UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES
ARAGÓN

SISTEMA DE ADQUISICIÓN DE DATOS PARA EXTENSIOMETRÍA ELÉCTRICA POR MEDIO DE USB

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO MECÁNICO ELECTRICISTA
P R E S E N T A :
CARLOS FERNANDO ORTEGA NAVA

ASESOR:
ING. ARTURO OCAMPO ÁLVAREZ

FES Aragón

MÉXICO 2011.

Índice

Introducción

La realización del presente estudio se basa en la implementación de microcontroladores, tomando en cuenta el diseño lógico, la programación, etc. que son parte de los requerimientos fundamentales para construir un dispositivo electrónico. El primer capítulo cuenta con una síntesis de los conocimientos necesarios, el segundo capítulo tiene como función establecer un orden y una explicación paso a paso en la implementación del sistema, y el último capítulo contiene los ajustes o modificaciones que conlleva el darle un toque final, y presentación del proyecto.

PLANTEAMIENTO DEL PROBLEMA

El problema surge con la necesidad de dar continuidad a los avances tecnológicos que se dan día a día, tomando en cuenta nuevas tecnologías. Dando seguimiento a las nuevas tecnologías, se pueden elaborar mejores proyectos, que funcionen con las nuevas herramientas, como la computadora personal (PC) y su interface de comunicación USB (Universal Serial Bus), convertidores analógico digital más potentes y rápidos, programas con software más complejo, entonces se da a la tarea de eslabonar estos recursos disponibles para facilitar la elaboración de un proyecto que cumpla con las nuevas perspectivas, como es en este caso la medición de esfuerzos y deformaciones en prácticas de laboratorio de IME (Ingeniería Mecánica Eléctrica). Surge una pregunta: ¿Cómo realizar un sistema de adquisición de datos de alta resolución utilizando comunicación por puerto USB, para medición de esfuerzos y deformaciones con extensiometría eléctrica?

OBJETIVOS

OBJETIVO GENERAL

Implementar un sistema de adquisición de datos de alta resolución, utilizando el puerto USB, para la medición de esfuerzos y deformaciones con extensiometría eléctrica.

OBJETIVOS PARTICULARES

Utilizar un microcontrolador para el control de la función.
Buscar un convertidor analógico digital de alta resolución, para extensiometría.
Manejar el protocolo de comunicación USB que cumpla con las necesidades del proyecto.
Elaborar el software necesario para el control del dispositivo.

DELIMITACIÓN Y JUSTIFICACIÓN

Este proyecto es una adaptación de versiones anteriores que utilizan otros protocolos de comunicación con la PC, que realizan la función de adquisición de datos por extensiometría para la medición de esfuerzos y deformaciones. Con base en esto se conoce bien la función o el objetivo del mismo, por lo que en síntesis se busca hacer una nueva versión de un dispositivo que transforme las pequeñas señales eléctricas analógicas, que recibe de las galgas extensiométricas, en señales digitales de alta resolución, que posteriormente serán enviadas por puerto USB a la computadora, la cual con el uso de un programa bien identificado recibe estos datos digitales para que sean procesados y mostrados en el monitor de la computadora.

Se toma la decisión de realizar este proyecto tomando en cuenta que se requieren relativamente pocos elementos electrónicos para la construcción del dispositivo en función, esto hace del proyecto un sistema muy económico en comparación con los sistemas que ofrecen las empresas. Es cierto que este dispositivo no puede competir con modelos creados, por ejemplo por National Instruments, pero fue implementado por con muy pocos recursos y herramientas, pero se pueden cumplir perfectamente con los objetivos de una práctica de laboratorio en la escuela. Esto hace de este tipo de proyectos una opción viable, flexible y de múltiples beneficios.

Como se usa un microcontrolador, también se debe hablar de programación, por lo cual se toma en cuenta que se deben poseer conocimientos de programación en varios lenguajes, así como el software al que se puede recurrir y si existe suficiente fuente de información disponible, ya sea en el caso de programación o de los dispositivos que integran el dispositivo, como son los manuales. También se puede conseguir casi cualquier dispositivo en alguna tienda de electrónica, incluso los microcontroladores o los convertidores analógico digital, y en caso de no haber en existencia, se pueden pedir muestras gratis con un costo de envío. El periodo de entrega es corto, máximo un mes. En este caso, si se pide con anticipación, se puede proceder sin ningún retraso.

El propósito es contar con una máquina didáctica para la elaboración de prácticas en el laboratorio de IME.

1. Fundamentos

Este capítulo expone los conocimientos necesarios para la comprensión del proyecto de este trabajo. Ya que en caso de ser nuevo en el tema, el lector tendrá la necesidad de recurrir a los fundamentos, para el estudio, análisis, implementación y pruebas del sistema.

Cabe señalar que esta obra no tiene la intención de discernir conocimientos básicos de cada materia como tal, ya que muchos subtemas del capítulo FUNDAMENTOS es muy amplio, en cambio resulta útil una síntesis con los conceptos importantes.

1.1. DISEÑO LÓGICO

El diseño lógico es un proceso ordenado que tiene la finalidad de implementar una o más expresiones reducidas, simplificadas y lógicas, haciendo uso de diversos métodos, mencionando el álgebra de Boole como herramienta importante para la solución y/o representación de un problema.

El álgebra de Boole no surge como un sistema algebraico para la solución y/o representación de la lógica de los circuitos, sino que tiene sus inicios en la teoría de conjuntos y el cálculo proposicional. La teoría de conjuntos parte de manera natural donde se llevan a cabo procesos sencillos de abstracción, y el cálculo proposicional que maneja oraciones simples unidas mediante conectivas. Estas teorías juegan un papel importante en las aportaciones de Boole. Lo mencionado hasta ahora es para saber que la aportación de Boole fue inicialmente desarrollado para el estudio de la lógica, y a partir de 1938 Claude Shannon publicó su obra Análisis Simbólico de Circuitos con Relés, y habla de los primeros conceptos de la actual teoría de conmutación.

En la actualidad esta herramienta sigue siendo tan vigente como es de esperarse de un análisis matemático. El algebra de Boole tiene una característica singular, pues debe plantearse a la variable como un elemento bivalente. En esta obra no necesitamos hacer un uso directo de los teoremas y postulados del algebra de Boole pero mencionaremos las reglas para justificar el comportamiento de las funciones lógicas que sí tienen más relación con lo que trata el tema principal.

También se hará mención de manera muy superficial y teórica sobre circuitos combinacionales, secuenciales, memorias, etc., que en algunos casos, como en el de la memoria, resulta indispensable conocer el tema.

1.1.1. SISTEMAS DE NUMERACIÓN

Un sistema numérico es una representación de números, es decir, un conjunto de símbolos y reglas para representar cantidades de forma secuencial.

Tiempo atrás, el hombre ha recurrido a diversos sistemas de numeración, como el babilónico con símbolos de cuña, la numeración azteca que se desarrollo en México durante la época prehispánica, el egipcio con sus jeroglíficos, el romano que aún es popular y se utiliza comúnmente para dar un orden al temario de algún libro, también tenemos un sistema numérico sorprendente, el maya que cuenta con el símbolo cero y con un orden posicional, y se definiría como un sistema vigesimal (base 20). Pero, ¿Por qué es tan sorprendente el cero?, ¿A que nos referimos con sistema posicional? ¿Qué es la base? Bueno, para comprender empezaremos con el sistema actual y más común para nosotros. Este sistema es el indoarábigo que debe su nombre a su origen en la india y a su divulgación por los árabes y cuenta con diez símbolos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), que gracias a la posición que ocupa contando a partir del punto a la izquierda, adquiere un valor superior o, con una posición a partir del punto decimal hacia la derecha, adquiere un valor menor. El sistema indoarábigo es un sistema base 10, lo que quiere decir que usa potencias de base 10 para indicar distintos órdenes de magnitud. Para que quede más claro, ver la Tabla (1.1 - 1).

