y el drenador D, por medio de la diferencia de potencial eléctrico sobre la compuerta G. Los transistores BJT, controlan la corriente que fluye entre el colector C, y el emisor E, por medio de la corriente que circula por la base B.

El uso de cargas inductivas como motores, relevadores o solenoides, tienen la propiedad de almacenar energía en forma de campo electromagnético, esto implica que cuando las cargas inductivas son des energizadas, estás devuelven su energía por medio de una corriente inversa, este fenómeno puede causar daños en los sistemas controlados con transistores, para evitar este problema se debe usar un diodo rectificador polarizado en inverso para descargar las cargas inductivas, este arreglo se muestra en el siguiente circuito:



Circuito 16-1

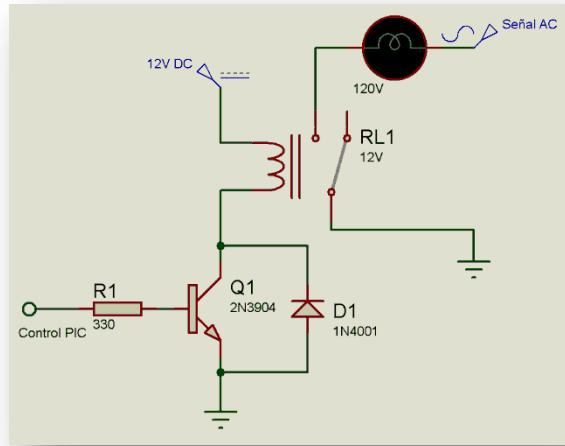
16.1.1 Relevadores

Los relevadores son dispositivos electromecánicos, que tienen un electroimán con el cual se cierra o se abre un circuito eléctrico por medio de uno o varios contactos. Los relevadores son ideales para aislar los circuitos de potencia con la etapa de control electrónico. De igual manera son dispositivos de bajo rendimiento, dado que no pueden hacer cambios de estado veloces, y con el uso constante los contactos eléctricos se deterioran con el tiempo. La apariencia física y la vista en ISIS, de estos dispositivos es la siguiente:



Figura 16-2

Por último el uso de estos dispositivos se debe hacer de forma final como se puede observar en el siguiente circuito, que tiene una carga de 120V AC:



Circuito 16-2

16.1.2 Motores DC

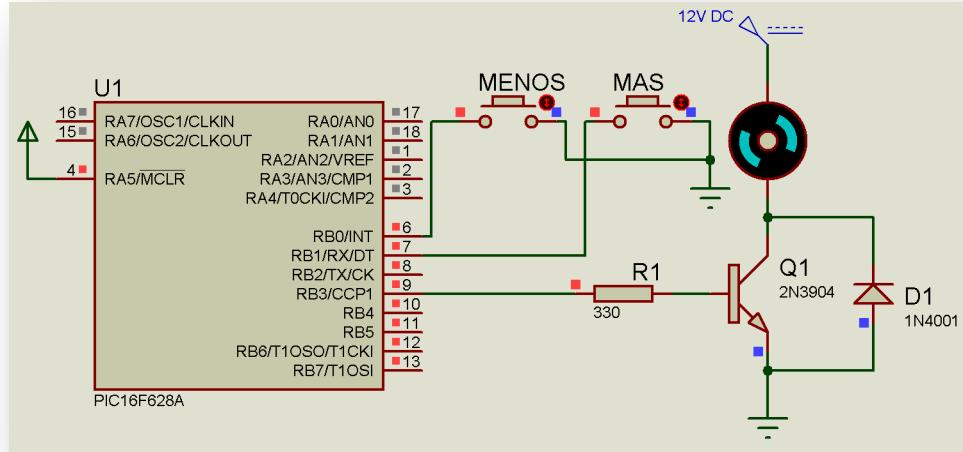
Los motores DC, son de naturaleza inductiva y por consiguiente merecen el mismo tratamiento de los relevadores. En el siguiente ejemplo se puede apreciar como un microcontrolador hace cambios de velocidad en un motor DC, para este fin se implementa el módulo PWM de un microcontrolador 16F628A, con oscilador de 4MHz. El código fuente en lenguaje C es el siguiente:

```

void main( void )
{
    unsigned short CU=0;
    OPTION_REG = 0; //Se activan las resistencias PULL-UP.
    PWM1_Init( 500 ); //Se inicia el módulo PWM a 500Hz.
    PWM1_Set_Duty(CU);
    PWM1_Start();
    while(1) //Bucle infinito.
    {
        //Bucle de incremento del PWM cuando se pulsa RB1.
        while( Button( &PORTB, 1, 10, 0 ) )
        {
            CU++; if( CU==0 )CU=255;
            PWM1_Set_Duty(CU);
            delay_ms(10);
        }
        //Bucle de decremento del PWM cuando se pulsa RB0.
        while( Button( &PORTB, 0, 10, 0 ) )
        {
            CU--; if( CU==255 )CU=0;
            PWM1_Set_Duty(CU);
            delay_ms(10);
        }
    }
}

```

Para poder hacer la simulación de este ejemplo se debe implementar en ISIS los dispositivos: 16F628A, BUTTON, RES, 2N3904, 1N4001, y MOTOR ACTIVE. Configurados como se puede apreciar en el siguiente circuito:

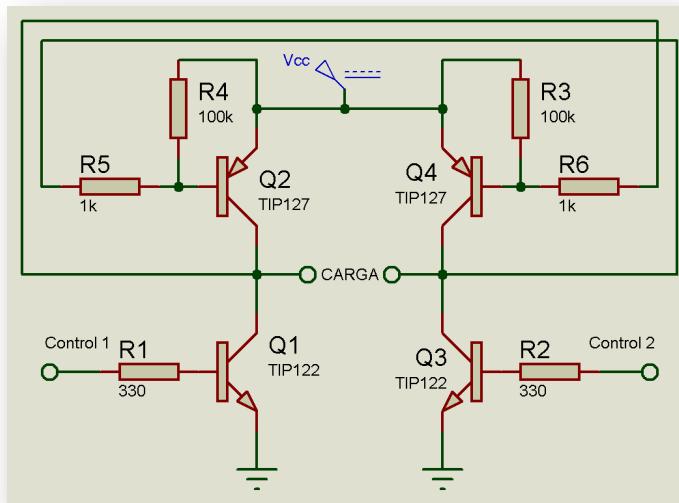


Circuito 16-3

En función de la intensidad de corriente se puede cambiar el transistor para optimizar el funcionamiento de los sistemas. Un transistor como el 2N3904 puede controlar una corriente de 200m Amperios, y un TIP31 puede controlar una corriente de 3 Amperios, en función de la referencia y sus características, se puede asumir un máximo de corriente.

16.1.3 Puente H

Los puentes H son arreglos transistorizados que permiten invertir la polaridad sobre una carga eléctrica, con el suministro de solo una fuente sencilla. Los puentes H, se pueden implementar en discreto con transistores o usar módulos integrados que se pueden conseguir comercialmente. Estos arreglos cuentan con dos señales de control que permiten activar la polaridad positiva, y negativa. Un diseño discreto de este arreglo se puede apreciar en el siguiente circuito:



Circuito 16-4

Para activar una polaridad positiva, se activa el control 1, y se apaga el control 2, para activar la polaridad negativa se hace la configuración contraria con los pines de control. Se debe tener presente que nunca se podrán activar los dos puntos de control al mismo tiempo, esta acción genera un corto circuito entre Vcc, y referencia, produciendo daños considerables sobre los transistores.

El siguiente ejemplo muestra el control de giro sobre un motor DC, por medio de un microcontrolador 16F628A:

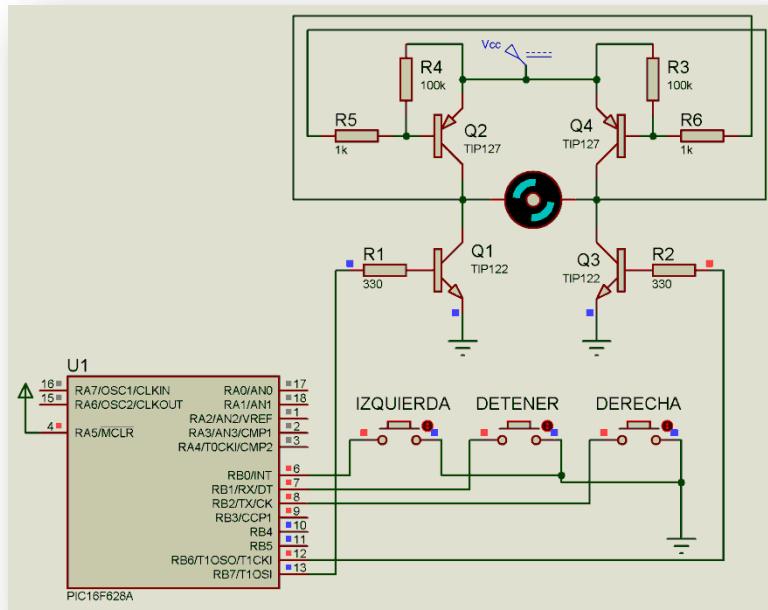
```
void main( void )
{
    OPTION_REG = 0; //Se activan las resistencias PULL-UP.
    //Configuración de salida y entrada del puerto B.
    PORTB = 0;
    TRISB = 0x0F;
    PORTB = 0;

    while(1) //Bucle infinito.
    {
        //Bucle para detectar el botón de giro a la izquierda.
        while( Button( &PORTB, 0, 10, 0 ) )
        {
            PORTB = 0; //Se apagan los pines de control.
            PORTB.F7 = 1; //Se activa el pin de control 1.
        }

        //Bucle para detectar el botón de detener.
        while( Button( &PORTB, 1, 10, 0 ) )
        {
            PORTB = 0; //Se apagan los pines de control.
        }

        //Bucle para detectar el botón de giro a la derecha.
        while( Button( &PORTB, 2, 10, 0 ) )
        {
            PORTB = 0; //Se apagan los pines de control.
            PORTB.F6 = 1; //Se activa el pin de control 2.
        }
    }
}
```

La simulación en ISIS, se realiza por medio del siguiente circuito electrónico:



Circuito 16-5

16.1.4 Motores Paso

Los motores paso, son dispositivos electromecánicos que permiten hacer giros fraccionados por grados, los motores paso son de dos naturalezas; unipolares y bipolares. Un motor unipolar cuenta con cuatro embobinados, que se energizan uno o dos a la vez, y siempre con la misma polaridad. Los motores paso bipolares cuentan con solo dos bobinas, las cuales se polarizan al mismo tiempo, pero alternando la polaridad. Este efecto genera una secuencia en las bobinas y su polaridad. Manipulando la velocidad de la secuencia se controla la velocidad de giro del motor, y con el orden de la secuencia se controla la dirección de giro del motor.

En las siguientes gráficas se puede apreciar la apariencia física de estos motores y su vista en ISIS:

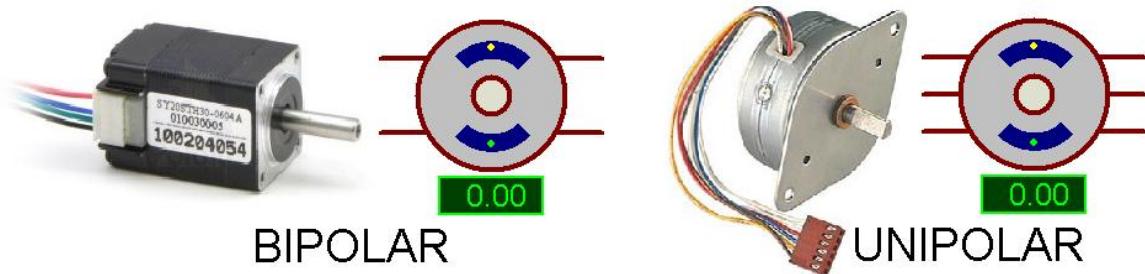
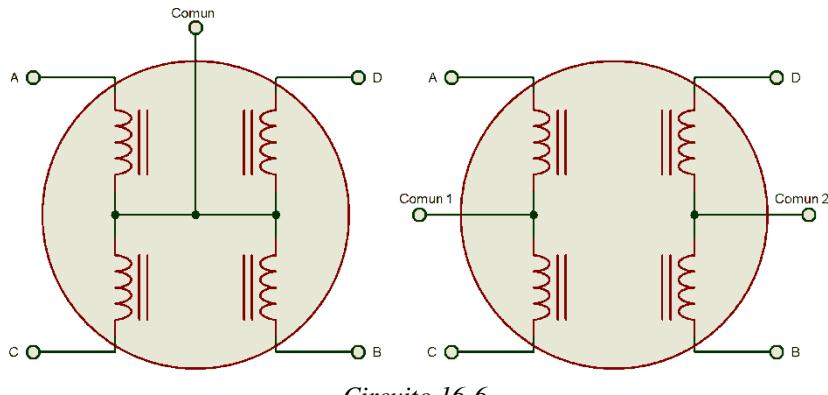


Figura 16-3

Los motores unipolares, cuentan generalmente con 5 o 6 terminales, de las cuales 4 corresponden a los embobinados, y 1 o 2 son terminales comunes. Las bobinas de un motor unipolar se pueden apreciar en el siguiente circuito:



Circuito 16-6

La secuencia de activación para los motores unipolares puede ser de dos formas; con una sola entrada activa, o dos entradas activas simultáneamente. Cuando se activan dos entradas simultáneamente el torque del motor es mayor pero al mismo tiempo la corriente es mayor. La siguiente tabla muestra la forma de activar las secuencias en los dos casos:

| PASO | Secuencias para motores paso unipolares | | | | | | | |
|------|---|-----|-----|-----|--------------------------------|-----|-----|-----|
| | Secuencia con una activación | | | | Secuencia con dos activaciones | | | |
| | LA | LB | LC | LD | LA | LB | LC | LD |
| 1 | ON | OFF | OFF | OFF | ON | ON | OFF | OFF |
| 2 | OFF | ON | OFF | OFF | OFF | ON | ON | OFF |
| 3 | OFF | OFF | ON | OFF | OFF | OFF | ON | ON |
| 4 | OFF | OFF | OFF | ON | ON | OFF | OFF | ON |