10^n	10^4	10^3	10^2	10^1	10^0	.	10^{-1}	10^{-2}	10^{-n}
10(10)…	10000	1000	100	10	1	Punto	0.1	0.01	0.00…1

Tabla (1.1 - 1) Ejemplo de la Base decimal.

REPRESENTACIÓN Y LECTURA

La transmisión verbal y precisa de los números que respeten las reglas de un sistema numérico, requiere de un procedimiento para indicar la posición de cada símbolo elemental en el número expresado. Entonces los números se dividen en el siguiente orden y que a partir del punto son unidades, decenas, centenas etc. Ver la tabla (1.1-2).

1	3	8	9	6	5	3	1	8	7.
unidad de millar de millón	centenas de millón	decenas de millón	unidades de millón	centenas de millar	decenas de millar	unidad de millar	centenas	decenas	unidades
posición 10	posición 9	posición 8	posición 7	posición 6	posición 5	posición 4	posición 3	posición 2	posición 1
millares de millón	millones			millares			unidades simples		
se lee: Mil trescientos ochenta y nueve millones, seiscientos cincuenta y tres mil, ciento ochenta y siete unidades.									

Tabla (1.1 - 2) Representación y Lectura de la Base Decimal.

La intención es recordar cómo se maneja un sistema numérico, ya que para los demás sistemas se aplica las mismas reglas, a excepción de la lectura.

Ahora los siguientes sistemas numéricos, la diferencia es la base. Estos son los que se usan para trabajar con la computadora. Así que nos familiarizaremos con ellos.

Estos son:

- Binario
- Octal
- Hexadecimal

SISTEMA NUMÉRICO BINARIO

Símbolos

- 0 Cero
- 1 Uno

Este sistema es de base dos, esto quiere decir que tiene potencias de dos (2^n), donde 2 es la base y n es la posición que ocupa el símbolo. Ver la tabla (1.1-3).

Número base2	1	0	1	0	1	1
Representación en potencia	$1(2^5)+$	$0(2^4)+$	$1(2^3)+$	$0(2^2)+$	$1(2^1)+$	$1(2^0)$
43 =	*32+*	*0+*	*8+*	*0+*	*2+*	*1+*

Tabla (1.1 - 3) El número 101011_2 (base2) es igual a 43_{10} (base 10).

SISTEMA NUMÉRICO OCTAL

Símbolos

- 0 Cero
- 1 Uno
- 2 Dos
- 3 Tres
- 4 Cuatro
- 5 Cinco
- 6 Seis
- 7 Siete

Este sistema es de base ocho, esto quiere decir que tiene potencias de ocho (8^n), donde 8 es la base y n es la posición que ocupa el símbolo. Ver la tabla (1.1-4).

Número base8	4	1	0	2	1	3
Representación en potencia	$4(8^5)+$	$1(8^4)+$	$0(8^3)+$	$2(8^2)+$	$1(8^1)+$	$3(8^0)+$
135307 =	*131072+*	*4096+*	*0+*	*128+*	*8+*	*3+*

Tabla (1.1 - 4) El número 410213_8 (base 8) es igual a 135307_{10} (base 10).

SISTEMA NUMÉRICO HEXADECIMAL

Símbolos

- 0 Cero
- 1 Uno
- 2 Dos
- 3 Tres
- 4 Cuatro
- 5 Cinco
- 6 Seis
- 7 Siete
- 8 Ocho
- 9 Nueve
- A Diez
- B Once
- C Doce
- D Trece
- E Catorce
- F Quince

Este sistema es de base Dieciséis, esto quiere decir que tiene potencias de dieciséis (16^n), donde 16 es la base y n es la posición que ocupa el símbolo. Ver la tabla (1.15).

Número base16	*1*	*C*	*2*	*1*	*A*
Representación en potencia	$1(16^4)+$	$C(16^3)+$	$2(16^2)+$	$1(16^1)+$	$A(16^0)+$
115226=	*65536+*	*49152+*	*512+*	*16+*	*10+*

Tabla (1.1 - 5) El número $1C21A_{16}$ (base 16) es igual a 115226_{10} (base 10).

Ya conocemos las bases numéricas más importantes. La tabla (1.1-6) resulta de lo anterior, trata de un simple conteo hasta el número 20 base 10, en comparación con su equivalente en otras bases.

Sistemas Numéricos			
Binario	**Octal**	**Decimal**	**Hexadecimal**
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10
10001	21	17	11
10010	22	18	12
10011	23	19	13
10100	24	20	14

Tabla (1.1 - 6) Tabla de conversión entre bases numéricas.

1.1.2. CONVERSIONES ENTRE BASES NUMÉRICAS

BINARIO A OCTAL

Esta conversión es muy sencilla, tomamos nuestro número binario y lo separamos de derecha a izquierda en grupos de 3. Después convertimos esos grupos en su equivalente en octal, por último juntamos los números en el orden en el que estaban.

Ejemplo:

1. Elegir el número binario a convertir.
   Número binario = 1011101100101_2
2. Separar el número en grupos de tres símbolos de derecha a izquierda.
   1 011 101 100 101
3. Convertir los grupos en su equivalente octal (utilizar tabla (1.1-6)).
   1 011 101 100 101

   1 3 5 4 5
4. Juntar los números en el orden en que estaban.
   Número Octal = 13545_8

BINARIO A DECIMAL

Elegimos el número binario a convertir, y hacemos sumas en potencias sucesivas (de 0 a n posición) en base 2, ejecutamos las operaciones en nuestro sistema ordinario.

Ejemplo:

1. Elegir el número binario a convertir.
   Número binario = 1011101100101_2
2. Hacer sumas en potencias sucesivas (de 0 a n posición).

   $1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1$

   Posición = $12\ 11\ 10\ \ 9\ \ 8\ \ 7\ \ 6\ \ 5\ \ 4\ \ 3\ \ 2\ \ 1\ \ 0$

   $1(2)^{12}+0(2)^{11}+1(2)^{10}+1(2)^9+1(2)^8+0(2)^7+1(2)^6+1(2)^5+0(2)^4+0(2)^3+1(2)^2+0(2)^1+1(2)^0$
3. Realizar las operaciones de manera normal.
   $4096+0+1024+512+256+0+64+32+0+0+4+0+1$

   Número decimal = 5989_{10}

BINARIO A HEXADECIMAL

El método de conversión es similar al que usamos para convertir a octal, solo que esta vez separamos en grupos de cuatro.

Ejemplo:

1. Elegir el número binario a convertir.
   Número binario = 1011101100101_2
2. Separar el número en grupos de cuatro símbolos de derecha a izquierda.
   $1\ 0111\ 0110\ 0101$
3. Convertir grupos en su equivalente hexadecimal (utilizar tabla (1.1-6)).
   Número binario $\underline{1}\ \underline{0111}\ \underline{0110}\ \underline{0101}$

   Equivalente en hexadecimal. $1\quad 7\quad 6\quad 5$
4. Juntar los números en el orden en que estaban.
   Número Hexadecimal = 1765_{16}

OCTAL A BINARIO

Se transforma cada número octal en su equivalente binario, conservando la posición u orden que llevaba.

Ejemplo:

1. Tomar el número octal a convertir.
   Número octal = 13545_8
2. Convertir en binario, número a número, conservando su posición original.
   Número octal $\underline{1} \quad \underline{3} \quad \underline{5} \quad \underline{4} \quad \underline{5}$

   Equivalente en binario. $1 \quad 011 \; 101 \; 100 \; 101$
3. Juntar los números.
   Número binario = 1011101100101_2

OCTAL A DECIMAL

Elegimos el número decimal a convertir, y hacemos sumas en potencias sucesivas (de 0 a n posición) en base 8, ejecutamos las operaciones en nuestro sistema ordinario.