Tabla 16-1

El siguiente ejemplo muestra como realizar el control de un motor unipolar con un PIC 16F628A, que cuenta con una frecuencia de reloj de 4MHz, para este fin la secuencia del motor se guarda en un arreglo. El código en lenguaje C es el siguiente:

```
//Constantes con la secuencia de pasos.
const unsigned short PASOS[4] =
{
    0b00000001,
    0b00000010,
    0b00000100,
    0b00001000
};

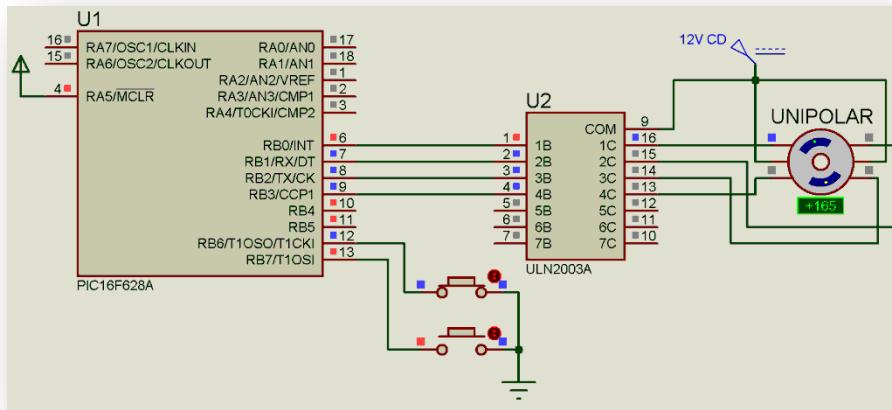
void main( void )
{
    //Declaración de variables.
    unsigned short PASO=0;
    //Inicio del puerto
    TRISB = 0xF0;
    PORTB = 0;
    OPTION_REG = 0; //Activación de las resistencias PULL-UP.
```

```

while(1)//Bucle infinito.
{
    //Bucle while para hacer girar en un sentido
    //por medio del pin RB6
    while( Button( &PORTB, 6, 100, 0 ) )
    {
        PORTB = PASOS[PASO];
        PASO++;
        if( PASO==4 )PASO=0;
    }
    //Bucle while para hacer girar en un sentido contrario
    //por medio del pin RB7
    while( Button( &PORTB, 7, 100, 0 ) )
    {
        PORTB = PASOS[PASO];
        PASO--;
        if( PASO==255 )PASO=3;
    }
}
}

```

Para simular este ejercicio se implementa en ISIS, el siguiente circuito electrónico, con el MOTOR-STEPPER, BUTTON, y el driver ULN2003A:



Circuito 16-7

Para realizar el ejercicio con la secuencia de doble activación simplemente se cambia la secuencia en el arreglo del mismo programa y se usa la misma simulación. El nuevo arreglo es el siguiente:

```

//Constantes con la secuencia de pasos.
const unsigned short PASOS[4] =
{
    0b00000011,
    0b00000110,

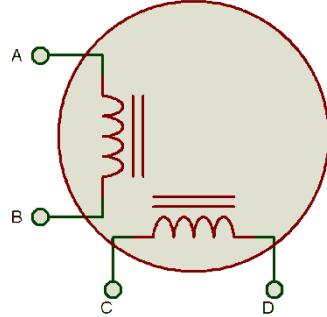
```

```

0b00001100,
0b00001001
};

```

Para los motores bipolares se implementa un puente H doble, de tal forma que se pueda hacer una secuencia con doble polaridad. La distribución eléctrica de este tipo de motores es la siguiente:



Circuito 16-8

Para el control de este tipo de motores se requiere de la siguiente secuencia de polaridad:

| PASO | Secuencia para Bipolar | | | |
|------|------------------------|----|----|----|
| | LA | LB | LC | LD |
| 1 | +V | -V | +V | -V |
| 2 | +V | -V | -V | +V |
| 3 | -V | +V | -V | +V |
| 4 | -V | +V | +V | -V |

Tabla 16-2

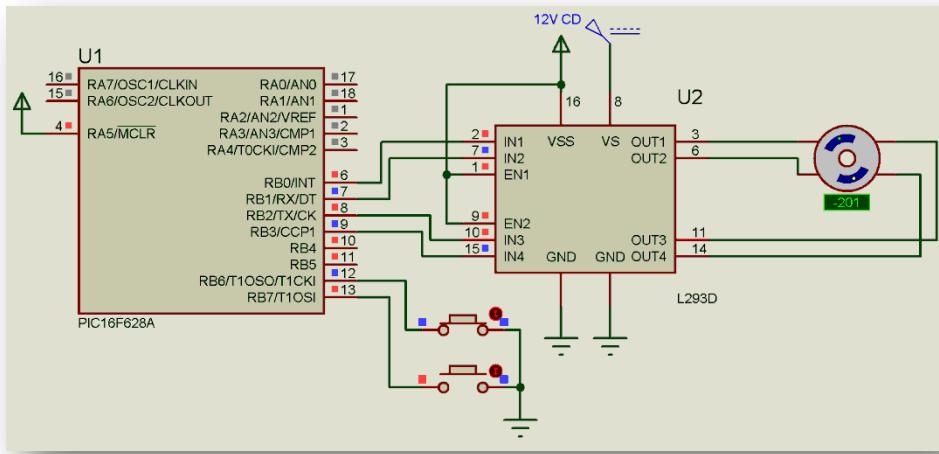
Para realizar el ejercicio con este motor se implementa el mismo programa del motor unipolar y se altera el arreglo de la secuencia, en este nuevo arreglo se asume un 1 lógico para los +V, de la tabla y un 0 lógico para los -V. De esta forma se obtiene el siguiente arreglo:

```

//Constantes con la secuencia de pasos.
const unsigned short PASOS[4] =
{
    0b00000101,
    0b00001001,
    0b00001010,
    0b00000110
};

```

Para la simulación del sistema se implementa en ISIS un circuito similar, pero con un motor bipolar y un driver para puente H integrado. Este integrado es el L293D, que es un circuito integrado que tiene en su interior dos puentes H simultáneamente, característica que lo hace ideal para este tipo de motores. La simulación en ISIS, implementa el 16F628A, BUTTON, MOTOR-BISTEPPER, y el L293D. El circuito para simular es el siguiente:



Circuito 16-9

16.1.5 Servomotores

Los servomotores son sistemas integrados que tienen un control de posición angular, y un sistema mecánico de piñones para ofrecer mayor fuerza pero menor velocidad. Un servomotor tiene incorporado un control de posición angular, que puede ser gobernado por medio de una señal PWM. Las aplicaciones de los servomotores están desde la robótica hasta los modelos a escala como aviones, helicópteros y carros. Los servomotores cuentan con un terminal de tres pines para la alimentación, la referencia y la señal de control PWM. La apariencia física y la vista en ISIS de estos dispositivos es la siguiente:



Figura 16-4

La característica de la señal de control es su periodo de PWM de 16m a 18m segundos y el periodo útil puede variar de 1 a 2 milisegundos, esto hace referencia respectivamente a 0 y 180 grados en la posición angular. La siguiente gráfica ilustra esta situación:

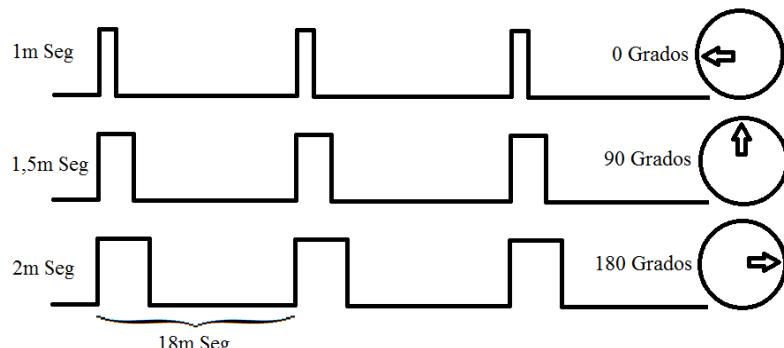


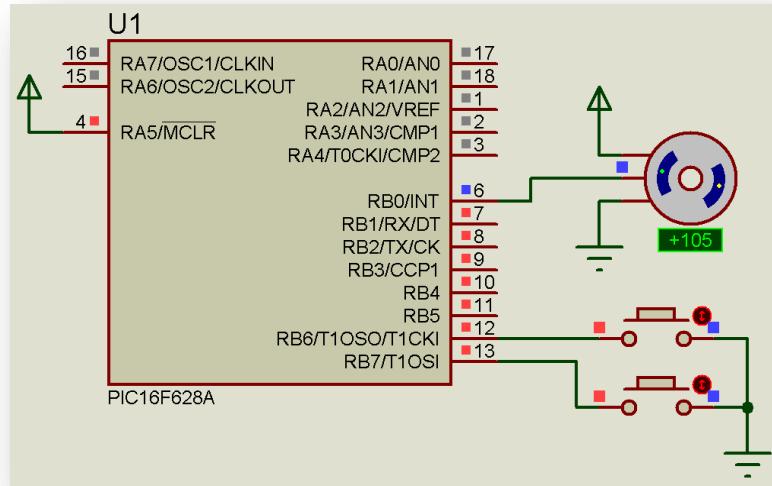
Figura 16-5

A continuación se muestra un ejemplo en lenguaje C, para controlar la posición de un servomotor:

```
//Función para generar el PWM de
//baja frecuencia, ajustado para
//cristal de 20MHz.
void Pwm_Servo(float ang)
{
    unsigned int n, max;
    max = 1.61*ang;
    PORTB.F0 = 1;
    delay_ms(1);
    for( n=0; n<max; n++ )
        delay_us(1);
    PORTB.F0=0;
    delay_ms(15);
}

void main( void )
{
    //Angulo inicial.
    float ANGULO=90;
    //Inicio del puerto
    TRISB.F0 = 0;
    //Se activan las resistencias PULL-UP.
    OPTION_REG=0;
    while(1)//Bucle infinito.
    {
        Pwm_Servo(ANGULO); //Se genera un pulso.
        //Si se pulsa el botón de RB6
        //se decrementan 10 grados.
        if( PORTB.F6==0 )
        {
            ANGULO-=10.0;
            if( ANGULO<0.0 )ANGULO=0.0;
            while( Button(&PORTB,6,5,0) );
        }
        //Si se pulsa el botón de RB7
        //se incrementan 10 grados.
        if( PORTB.F7==0 )
        {
            ANGULO+=10.0;
            if( ANGULO>180.0 )ANGULO=180.0;
            while( Button(&PORTB,7,5,0) );
        }
    }
}
```

Para simular este ejercicio, se implementa en ISIS, el PIC 16F628A, con frecuencia de reloj de 20MHz, MOTOR-PWMSERVO, y el BUTTON. El circuito correspondiente es el siguiente:



Circuito 16-10

16.2 Actuadores AC

Para el uso de actuadores AC, se pueden usar esencialmente dos técnicas, la primera es la implementación de relevadores con la limitación de que solo pueden hacerse acciones de conmutación ON, OFF. Sin embargo se pueden implementar dispositivos en estado sólido como los TRIAC, estos dispositivos permiten manipular cargas AC, y pueden regular su potencia haciendo recorte de fase de la señal AC, que generalmente es de 60Hz. Los TRIAC son de fácil implementación y su uso puede hacerse aislando las corrientes de potencia con optóacopladores como los MOC3011, 3010, 3021, etc. Los TRIAC se pueden adquirir en capacidad de la máxima corriente que pueden soportar. Un TRIAC cuenta con tres terminales que son: A1, A2, y gate, los TRIAC cortocircuitan las terminales A1, y A2, cuando una corriente circula entre el gate y A1. y solo se desconecta cuando la corriente de A2 se hace 0, esto es posible cuando la señal AC, hace su cruce por cero.

La apariencia física y la vista en ISIS de estos dispositivos es la siguiente:

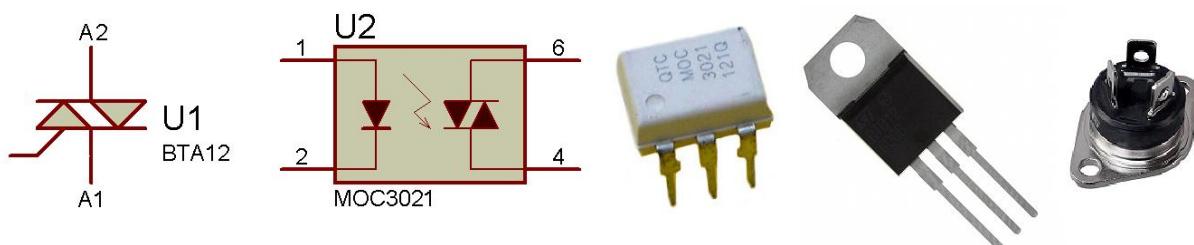
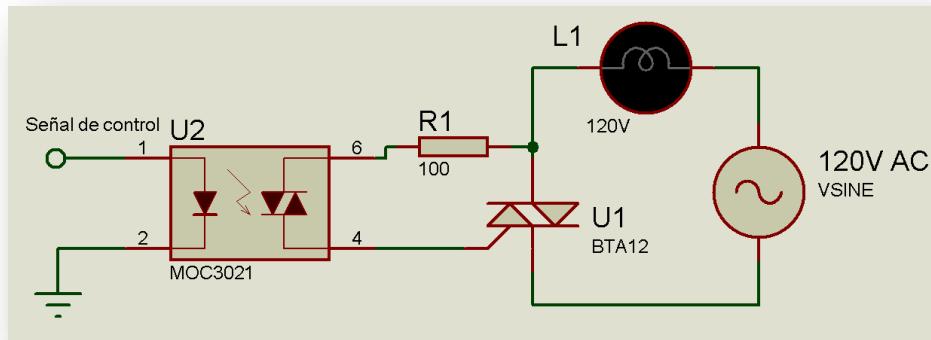


Figura 16-6

Como se puede observar en la gráfica los MOC, son dispositivos que internamente tienen un TRIAC, pero la corriente que pueden soportar es relativamente pequeña, y a su vez lo suficientemente grande para disparar otro TRIAC de mayor potencia. También se puede observar

que la compuerta gate de los MOC, se activa por medio de la luz emitida por un LED, este también está incorporado en el mismo encapsulado. En conclusión un TRIAC de gran potencia puede ser activado por medio de un circuito digital con solo la activación de un LED.