Ejemplo:

1. Elegimos el número octal a convertir.
   Número Octal = 13545_8
2. Hacemos sumas en potencias sucesivas (de 0 a n posición).
   Número decimal. $1 \; 3 \; 5 \; 4 \; 5$

   Posición = $4 \; 3 \; 2 \; 1 \; 0$

   $1(8)^4 + 3(8)^3 + 5(8)^2 + 4(8)^1 + 5(8)^0$
3. Realizamos las operaciones de manera normal.
   $4096+1536+320+32+5$

   Número decimal = 5989_{10}

OCTAL A HEXADECIMAL

No existe una conversión directa, lo más práctico es transformar de octal a binario y después de binario a hexadecimal.

DECIMAL A BINARIO

Esto se logra realizando divisiones consecutivas entre dos, ordenando el primer residuo a la izquierda y los siguientes a la derecha consecutivamente, hasta descomponer el número decimal en un cociente 1.

Ejemplo:

1. Elegir el número decimal a convertir.
   Número Decimal = 5989_{10}
2. Realizar divisiones consecutivas entre dos.

$$\frac{5989}{2} = 2994 \; y \; residuo \; 1$$
$$\frac{2994}{2} = 1497 \; y \; residuo \; 0$$
$$\frac{1497}{2} = 748 \; y \; residuo \; 1$$
$$\frac{748}{2} = 374 \; y \; residuo \; 0$$
$$\frac{374}{2} = 187 \; y \; residuo \; 0$$
$$\frac{187}{2} = 93 \; y \; residuo \; 1$$
$$\frac{93}{2} = 46 \; y \; residuo \; 1$$
$$\frac{46}{2} = 23 \; y \; residuo \; 0$$
$$\frac{23}{2} = 11 \; y \; residuo \; 1$$
$$\frac{11}{2} = 5 \; y \; residuo \; 1$$
$$\frac{5}{2} = 2 \; y \; residuo \; 1$$
$$\frac{2}{2} = \mathbf{1} \; y \; residuo \; 0$$
$$El \; cociente \; \mathbf{1}$$

3. Ordenar del primer residuo hacia la derecha hasta el cociente 1.
   Número Binario = 1011101100101_2

DECIMAL A OCTAL

El mismo método, solo que divisiones consecutivas entre 8 hasta un cociente menor o igual a 7.

Ejemplo:

1. Elegir el número Decimal a convertir $5989_{10.}$
   Número Decimal = 5989_{10}
2. Realizar divisiones consecutivas entre *8.*

$$\frac{5989}{8} = 748\ y\ residuo\ 5$$

$$\frac{748}{8} = 93\ y\ residuo\ 4$$

$$\frac{93}{8} = 11\ y\ residuo\ 5$$

$$\frac{11}{8} = \mathbf{1}\ y\ residuo\ 3$$

$$cociente\ igual\ a\ \mathbf{1}$$

3. Ordenar del primer residuo hacia la derecha hasta el cociente menor o igual a *7.*
   Número Octal = 13545_{8}

DE DECIMAL A HEXADECIMAL

Se debe seguir el mismo método anterior, pero con divisiones consecutivas entre 16, que es la base que queremos.

Ejemplo:

1. Elegir el número Decimal a convertir $5989_{10.}$
   Número Decimal = 5989_{10}
2. Realizar divisiones consecutivas entre *16.*

$$\frac{5989}{16} = 374\ y\ residuo\ 5$$

$$\frac{374}{16} = 23\ y\ residuo\ 6$$

$$\frac{23}{16} = \mathbf{1}\ y\ residuo\ 7$$

$$cociente\ igual\ a\ \mathbf{1}$$

3. Ordenar del primer residuo hacia la derecha, hasta el cociente menor o igual a 15. Ojo, el número 10 se escribe como A, el 11 como B etc. Debemos estar atentos al pasar los residuos y el número final (cociente) a su equivalente en Hexadecimal. Si existen problemas con esto, consultar la tabla (1.1 - 6).
   Número Hexadecimal = 1765_{16}

HEXADECIMAL A BINARIO

Se toma cada número en hexadecimal y se transforma uno por uno en su equivalente en binario, conservando el orden.

Ejemplo:

1. Elegir el número hexadecimal a convertir.
   Número Hexadecimal = $1A6F_{16}$
2. Convertir cada número en su equivalente binario, conservando el orden.
   Número hexadecimal. *1 A 6 F*

   Equivalente binario. *0001 1010 0110 1111*
3. Juntar el número.
   Número binario = 1101001101111_2

HEXADECIMAL A OCTAL

No existe una conversión directa, lo más práctico es transformar de hexadecimal a binario, y después de binario a octal.

HEXADECIMAL A DECIMAL

Elegimos el número Hexadecimal a convertir, y hacemos sumas en potencias sucesivas (de 0 a n posición) en base 8, ejecutamos las operaciones en nuestro sistema ordinario.

Ejemplo:

1. Elegir el número Hexadecimal a convertir.
   Número Hexadecimal = $1A6F_{16}$
2. Hacer sumas en potencias sucesivas (de 0 a n posición).
   Número hexadecimal. *1 A 6 F*

   Posición = *3 2 1 0*

   $1(16)^3 + A(16)^2 + 6(16)^1 + F(16)^0 =$

   $1(16)^3 + 10(16)^2 + 6(16)^1 + 15(16)^0$
3. Realizar las operaciones de manera normal.
   $4096 + 2560 + 96 + 15$

   Número decimal = 6767_{10}

1.1.3. CÓDIGO BCD

El código Binary Coded Decimal (Binario Codificado a Decimal) es un sistema numérico que se utiliza mucho en las computadoras. Dedicado a establecer una fácil comunicación entre el código máquina (unos y ceros) y el sistema numérico decimal (0 al 9). Es muy utilizado en teclados, o para desplegar cantidades decimales en display.

El sistema es el mostrado en la Tabla (1.1 - 7):

Decimal	Binario	BCD
0	*0*	*0000*
1	*1*	*0001*
2	*10*	*0010*
3	*11*	*0011*
4	*100*	*0100*
5	*101*	*0101*
6	*110*	*0110*
7	*111*	*0111*
8	*1000*	*1000*
9	*1001*	*1001*

Tabla (1.1 - 7) Código BCD.

Como se observa en la tabla (1.1-7), se debe utilizar cuatro números (unos y ceros) para representar el código BCD, además solo se utilizan los primeros 10 números que se pueden representar con cuatro unos y ceros. A continuación un ejemplo de conversión de un número decimal a BCD.

DECIMAL A BCD

El método es convertir cada número decimal en su equivalente BCD conservando el orden.

Ejemplo:

1. Primero elegimos el número a convertir.
   Número decimal = 8136_{10}
2. Convertir cada uno de los números decimales en su equivalente BCD (Usar tabla).
   Decimal *8 1 3 6*

   BCD *1000 0001 0011 0110*

   Este código no se junta, ya que adquiere significado por separado.

1.1.4. OPERACIONES ARITMÉTICAS BINARIAS

COMPLEMENTO A UNO

Existen dos condiciones que hacen referencia a lo que llamamos complemento a uno, una es la operación matemática, y la otra es la convención que se usa para representar números enteros en código binario, ambas con el mismo nombre; Complemento a uno".