Para entender este arreglo de forma clara, se puede observar el siguiente circuito:



Circuito 16-11

Para lograr la regulación de la potencia sobre una carga AC, se hace necesario hacer un recorte sobre la fase, de la señal de poder seno. La red eléctrica comercial tiene 120V AC, y una frecuencia de 60Hz. Lo que indica que un ciclo de la red eléctrica dura 1/60, es decir 16,666m segundos. El trabajo de recorte de la fase se debe hacer sobre cada medio ciclo es decir en 8,333m segundos. Lo anterior indica que el recorte de fase se hace entre 0 y 180 grados, y se repite igual entre 180 y 360 grados. Para comprender de forma clara este concepto se puede observar la siguiente gráfica:

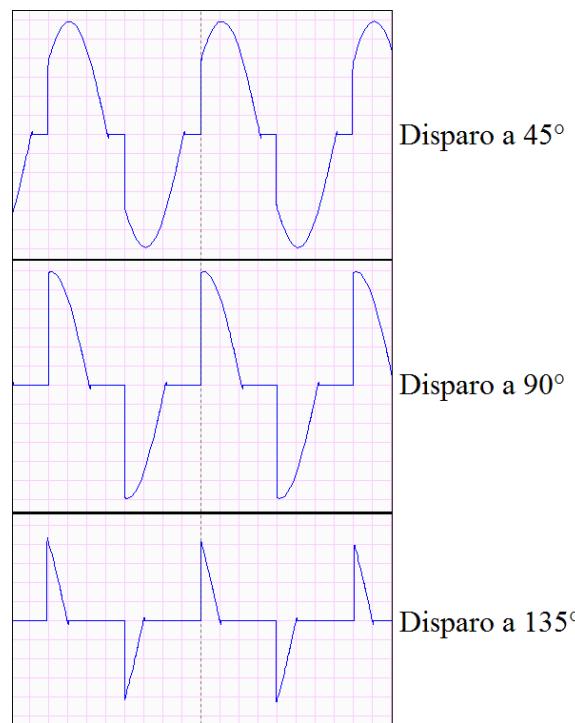


Figura 16-7

Para poder realizar el recorte de fase, es indispensable detectar el cruce por cero de la señal seno, para sincronizarse con ella. Para detectar el cruce por cero existen diversas estrategias, sin embargo en el próximo ejemplo se mostrará una que implica la implementación de opto acopladores. El uso de opto acopladores es ideal para hacer un aislamiento eléctrico de la etapa de potencia con la etapa de electrónica de control. El siguiente programa usa un PIC 16F628A, con frecuencia de reloj de 20MHz:

```

//Variables globales.
unsigned short PWM=41, CONT=0;

//Función de interrupciones.
void interrupt()
{
    //Interrupción externa por RB0.
    if( INTCON.F1 )
    {
        //Se apaga el pin de disparo.
        PORTB.F7=0;
        //Se reinicia el contador.
        CONT = 0;
        //Se apaga la bandera de la interrupción
        //externa.
        INTCON.F1=0;
    }
    //Interrupción por Timer 0 a 102.4u Seg.
    if( INTCON.F2 )
    {
        //Se incrementa el contador.
        CONT++;
        //Se compara el contador con el punto,
        //PWM, y se apaga el disparo y es mayor el contador.
        if( CONT > PWM )PORTB.F7=1;
        //Se apaga la bandera de la interrupción por
        //Timer 0.
        INTCON.F2 =0;
    }
}

void main( void )
{
    //Configuración de puertos.
    TRISB = 0b0111111;
    PORTB = 0;
    //Activación del Timer0, y flanco de
    //interrupción externa.
    OPTION_REG = 0b01000000;
    INTCON = 0b10110000;
    while( 1 )// Bucle infinito.
    {
}

```

```

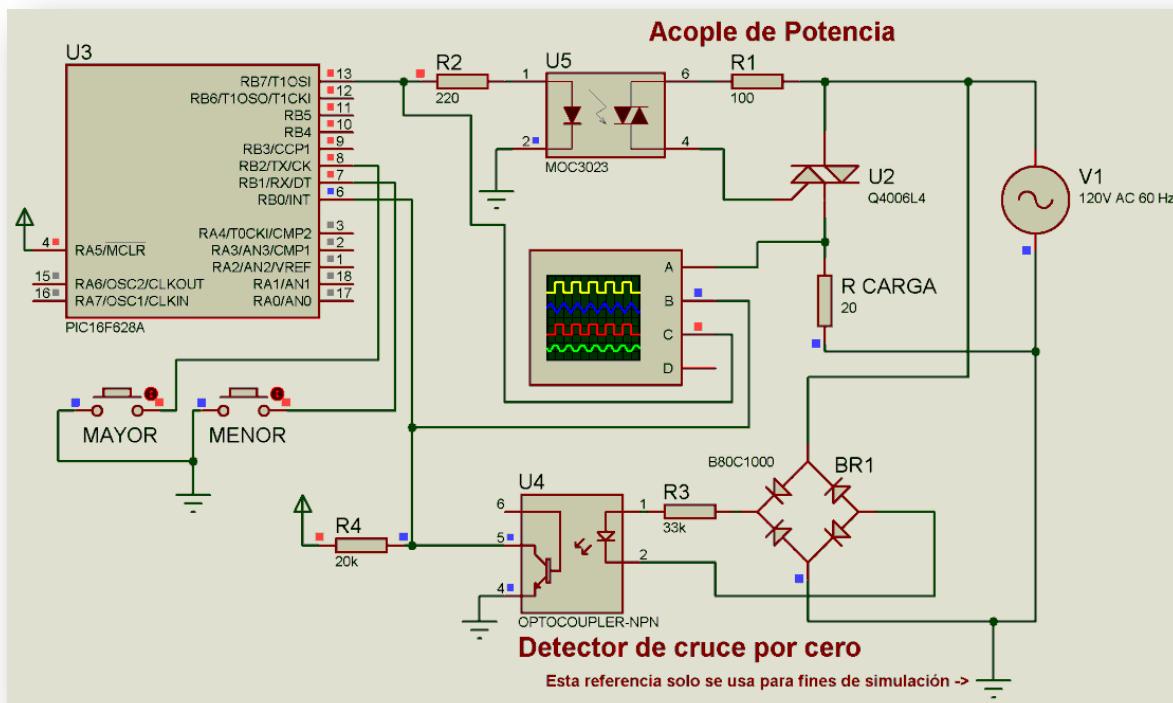
//Pulsador de menor potencia.
while( Button( &PORTB, 1, 10, 0 ) )
{
    //Decremento del recorte de fase.
    PWM ++; if( PWM > 84 )PWM=84;
    delay_ms(10); //Retardo de cambio.
}

//Pulsador de mayor frecuencia.
while( Button( &PORTB, 2, 10, 0 ) )
{
    //Incremento del recorte de fase.
    PWM --; if( PWM == 255 )PWM=0;
    delay_ms(10); //Retardo de cambio.
}

}

```

Para realizar la simulación de este ejercicio se implementa en ISIS los dispositivos: 16F628A, RES, BUTTON, MOC3023, Q4006L4, B80C1000, OPTOCOUPLER-NPN, VSINE, y el OSCILLOSCOPE. El circuito electrónico es el siguiente:



Circuito 16-12

Después de correr la simulación se puede apreciar en el osciloscopio virtual, la siguiente vista:

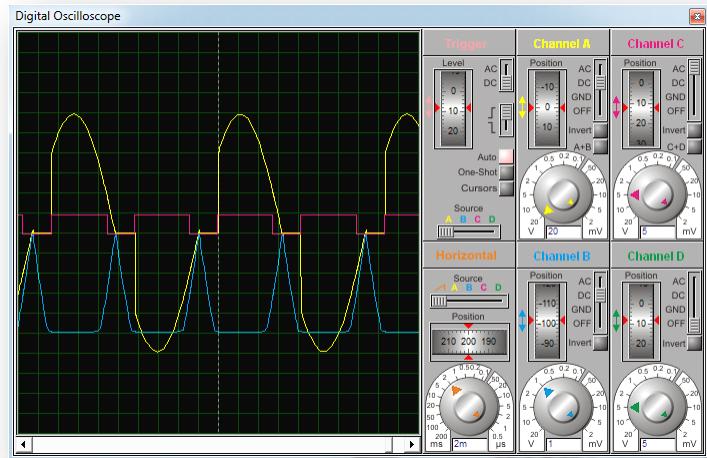


Figura 16-8

El canal A representa la señal sobre la carga de potencia y se puede apreciar sobre está, el recorte de fase. El canal B, muestra un pico cuando la señal seno presenta su cruce por cero, este pico es usado por el microcontrolador para sincronizarse con la red por medio de la interrupción externa. El canal C, muestra la señal de control que dispara el juego de TRIACs.

17 Anexos

17.1 Tabla ASCII

| ASCII | Hex | Símbolo | ASCII | Hex | Símbolo | ASCII | Hex | Símbolo | ASCII | Hex | Símbolo |
|-------|-----|---------|-------|-----|---------|-------|-----|-----------|-------|-----|---------|
| 0 | 0 | NUL | 16 | 10 | DLE | 32 | 20 | (Espacio) | 48 | 30 | 0 |
| 1 | 1 | SOH | 17 | 11 | DC1 | 33 | 21 | ! | 49 | 31 | 1 |
| 2 | 2 | STX | 18 | 12 | DC2 | 34 | 22 | " | 50 | 32 | 2 |
| 3 | 3 | ETX | 19 | 13 | DC3 | 35 | 23 | # | 51 | 33 | 3 |
| 4 | 4 | EOT | 20 | 14 | DC4 | 36 | 24 | \$ | 52 | 34 | 4 |
| 5 | 5 | ENQ | 21 | 15 | NAK | 37 | 25 | % | 53 | 35 | 5 |
| 6 | 6 | ACK | 22 | 16 | SYN | 38 | 26 | & | 54 | 36 | 6 |
| 7 | 7 | BEL | 23 | 17 | ETB | 39 | 27 | ' | 55 | 37 | 7 |
| 8 | 8 | BS | 24 | 18 | CAN | 40 | 28 | (| 56 | 38 | 8 |
| 9 | 9 | TAB | 25 | 19 | EM | 41 | 29 |) | 57 | 39 | 9 |
| 10 | A | LF | 26 | 1A | SUB | 42 | 2A | * | 58 | 3A | : |
| 11 | B | VT | 27 | 1B | ESC | 43 | 2B | + | 59 | 3B | ; |
| 12 | C | FF | 28 | 1C | FS | 44 | 2C | , | 60 | 3C | < |
| 13 | D | (Enter) | 29 | 1D | GS | 45 | 2D | - | 61 | 3D | = |
| 14 | E | SO | 30 | 1E | RS | 46 | 2E | . | 62 | 3E | > |
| 15 | F | SI | 31 | 1F | US | 47 | 2F | / | 63 | 3F | ? |

| ASCII | Hex | Símbolo |
|-------|-----|---------|-------|-----|---------|-------|-----|---------|-------|-----|---------|
| 64 | 40 | @ | 80 | 50 | P | 96 | 60 | ` | 112 | 70 | p |
| 65 | 41 | A | 81 | 51 | Q | 97 | 61 | a | 113 | 71 | q |
| 66 | 42 | B | 82 | 52 | R | 98 | 62 | b | 114 | 72 | r |
| 67 | 43 | C | 83 | 53 | S | 99 | 63 | c | 115 | 73 | s |
| 68 | 44 | D | 84 | 54 | T | 100 | 64 | d | 116 | 74 | t |
| 69 | 45 | E | 85 | 55 | U | 101 | 65 | e | 117 | 75 | u |
| 70 | 46 | F | 86 | 56 | V | 102 | 66 | f | 118 | 76 | v |
| 71 | 47 | G | 87 | 57 | W | 103 | 67 | g | 119 | 77 | w |
| 72 | 48 | H | 88 | 58 | X | 104 | 68 | h | 120 | 78 | x |
| 73 | 49 | I | 89 | 59 | Y | 105 | 69 | i | 121 | 79 | y |
| 74 | 4A | J | 90 | 5A | Z | 106 | 6A | j | 122 | 7A | z |
| 75 | 4B | K | 91 | 5B | [| 107 | 6B | k | 123 | 7B | { |
| 76 | 4C | L | 92 | 5C | \ | 108 | 6C | l | 124 | 7C | / |
| 77 | 4D | M | 93 | 5D |] | 109 | 6D | m | 125 | 7D | } |
| 78 | 4E | N | 94 | 5E | ^ | 110 | 6E | n | 126 | 7E | ~ |
| 79 | 4F | O | 95 | 5F | - | 111 | 6F | o | 127 | 7F | ▷ |

Bibliografía

Behrouz A. Forouzan, Transmisión de datos y redes de comunicaciones, Segunda Edición, Mc Graw Hill, ISBN 84-481-3390-0.

Boylestád Robert, Electronica: Teoria de Circuitos, Sexta Edicion, PRENTICE HALL, ISBN 968-880-805-9.

Coughlin Robert, Amplificadores operacionales y circuitos integrados lineales, Quinta Edicion, PRENTICE HALL, ISBN 970-17-0267-0.

Dorf Richard, CIRCUITOS ELECTRICOS Introducción al Analisis y Diseño, Tercera Edicion, Alfaomega, ISBN 970-15-0517-4.

Galeano Gustavo, Programación de Sistemas Embebidos en C, Primera Edicion, Alfaomega, ISBN 978-958-682-770-6.

Ogata Katsuhiko, Ingenieria de Control Moderna, Tercera Edicion, PRENTICE HALL, ISBN 970-17-0048-1.

M. Morris Mano, Logica Digital y Diseño de Computadores, Primera Edicion, PRENTICE HALL, ISBN 968-880-016-3.

Oppenheim Alan, Señales y Sistemas, Segunda Edicion, PEARSON PRENTICE HALL, ISBN 970-17-0116-X.

Proakis John, Tratamiento digital de señales, Cuarta Edicion, PEARSON PRENTICE HALL, ISBN 978-84-8322-347-5.

Índice

1

16F628A · 58, 162, 166, 176, 264, 265, 266, 268, 270, 273, 275, 276
16F84A · 39, 46, 47, 49, 52, 58
16F877A · 65, 71, 75, 77, 79, 89, 94, 110, 113, 118, 120, 121, 123, 124, 126, 129, 130, 133, 137, 140, 146, 170, 174, 189, 241
18F2550 · 88, 95, 96
18F452 · 71, 150, 159, 181, 202, 229, 231
18F4550 · 88, 95
18F4585 · 234, 238

2

24LC00 · 89

7

74LS04 · 176

A

ACK · 88, 141, 143, 244, 245, 246, 247, 248, 249, 250, 251, 256, 257, 259, 260, 279
acknowledge · 88
Actuadores AC · 273
Actuadores DC · 262
Actuadores y potencia · 262
AD · 109, 128, 129, 132, 136, 139, 202
ADC · 17, 109, 124, 128, 129, 130, 133, 136, 137, 140, 173, 203, 206, 216, 218, 223, 230, 236
ADC_Read · 109
ADCON1 · 90, 129, 136
AND · 25, 26, 27, 30, 177
Anexos · 279
ASCII · 20, 21, 50, 54, 59, 60, 61, 85, 86, 87, 94, 95, 126, 149, 175, 244, 279
ASK · 161, 174, 175

B

Bibliografía · 281
bilineal · 212, 213, 219, 221, 239
bit · 20, 21
BJT · 262, 263
Button · 74, 76, 91, 173, 264, 266, 269, 272, 276
BUTTON · 35, 37, 75, 79, 89, 118, 126, 166, 174, 231, 265, 269, 270, 273, 276
ByteToStr · 59, 80, 189