OPERACIÓN MATEMÁTICA COMPLEMENTO A UNO

Si tenemos un número binario, por ejemplo 1011_2, y le aplicamos el complemento a uno, que podemos representar con C'1, daría por resultado 0100_2, observemos que simplemente se invierten los unos por ceros y los ceros por unos. La forma adecuada de escribir esta operación es:

$$C'1(1011_2) = 0100_2$$

CONVENCIÓN COMPLEMENTO A UNO

La convención se utiliza para representar números enteros en código binario, esto es, números enteros positivos y números enteros negativos. Pero, se sabe que en nuestro sistema tradicional para representar los números positivos usamos el símbolo (+) y para los números negativos el símbolo (-), y el código binario en nuestras computadoras solo permite el 1 y, el 0, por lo que se creó esta convención, en donde los números positivos llevan a la izquierda un cero extra para saber que son positivos, y los números negativos se le aplica el complemento a uno a su representación positiva en complemento a uno, quedando así un bit uno de signo negativo.

Ejemplo:

Si tenemos el número $+93_{10}$ que equivale a 1011101_2, entonces su representación en complemento a uno positiva sería 01011101_{c1}, se observa que, no se le aplica el complemento a uno y solamente se le agrega un cero a la izquierda. Ahora, ese mismo número *(93)* pero de forma negativa *(-93)*, utilizamos la representación positiva en complemento a uno (01011101_{c1}) y le aplicamos la operación matemática complemento a uno $C'1(0101101) = 1010010_{c1}$, y el resultado es la representación complemento a uno de -93. Este número (1010010_{c1}) nos dice que es negativo, pero la magnitud no es muy clara, entonces lo que tendríamos que hacer es lo siguiente. Considerando el siguiente número (10001011_{c1}), sabemos es que es negativo por el uno a la izquierda, ahora le aplicamos la operación matemática complemento a uno $C'1(10001011)$, lo que nos dará como resultado (01110100), entonces transformamos a decimal, lo que nos da 116_{10}, por lo tanto, nuestro numero es *-116.*

Por último, se observa que, utilizando este tipo de representación, da por resultado un rango simétrico de números, esto quiere decir que, tenemos la misma cantidad de números positivos y negativos por tener un cero positivo y un cero negativo. La tabla (1.1 - 8) explica esto, considerando una trama de 8 bits, de esta manera se comprenden mejor las ecuaciones generales para rangos, que son las siguientes:

Rango para números positivos = $+[2^{n-1} - (2^{n-1} - j)]$; $j = 1,2,3,\dots,(2^{n-1} - 1)$

Rango para números negativos = $-[2^{n-1} - (2^{n-1} - j)]$; $j = 0,1,2,3,\dots,(2^{n-1} - 1)$

Donde: n = número de bits

Positivos		Negativos	
Binario	Decimal	Binario	Decimal
00000000	*+0*	*11111111*	*-0*
00000001	*+1*	*11111110*	*-1*
00000010	*+2*	*11111101*	*-2*
00000011	*+3*	*11111100*	*-3*
...	*...*	*...*	*...*
01111110	*+126*	*10000001*	*-126*
01111111	*+127*	*10000000*	*-127*

Tabla (1.1 - 8). Números positivos y negativos en complemento a uno, en una trama de 8 bits.

COMPLEMENTO A DOS

Al igual que el complemento a uno, existen dos condiciones, la primera es una operación matemática y la otra es una representación, ambas con el mismo nombre; Complemento a dos (C'2).

OPERACIÓN MATEMÁTICA COMPLEMENTO A DOS

Si tenemos el número 101101_2 y le queremos aplicar la operación matemática complemento a dos, primero se cambian ceros por unos y unos por ceros (*010010*), y por último se suma 1, entonces queda *010011*. El siguiente ejemplo es más formal:

$$C'2(010110_2) = (101001)+1 = 101010_{c2}$$

CONVENCIÓN COMPLEMENTO A DOS

Por ejemplo, con el número $+48_{10}$ que, en binario es 110000_2, si se representa en un byte (8bits), en convención complemento a dos, lo que da *00110000* ; aunque le agregamos dos ceros a la izquierda, solo es para representarlo en un byte, lo importante es el último cero a la izquierda que indica que se trata de un número entero positivo. Ahora para representarlo como un número entero negativo, se utiliza su representación entera positiva en complemento a dos (*00110000*) y le aplicamos la operación matemática complemento a dos. Entonces $C'2(00110000) = (11001111)+1$ que es igual a 11010000_{c2}, se observa que, el bit más significativo (primer bit a la izquierda) es el bit de signo y es un 1, lo que por convención nos dice que es negativo, pero no se ve claramente la magnitud.

Utilizando un ejemplo con el número 10111011_{c2}, se ve por el bit de signo que se trata de un número negativo, por el contrario, no podemos decir lo mismo de la magnitud que no se ve claramente. En este caso se aplica la operación complemento a dos a 10111011_{c2}, entonces: $C'2(10111011) = (01000100)+1 = 01000101$ que en decimal es *69,* por lo tanto $10111011_{c2} = -69_{10}$.

Por último, con este método nos damos cuenta de que el rango de representación de números positivos y negativos es distinto, ya que solo existe un cero y se considera positivo. La tabla (1.1 - 9) explica esto, considerando una trama de 8 bits, de esta manera se comprende mejor la ecuación general para rangos, que es la siguiente:

Rango para números positivos = $+[2^{n-1} - (2^{n-1} - j)]; \quad j = 0,1,2,3,\dots,(2^{n-1} - 1)$

Rango para números negativos = $-[2^{n-1} - j]; \quad j = 0,1,2,3,\dots,(2^{n-1} - 1)$

Donde: n = número de bits

Positivos		Negativos	
Binario	Decimal	Binario	Decimal
00000000	*+0*	*10000000*	*-128*
00000001	*+1*	*10000001*	*-127*
00000010	*+2*	*10000010*	*-126*
00000011	*+3*	*10000011*	*-125*
...	*...*	*...*	*...*
01111110	*+126*	*11111110*	*-2*
01111111	*+127*	*11111111*	*-1*

Tabla (1.1 - 9). Tabla de números positivos y negativos en complemento a dos, demostrando que su rango no es simétrico, con una arquitectura de 8 bits.

1.1.5. ÁLGEBRA DE BOOLE

SIMBOLOGÍA

- A, B, C. . . n. Son variables y solo toman dos estados de conmutación, el cero “0“ y el uno “ 1 “.
- AND o Y (PRODUCTO), OR u O (SUMA) y NOT o NO (NEGACIÓN) como operaciones fundamentales.

La operación más básica es:

OPERACIÓN	SIMBOLO
NOT	$\bar{}$

Entonces tenemos:

Expresión Se lee

- $\bar{0} = 1$ Cero negado es igual a uno
- $\bar{1} = 0$ Uno negado es igual a cero
- $\bar{\bar{A}} = A$ A doble negada es igual a A

La segunda operación en orden de precedencia es:

OPERACIÓN	SIMBOLO
AND	.