C

CAP · 113, 174, 231
Características básicas del ISIS para simular · 35

Ch

char · 20, 21

C

COM · 47, 167
COMPIM · 159
Comunicación con dispositivos · 147
Comunicación con memorias SD · 177
Comunicación RS 485 · 172
Comunicaciones seriales · 88
Conicionales e iteraciones en lenguaje C · 30
Control de display de 7 segmentos · 45
Control de displays 7 segmentos dinámicos · 47
Control de errores · 252
Control digital PID · 238
Conversión AD y DA · 109
Conversión AD, o ADC · 109
Conversión DA con arreglo R-2R · 114
Conversión DA con PWM · 110
Conversión DA o DAC · 110
Convolución · 192
coordenadas · 55, 72, 73, 148, 149, 151, 153, 155, 158
CRC · 252, 253, 254, 255, 256, 257, 258, 260
Creación de caracteres propios · 59
Creación de condiciones lógicas en lenguaje C · 30
Creación de un programa en lenguaje C · 29
Creación del primer programa en MikroC PRO · 38
cruce por cero · 273, 275, 277

D

DA · 110, 114, 203
DB9 · 92
Declaración de variables en lenguaje C · 20, 244, 247, 252
Definit.h · 99, 100
delay_ms · 46, 47, 48, 73, 91, 99, 130, 133, 137, 140, 142, 144, 145, 164, 165, 173, 176, 184, 185, 189, 264, 272, 276
delay_us · 46, 113, 116, 142, 163, 164, 272
DIN · 84
Dip-Switch · 76, 77, 78
display · 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 64, 65, 70, 73, 79, 83, 87, 154, 158, 159
Display de 7 segmentos · 44
Display LCD de caracteres · 50
Display LCD gráficos · 64

double · 20, 21
DS1307 · 183, 184, 189
DTF · 228

E

EEPROM · 117, 118
EEPROM_Read · 117
EEPROM_Write · 117
Ejemplo de diseño filtro rechaza banda · 222
Ejemplo de diseño oscilador doble cuadrado · 225
Ejemplo de diseño para filtro multi banda · 210
Ejemplo de diseño para filtro pasa altas · 207, 217
Ejemplo de diseño para filtro pasa bajas · 204, 215
Ejemplo de diseño para filtro pasa banda · 208, 220
Ejemplo de diseño para filtro rechaza banda · 209
Ejemplo de diseño para oscilador de acople en cuadratura · 226
Ejemplos de diseño para filtros FIR · 202
El autor · 11
El ciclo iterativo *for* · 33
El ciclo iterativo *while y do while* · 32
El compilador MikroC PRO · 18
El microcontrolador PICMicro · 17
El simulador ISIS de Proteus · 34
ENQ · 246
EOT · 244, 245, 246, 247, 249, 250, 251, 257, 260, 279

F

FFT · 232, 233, 234, 235, 237
Filtro Multi Band · 196
Filtro Pasa Altas IIR · 216
Filtro Pasa Bajas IIR · 213
Filtro Pasa Banda · 194
Filtro Pasa Banda IIR · 219
Filtro Rechaza Banda · 195
Filtro Rechaza Banda IIR · 221
Filtros FIR · 192
Filtros IIR · 212
Filtros Pasa Altas · 194
FIR · 191, 192, 193, 196, 197, 202, 212, 213, 215, 226, 229
FLASH · 117, 118, 119, 120
FLASH_Read · 119, 120
FLASH_Write · 118, 120
float · 20, 21
FloatToStr · 58, 146, 154, 157, 158, 237
Formatos numéricos usados en el lenguaje C · 23
FSK · 161
función de transferencia · 131, 134, 135, 136, 137, 191, 193, 194, 195, 197, 205, 212, 214, 219, 221, 239
Funciones de transferencia · 191
Funciones en lenguaje C · 27
Funciones para imprimir cadenas de texto · 55
Funciones para imprimir caracteres · 54
Fundamentos de lenguaje C · 20, 243

G

Gibbs · 197, 198, 199, 200, 201, 202
GIE · 124, 125

GLCD · 65, 66, 67, 69, 71
GLCD Bitmap Editor · 66
Glcd_Box · 72, 73
Glcd_Circle · 72, 73
Glcd_Dot · 72, 73
Glcd_Fill · 70, 71, 73
Glcd_Init · 65, 70, 73
Glcd_Line · 72, 73
Glcd_Rectangle · 72
GP2D12 · 134, 137
GPS · 88, 147, 148, 149, 152, 160, 161
GROUND · 36

H

HID · 88, 98, 99, 100, 101, 103, 104, 105, 107, 108
HID Terminal · 100, 107
Hid_Disable · 99
Hid_Enable · 98, 99
Hid_Read · 99
Hid_Write · 99

I

I2C · 17, 88, 90, 184, 189
PC · 88, 141, 183
I2C1_Init · 88, 90, 184
I2C1_Is_Idle · 88
I2C1_Rd · 88, 90, 184, 185, 186
I2C1_Repeated_Start · 88, 90, 184, 185, 186
I2C1_Start · 88, 90, 91, 183
I2C1_Stop · 89, 91, 184, 185, 186
I2C1_Wr · 89, 90, 91, 183, 184, 185, 186
I2CDEBUGGER · 89
IEEE 801.15.4 · 167
IIR · 191, 212, 215, 226, 229, 239
Impresión de valores numéricos · 57
infrarrojos · 174, 176
int · 20, 21
INTCON · 98, 124, 125, 126, 150, 158, 203, 204, 206, 216, 218, 219, 223, 224, 226, 227, 228, 230, 231, 236, 237, 275
INTE · 124, 125
Interrupciones · 124
INTF · 100, 103, 104, 124, 125
Introducción · 15
IntToStr · 57, 58, 130, 133, 137, 153, 156, 157, 159, 237
IRLINK · 175, 176
ISIS · 1, 5, 34, 35, 37, 40, 47, 49, 50, 52, 57, 61, 63, 64, 65, 71, 73, 75, 76, 77, 79, 84, 89, 91, 94, 96, 110, 116, 118, 120, 123, 126, 128, 130, 131, 133, 134, 137, 138, 140, 141, 146, 147, 159, 160, 161, 167, 175, 176, 181, 183, 189, 202, 203, 231, 238, 262, 263, 265, 267, 269, 270, 271, 273, 276

K

Keypad_Init · 79
Keypad_Key_Click · 79
Keypad_Key_Press · 79, 80, 82, 83

L

L293D · 241, 270
La sentencia condicional if e if else · 31
La sentencia switch case · 32
LCD · 44, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 71, 72, 73, 80, 82, 83, 86, 87, 129, 132, 136, 137, 139, 140, 146, 154, 158, 159, 169, 170
LCD Custom character · 60
Lcd_Ch · 54, 55, 59, 83, 87
Lcd_Chr_Cp · 54, 55, 59
Lcd_Cmd · 64, 80, 83, 87, 129, 132, 137, 140, 146, 158
Lcd_Init · 51, 52, 54, 55, 56, 57, 58, 61, 63, 80, 83, 87, 129, 132, 136, 140, 146, 158, 169
Lcd_Out · 55, 56, 57, 58, 80, 83, 87, 129, 130, 132, 133, 137, 140, 146, 158, 159, 170
Lcd_Out_Cp · 55, 57
LDR · 138, 139, 140
LED · 35, 44, 75, 77, 89, 96, 110, 118, 126, 166, 189, 231, 274
Library Manager · 51
LM35 · 128, 130, 131
long · 20, 21
LongToStr · 59, 140, 180, 181
LongWordToStr · 59