Entonces tenemos:

Expresión Se lee

- $0 \cdot 0 = 0$ Cero AND cero igual a cero
- $0 \cdot 1 = 0$ Cero AND uno igual a cero
- $1 \cdot 0 = 0$ Uno AND cero igual a cero
- $1 \cdot 1 = 1$ Uno AND uno igual a uno
- $0 \cdot A = 0$ Cero AND “A” igual a cero
- $A \cdot 0 = 0$ “A” AND cero igual a cero
- $1 \cdot A = A$ Uno AND “A” igual a “A”
- $A \cdot 1 = A$ “A” AND uno igual a “A”
- $A \cdot A = A$ “A” AND “A” igual a “A”
- $A \cdot \bar{A} = 0$ “A” AND “A” negada igual a cero
- $\bar{A} \cdot A = 0$ “A” negada AND “A” igual a cero

Última operación fundamental en orden de precedencia es:

OPERACIÓN	SIMBOLO
OR	+

Entonces tenemos los siguientes casos:

Expresión Se lee

- $0 + 0 = 0$ Cero OR cero igual a cero
- $0 + 1 = 1$ Cero OR uno igual a uno
- $1 + 0 = 1$ Uno OR cero igual a uno
- $1 + 1 = 1$ Uno OR uno igual a uno
- $0 + A = A$ Cero OR “A” igual a “A”
- $A + 0 = A$ “A” OR cero igual a “A”
- $1 + A = 1$ Uno OR “A” igual a uno
- $A + 1 = 1$ “A” OR uno igual a uno
- $A + A = A$ “A” OR “A” igual a “A”
- $A + \bar{A} = 1$ “A” OR “A” negada igual a uno
- $\bar{A} + A = 1$ “A” negada OR “A” igual a uno

Ahora, de manera formal, el orden de precedencia.

NOT	PRIMERO
PARENTESIS	
AND	
OR	ÚLTIMO

Las leyes del algebra tradicional sirven también para el algebra de Boole. Estas son:

LEY CONMUTATIVA
$A \cdot B = B \cdot A$
$A + B = B + A$

LEY ASOCIATIVA
$(A \cdot B) \cdot C = A \cdot (B \cdot C) = A \cdot B \cdot C$
$(A + B) + C = A + (B + C) = A + B + C$

LEY DISTRIBUTIVA
$(A \cdot B) + (A \cdot B) = A \cdot (B + C)$
$(A + B) \cdot (A + C) = A + (B \cdot C)$

1.1.6. SISTEMAS DIGITALES

SISTEMA DIGITAL

Un sistema digital es cualquier dispositivo destinado a la generación, transmisión, procesamiento o almacenamiento de señales digitales. También, es una combinación de dispositivos diseñados para manipular cantidades físicas o información que estén representadas en forma digital; es decir, que sólo puedan tomar valores discretos.

COMPUERTAS

Las compuertas son los elementos más básicos que componen un circuito integrado. Ver la figura (1.1-1). La forma en que se deben unir estas compuertas depende de cómo se estructure la ecuación en el álgebra de Boole, que a su vez depende del planteamiento del problema.

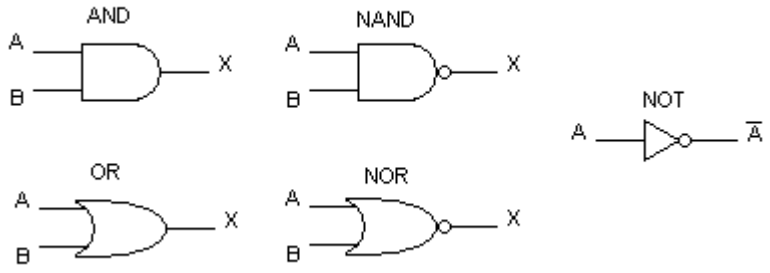


Figura (1.1 - 1) Compuertas básicas.

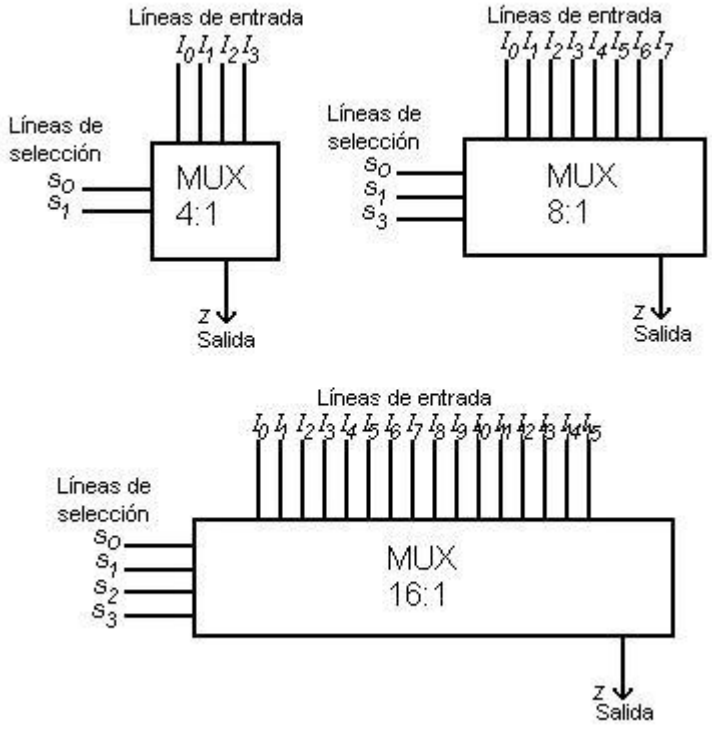
CIRCUITO COMBINACIONAL

Este tipo de circuito digital (señales de unos y ceros) se caracteriza por ser relativamente sencillo, y se comporta como una relación, en donde la función ó señales de salida, corresponden a la o las variables (señales de entrada o combinaciones de entrada), recordando que la función está compuesta por un arreglo algebraico de Boole. Esto lo podemos representar de forma general. Ver la figura (1.1-2).

Figura (1.1 - 2) Diagrama a bloques del circuito combinacional.

MULTIPLEXOR

El multiplexor es un circuito combinacional que consta de varias entradas, una salida y líneas de selección. Su operación es muy sencilla, las líneas de selección, escogen cual señal de entrada es la que va a salir. Ver la figura (1.1-3)



Figura(1.1 - 3) Multiplexores (MUX) de 4:1, 8:1, 16:1.

DEMULTIPLEXOR

También es un circuito combinacional, y realiza una operación similar a la del multiplexor, en este caso la salida z la toma como entrada y las líneas de selección eligen la línea donde saldrá la señal de entrada.

CODIFICADOR

El codificador, es un dispositivo que toma una información y la representa en otro código, para que otro dispositivo la pueda interpretar.

DECODIFICADOR

La función de este dispositivo es inversa a la del codificador. Este elemento también es un circuito combinacional, el cual revierte el cambio de código en la información. En pocas palabras, este circuito revierte el proceso del codificador.

CIRCUITO SECUENCIAL

Este circuito se caracteriza por mandar una salida diferente, aún con la misma combinación dependiendo de la secuencia que la precede. Esto quiere decir que tiene la particularidad de poseer retroalimentación y memoria. Este tipo de circuitos generalmente hacen uso de los flip-flops, que conceptualizaremos más adelante. También pueden hacer uso de un oscilador, lo que quiere decir que actúan automáticamente y mandan salidas continuamente, dependiendo de la frecuencia del oscilador. En resumen, está compuesto por un oscilador, un circuito combinacional y por flip-flops. En la figura (1.1-4) muestra la forma general de un circuito secuencial con señales de entrada que, pueden no existir dependiendo del diseño. Los circuitos secuenciales más comunes son los contadores.