M

main · 29, 30, 39, 46, 48, 49, 52, 54, 55, 56, 57, 58, 61, 63, 65, 70, 72, 75, 76, 77, 80, 83, 84, 87, 90, 94, 98, 109, 112, 115, 117, 119, 122, 124, 126, 129, 132, 136, 139, 145, 150, 158, 165, 166, 169, 172, 175, 178, 188, 204, 206, 216, 219, 223, 226, 228, 231, 237, 264, 266, 268, 272, 275
matriz de rotación · 224, 225, 227
MAX232 · 93
MAX485 · 172, 173, 174
MAX487 · 174
MCLR · 40
Memoria EEPROM · 117
Memoria FLASH · 118
Memorias EEPROM y FLASH · 117
MikroC PRO · 1, 5, 13, 15, 18, 20, 38, 45, 50, 51, 59, 62, 65, 66, 71, 72, 74, 79, 80, 84, 88, 89, 93, 96, 98, 99, 109, 112, 117, 124, 177, 190
Mmc_Init · 178
Mmc_Read_Sector · 178, 181
Mmc_Write_Sector · 178, 181
MOC3011 · 273
MODEM · 167
Módulo de acople con un ordenador personal · 168
Modulo serial I^C · 88
Módulo USART · 91
Módulo USB · 95, 108
Módulos GPS · 147
Módulos inalámbricos bidireccionales · 167
Módulos inalámbricos infrarrojos · 174
Módulos inalámbricos unidireccionales · 161
Módulos Timer · 121
MOSFET · 262, 263
MOTOR · 265, 269, 270, 273
motor unipolar · 267, 268, 270
Motores AC · 262

motores bipolares · 270
Motores DC · 264
Motores Paso · 267
MPX4115 · 131, 133
Muestreo · 190

N

NAK · 244, 245, 246, 247, 248, 249, 250, 257, 259, 260, 261, 279
NOT · 25, 26, 30
Nyquist · 190

O

Operadores en lenguaje C · 23
OPTION_REG · 122, 126, 165, 173, 264, 266, 268, 272, 275
opto acopladores · 275
OR · 25, 26, 27, 30
Osciladores digitales · 224
OSCILLOSCOPE · 113, 123, 126, 202, 276

P

PICMicro · 17, 18, 34, 38, 109, 190
PID · 99, 103, 238, 239, 241
PIE1 · 98, 124, 125, 126, 150, 158
PIE2 · 98, 124, 125, 126, 150, 158
PIR1 · 98, 124, 125, 126, 150, 157, 158
PIR2 · 98, 124, 126, 150, 158
PLL · 98, 202
PORTA · 48, 49, 90, 91
PORTB · 30, 39, 40, 46, 48, 75, 76, 77, 90, 91, 98, 99, 109, 115, 116, 117, 119, 122, 125, 126, 158, 165, 166, 172, 173, 203, 204, 206, 216, 219, 223, 224, 226, 228, 231, 237, 264, 266, 268, 269, 272, 275, 276
POWER · 36, 100
Prologo · 13
PS2 · 74, 84, 86, 122
Ps2_Config · 84
Ps2_Key_Read · 84, 87
Puente H · 265
PULL-UP · 165, 173, 174, 264, 266, 268, 272
PWM · 17, 109, 110, 111, 112, 113, 114, 115, 116, 162, 173, 175, 176, 264, 271, 272, 275, 276
PWM1_Init · 112, 173, 175, 264
PWM1_Set_Duty · 112, 113, 173, 176, 264
PWM1_Start · 112, 173, 175, 264
PWM1_Stop · 112

R

R-2R · 114, 203
RAM · 18, 59, 63, 103, 177, 234
RCIE · 124, 125
RCIF · 124, 125
recorte de fase · 273, 274, 275, 276, 277
Relevadores · 263

reloj · 84, 88, 91, 98, 121, 141, 142, 143, 145, 159, 162, 178, 183, 188, 189, 202, 241, 268, 273, 275
Reloj en tiempo real · 183
RES · 35, 47, 49, 75, 77, 79, 89, 96, 110, 113, 118, 126, 130, 137, 140, 146, 166, 174, 176, 189, 202, 231, 265, 276
RS 485 · 172
RS232 · 34, 92, 93

S

SD · 88, 105, 177, 178, 179
Sensor de temperatura LM35 · 128
Sensores · 128
Sensores de distancia · 134
Sensores de humedad y temperatura · 141
Sensores de presión · 131
Sensores LDR · 138
Servomotores · 271
short · 20, 21
ShortToStr · 59
SHT71 · 146
SHT7x · 141, 145, 146
SPI · 88, 177, 178, 181
SPII_Init_Advanced · 178, 181
switch case · 30, 32, 82, 83

T

teclado · 18, 78, 79, 80, 81, 82, 83, 84, 86, 87
Teclados y sistemas de entrada de datos · 74
Timer · 17, 121, 122, 125, 126, 275
Timer 0 · 121, 122
TMR0IE · 124, 125
TMR0IF · 124, 125
Transformación bilineal · 213
Transformada discreta de Fourier DFT · 228
Transformada rápida de Fourier FFT · 232
Transmisión · 243
Tratamiento digital de señales · 190
TRIAC · 273
TRISA · 49
TRISB · 30, 39, 40, 46, 49, 75, 76, 77, 90, 98, 109, 115, 117, 119, 122, 126, 158, 165, 166, 172, 204, 206, 216, 219, 223, 226, 228, 231, 237, 266, 268, 272, 275
TRISE · 90
TTL · 172
typedef struct · 22

U

UART1_Data_Ready · 93, 95, 119, 170, 173, 179, 180, 181
UART1_Init · 93, 94, 119, 126, 150, 158, 169, 173, 175, 178, 189, 237
UART1_Read · 93, 95, 119, 126, 150, 157, 170, 173, 179, 180, 181
UART1_Read_Text · 93

UART1_Tx_Idle · 93
UART1_Write · 93, 94, 95, 119, 120, 126, 170, 173, 176, 178, 179, 180, 181, 189, 237
UART1_Write_Text · 94, 95, 119, 120, 126, 170, 176, 178, 179, 180, 181, 189, 237
unsigned char · 21
unsigned int · 21
unsigned long · 21
unsigned short · 21
USART · 17, 18, 58, 88, 91, 93, 94, 124, 149, 169, 172, 173, 175, 189
USB · 18, 34, 88, 95, 96, 98, 99, 100, 101, 102, 103, 105, 107
USBCONN · 96
USBdesc.c · 99, 107
Uso anidado de ciclos iterativos · 33
Uso de Dip-Switch · 76
Uso de pulsadores · 74
Uso de teclados matriciales · 78
Uso de teclados PS2 o Din · 84

V

variable compleja · 191, 212
VARs.h · 99, 104
Ventanas Blackman · 201
Ventanas fijas · 196
Ventanas Hamming · 198
Ventanas Hanning · 199
Ventanas Rectangulares · 197
VIP · 99
VIRTUAL TERMINAL · 94, 120, 126, 170, 176, 181, 189, 238
Visualización de datos · 44
VSINE · 276
VUSB · 96

W

WordToStr · 59
www.digi.com · 167
www.labcenter.com · 34
www.mikroe.com · 18
www.virtual-serial-port.com · 147

X

XBee · 88, 167, 169
XBee PRO · 167
X-CTU · 167, 169

Z

Zigbee · 169