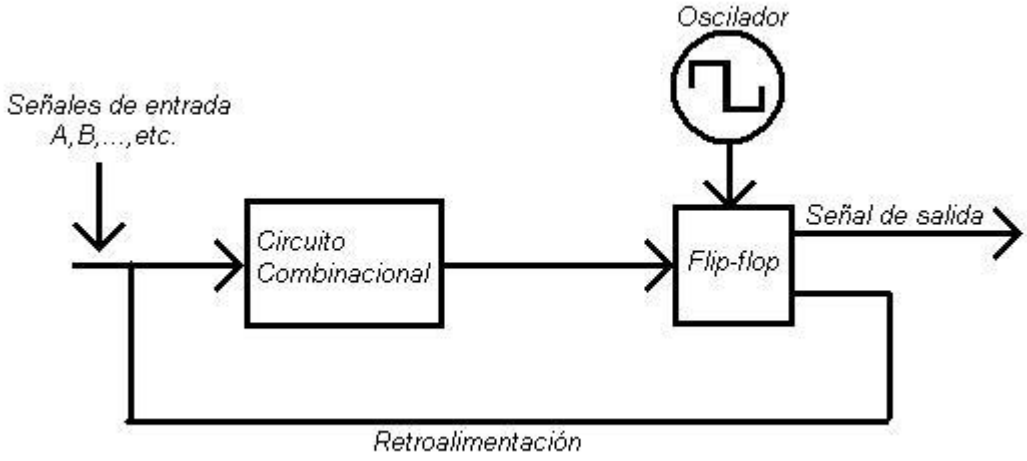


Figura (1.1 - 4): Diagrama a bloques del circuito secuencial.

FLIP-FLOPS

También conocidos con el nombre de registros básicos, o biestable. Estos dispositivos se caracterizan por tener dos estados, el presente (Q(t)) y el futuro (Q(t+1)), donde el estado futuro depende de la variación de sus entradas. Los tipos de flip-flops son los siguientes: se identifican por sus entradas (SR, JK, D, y T) y por su sincronismo (Síncrono y asíncrono). Los asíncronos no cuentan con una entrada de reloj o de sincronía. Los síncronos si cuentan con esta entrada, que en algunos casos la denotan CP. Existe también otra característica que diferencia unos flip-flops de otros, y es que estos cambian de estado ya sea por flanco o por nivel de voltaje, aunque estos últimos que cambian por nivel de voltaje les llaman latch. Ver la tabla (1.1-10).

- Asíncronos: No necesitan una señal de reloj para su funcionamiento y se activan por nivel de voltaje (latch). Ejemplo: D y RS.
- Síncronos: Utilizan una señal de reloj para su sincronía y se excitan por flancos (flip-flop). Ejemplo JK, T y D.

Entradas		Presente	Futuro
S	**R**	**Q(t)**	**Q(t+1)**
0	x	0	0
1	0	0	1
0	1	1	0
x	0	1	1

Entradas		Presente	Futuro
J	**K**	**Q(t)**	**Q(t+1)**
0	x	0	0
1	x	0	1
x	1	1	0
x	0	1	1

Entrada	Presente	Futuro
D	**Q(t)**	**Q(t+1)**
0	0	0
1	0	1
0	1	0
1	1	1

Entrada	Presente	Futuro
T	**Q(t)**	**Q(t+1)**
0	0	0
1	0	1
1	1	0
0	1	1

Tablas (1.1 - 10): Tablas de excitación de los flip-flops.

MEMORIAS

Las memorias son elementos compuestos por bancos de registros de memoria. Los registros son un conjunto de biestables, que pueden almacenar información (1 y 0). Las memorias son consideradas de alta escala de integración, ya que existen de diversas capacidades, superiores a los Mega bits. Básicamente existen estos tipos de memoria: las de lectura y las de acceso aleatorio. Las de lectura, son memorias que se utilizan para almacenar información y conservarla aún después de interrumpir el suministro eléctrico, como ejemplo la memoria ROM (read-only memory) o memoria de solo lectura. Las de acceso aleatorio son aquellas memorias que se utilizan para estar continuamente escribiendo y borrando datos en sus registros, pero este tipo de memoria pierde la información que contenía al interrumpir el

suministro eléctrico. Como ejemplo, la memoria RAM (random acces memory) o memoria de acceso aleatorio.

Para su manejo, se usa el bus de dirección, el bus de datos y el bus de control. El bus de dirección es quien le dice a la memoria cual es la localidad del registro que queremos que nos muestre a la salida, o sea, por el bus de datos y controlado por un bus de control, por lo regular el bit enable. Su representación en un ejemplo de una memoria de 1Kbyte, direccionado con un bus de diez líneas y un bus de datos de 8 líneas se muestra en la figura (1.1-5).

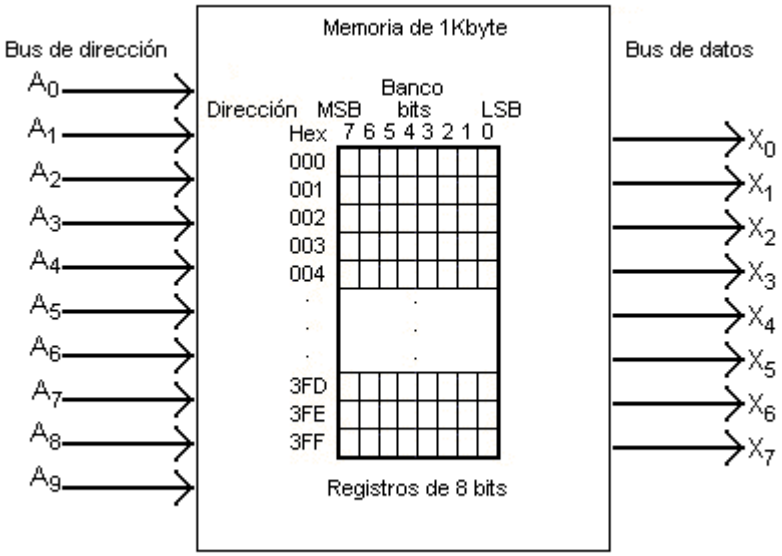


Figura (1.1 - 5) Memoria con un banco de 1Kbyte.

1.2. CARACTERÍSTICAS PRINCIPALES DE LOS SISTEMAS ANALÓGICOS Y DIGITALES

Un sistema digital es cualquier dispositivo destinado a la generación, transmisión, procesamiento o almacenamiento de señales digitales. Una señal digital corresponde a magnitudes físicas limitadas a tomar sólo unos determinados valores discretos. A diferencia de éstas, las señales son analógicas cuando las magnitudes de las mismas se representan mediante variables continuas, que pueden tomar cualquier valor en cada instante.

En el mundo real, las señales (tales como sonido y ondas de radio) se originan en forma analógica (su naturaleza es continua, no discreta). Los computadores digitales, por otro lado, manejan la información discontinuamente, como una serie de números binarios, por lo que se hace necesario como primera etapa en la mayoría de los sistemas transformar las señales analógicas (continuas) en digitales (discretas). Esta transformación la hacen los Conversores Analógico – Digital (ADC, en inglés).

Supóngase, pues, que la información que se desea enviar en un momento dado puede caracterizarse por una muestra de n posibles entradas, todas con la misma probabilidad de ocurrencia. Por ejemplo, esta muestra podría ser uno de 256 niveles de tensión igualmente probables. Para enviar esta muestra usando un sistema binario, primero se genera una palabra digital compuesta por m símbolos binarios. Por tanto, cada palabra binaria consta de $m = log_2 n$ dígitos binarios para representar una muestra de entre $n = 2^m$ posibilidades. De esta manera, m se llama número de bits (*binary digits,* dígitos binarios) necesarios para representar a uno de entre n posibles estados de entrada.

En un sistema binario, estos bits se representan con símbolos binarios (ejemplo, +1 y -1) que se generan a razón de r símbolos por segundo. La tasa de transmisor es $R = mr$ bits por segundo (bps). En el receptor la señal transmitida se adultera por la adición de ruido y, como resultado, el receptor cometerá algunos errores. Es razonable que la cantidad de errores disminuya si aumenta S/N (razón señal a ruido). Para el tipo de canal considerado, la capacidad está dada por la ley de Harley-Shannon.

$$C = Blog_2(1 + S/N) \text{ bps}$$

- C = capacidad del canal
- B =ancho de banda del canal [Hz]
- S/N= razón señal ruido
- Se cumple para $R \leq C$

Si se intenta enviar la información con mucha rapidez, es decir, $R > C$, los errores empiezan a aumentar aceleradamente y no tiene sentido diseñar un sistema para mejorar la situación. Por otra parte, si $R < C$, hay esperanza de mejorar a través de un buen diseño de sistema.

MUESTREO

La señal se podrá reconstruir si su espectro frecuencial no tiene componentes por encima de la frecuencia de Nyquist (que es dos veces la frecuencia natural del sistema).

Expresión matemática del teorema de muestreo.

$$T < \frac{1}{2B}$$

- T= intervalo de Nyquist [s]

ESPECTRO FRECUENCIAL.

El espectro frecuencial es un diagrama que muestra la amplitud en decibelios y la frecuencia de cada componente de la descomposición en serie de Fourier (lo que Fourier dijo es que cualquier señal se puede descomponer en una suma de senoidales y cosenoidales, una de las cuales una será la fundamental y el resto conformarán los *armónicos*, más un valor de continua). Otro detalle a tener en cuenta es que si la descomposición en serie es infinita y el espectro frecuencial tiene por tanto infinitas componentes, tendrá con total seguridad componentes por encima de Nyquist y por tanto no se podrá muestrear en condiciones, lo que nos fuerza a usar filtros (Bessel, Butterworth, Chebyshev etc.) que nos conviertan nuestra señal de banda ilimitada en limitada (con un número de componentes finitas en su espectro).

La serie trigonométrica de Fourier se puede representar por una sucesión de senos y cosenos.

$$f(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n cos\omega_0 t + \sum_{n=1}^{\infty} b_n sen\omega_0 t$$

- $1/2\,a_0$: Fundamental o componente de directa
- $\omega_0 = 2\pi f$; $f = 1/T$; $\omega_0 = 2\pi/T$: Frecuencia angular fundamental
- a_n; b_n: Amplitudes armónicas

TRANSFORMADA CONTINUA DE FOURIER

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t}\, dt$$

TRANSFORMADA INVERSA DE FOURIER

$$F(\omega) = \mathfrak{F}\{f(t)\} = \int_{-\infty}^{\infty} f(t) e^{-j\omega t}\, dt$$

En estricto rigor, el procesamiento digital de la señal se refiere al procesamiento electrónico de señales tales como sonido, radio y microondas usando técnicas matemáticas para realizar transformaciones o extraer información. En la práctica esta operación o transformación de la señal se hace en un hardware digital según unas reglas bien definidas las cuales son introducidas al hardware a través de un software específico que puede o no manejar lenguajes tanto de alto como de bajo nivel. Los dispositivos hardware adecuados para esto son los DSP (Digital Signal Processors, procesadores digitales de señal). Algunos conocidos son los DSPic de Microchip (que utiliza MPLAB como software de desarrollo integrado) o los TMS de Texas Instruments.

Uno de los beneficios principales del procesamiento digital es que las transformaciones de señales son más sencillas de realizar que con métodos analógicos. Una de las más importantes transformadas es la Transformada de Fourier discreta (DFT). Esta transformada convierte la señal del dominio del tiempo al dominio de la frecuencia. La DFT permite un análisis más sencillo y eficaz sobre la frecuencia, sobre todo en aplicaciones de eliminación de ruido y en otros tipos de filtrado.

TRANSFORMADA DISCRETA DE FOURIER (DTF).

El equivalente en tiempo y frecuencia discreta es la Transformada Discreta de Fourier (DFT)

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{2\pi j}{N} kn}$$

- N: Número de muestras en $x(n)$
- $x(n)$: Señal de prueba discreta (con índice n)
- $X[k]$: Espectro en función de la frecuencia discreta (con índice K)
- $e^{-jk\omega n/N}$: Fasor de sondeo discreto

Existe un algoritmo de cálculo de DFT muy eficiente que se conoce como FFT (fast fourier transform) y que es la base matemática de casi cualquier análisis de señal hoy en día. La transformada de Fourier rápida es un algoritmo eficiente para el cálculo numérico de la DTF. La versión más utilizada de la FFT calcula N componentes de frecuencia a partir de N muestras en el tiempo para $N = 2^r$, con r como cualquier entero positivo.

Otra herramienta importante es la integral de convolución, para la solución de problemas de análisis de circuitos y en general sistemas lineales invariantes e el tiempo, ya que este teorema de la convolución permite evaluar la respuesta de un sistema lineal a una excitación arbitraria en términos de su respuesta a un impulso unitario. En comunicaciones es muy útil para la modulación. Dadas dos funciones, $f_1(t)$ y $f_2(t)$ podemos formar la integral siguiente.

$$f(t) = \int_{-\infty}^{\infty} f_1(\tau) f_2(t-\tau) d\tau$$

Teorema de convolución en el tiempo

si: $f_1(t) \leftrightarrow F_1(\omega)$

$f_2(t) \leftrightarrow F_2(\omega)$

Entonces con esa condición:

$$\int_{-\infty}^{\infty} f_1(\tau) f_2(\tau) \leftrightarrow F_1(\omega) F_2(\omega)$$

Es decir por lo tanto: $f_1(t) \cdot f_2(t) \leftrightarrow F_1(\omega) F_2(\omega)$

El procesamiento digital de señales es por tanto una técnica que convierte señales de fuentes del mundo real (usualmente en forma analógica), en datos digitales que luego pueden ser analizados. Si en la naturaleza las señales son analógicas, ¿porqué realizar transformaciones a señales digitales para hacer el procesado? Este análisis es realizado en forma digital pues una vez que una señal ha sido reducida a valores numéricos discretos, sus componentes pueden ser aisladas, analizadas y reordenadas más fácilmente que en su primitiva forma analógica. Muchas son las ventajas de la 'digitalización':

- Se facilita su transmisión o almacenamiento.
- Es posible realizar mediante procesamiento digital acciones imposibles de obtener mediante el procesamiento analógico (por ejemplo, filtros con respuesta de frecuencia arbitraria).
- Es más cómodo de realizar y más barato de implementar que en el procesamiento analógico.
- Las señales digitales requieren usualmente menos ancho de banda y pueden ser comprimidas.

1.3. ELECTRÓNICA BÁSICA

1.3.1. SEÑALES ELÉCTRICAS

Las señales eléctricas son de tipo longitudinal, por que se trasladan en forma paralela con el tiempo, tienen amplitud, duración y forma.

Las señales eléctricas pueden ser generadas por una fuente eléctrica. Algunas señales no tienen forma eléctrica, por esto deben ser convertidas con ayuda de un transductor.

LA SEÑAL DE CORRIENTE DIRECTA

Esta señal le debe su nombre a su comportamiento, donde mantiene una amplitud constante con respecto del tiempo, esto quiere decir que tiene un periodo infinito. Comúnmente se abrevia DC, que en inglés es Direct Current. La figura (1.3-1) muestra la forma de esta señal.

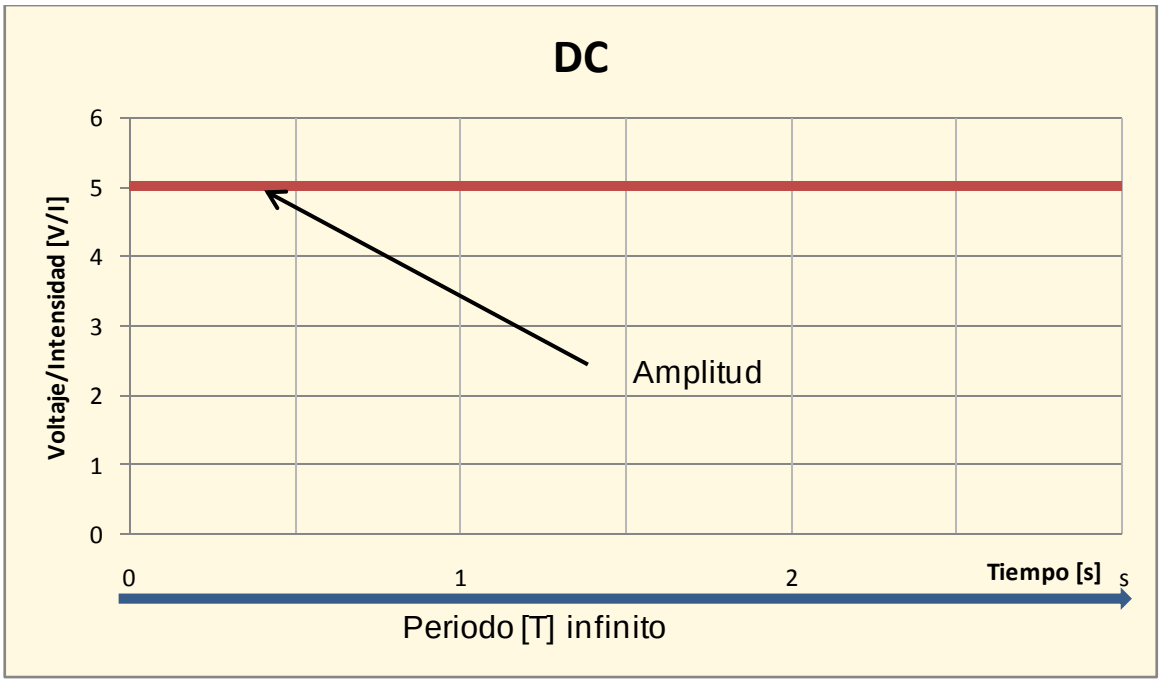


Figura (1.3 - 1) Señal de corriente directa.

Esta señal tiene las siguientes características:

a) Amplitud: Es la magnitud en Volts (V) o Amperes (I) que tiene la señal.
b) Duración: Es el tiempo que tarda en repetirse el ciclo. A esta característica se le nombra Periodo (T).
c) Frecuencia: Es el número de repeticiones que tiene la señal durante una unidad de tiempo. Es dual con respecto del periodo. A esta característica se le llama frecuencia (Hz).
d) Forma: Es una línea recta horizontal, y está arriba del cero porque es positiva (+), aunque también puede estar abajo del cero cuando es negativa (-).

e) Este tipo de señal se utiliza para polarizar (Alimentar) a las componentes electrónicas. Esta señal se puede simbolizar, para su representación en un circuito. Ver la figura (1.3-2).

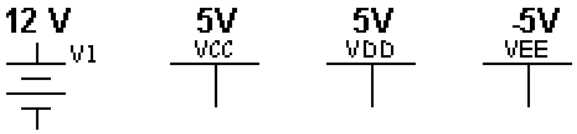


Figura (1.3 - 2) Símbolos de Corriente directa .

SEÑAL DIGITAL

Es similar a la de DC (Corriente Directa), pero por convenio se establece que existan dos niveles de tensión (Amplitud) distinto, esto para conservar la lógica de uno (1) y cero (0), que es la señal con la que trabajan los sistemas digitales, con esto se facilita su uso y almacenamiento. El ejemplo siguiente es el de una señal digital que utiliza una amplitud de 5V y 0V respectivamente.

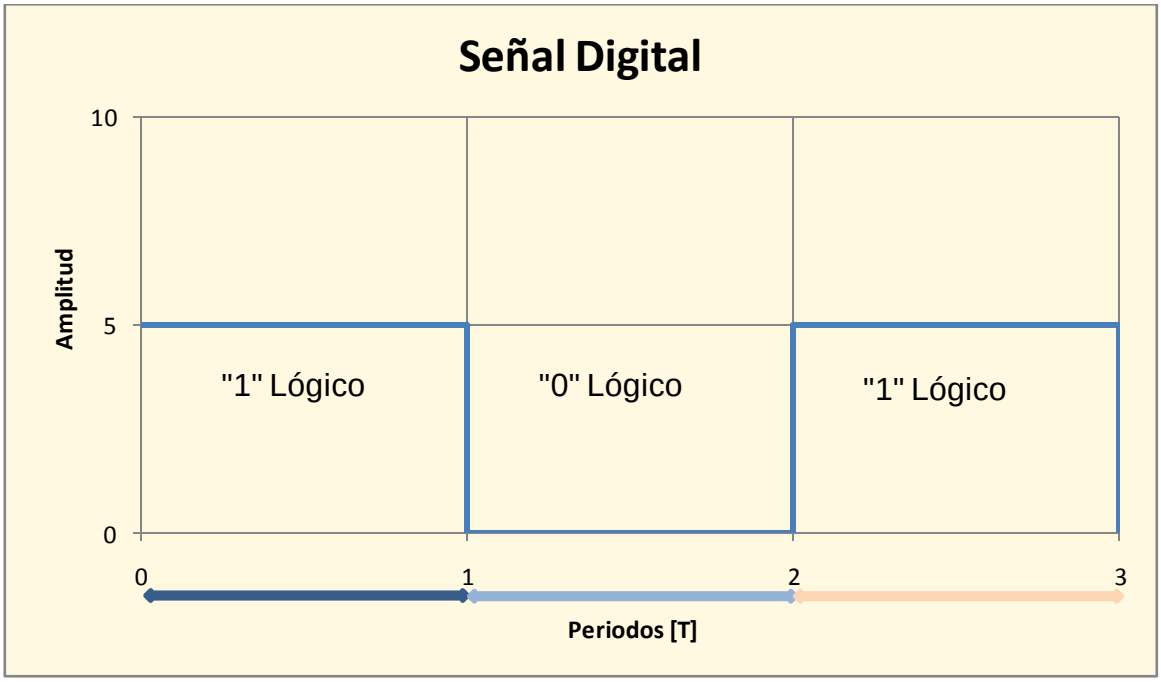


Figura (1.3 - 3) Señal digital.

a) Amplitud: Es la magnitud de Voltaje (V) o de Intensidad (I), que representa ya sea un cero lógico (0) o un uno lógico (1).
b) Forma: Es la de un tren de pulsos.
c) Duración: Es el tiempo que tardan los pulsos (bits), y la duración puede ser la misma o diferente.
d) El símbolo de su fuente es como en la figura (1.2-4).
e) Algunas de sus aplicaciones:

- Sincronizar componentes electrónicas digitales.
- Señal de entrada (*Vi*) de un circuito o equipo digital para que sea procesado.
- Señal de control de un circuito, para que actúe una carga (R_L); como un motor, lámpara, componente electrónica u otro circuito.

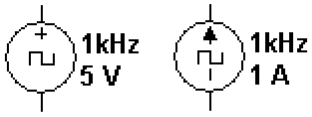


Figura (1.3 - 4) Símbolo de fuente de pulso cuadrado.

SEÑAL ANALÓGICA

La señal analógica es aquella que cuenta con puntos infinitos con respecto del tiempo, es decir, una línea continua, a diferencia de la digital sus puntos son finitos. En un análisis de una señal analógica se dice que es en tiempo continuo, mientras que el análisis de una señal digital se dice que es en tiempo discreto.

SEÑAL PERIÓDICA

Son aquellas señales que, luego de pasar por una serie de valores con una secuencia dada, vuelven a repetirse esos mismos valores con igual frecuencia, en forma cíclica e indefinida, aún para tiempos negativos. En la figura (1.3-5) aparecen unos ejemplos. Una *función periódica* se puede definir como una función para la cual $f(t) = f(t + T)$ para todo valor de t (tiempo).La constante T que satisface la relación se llama *período* de la función.

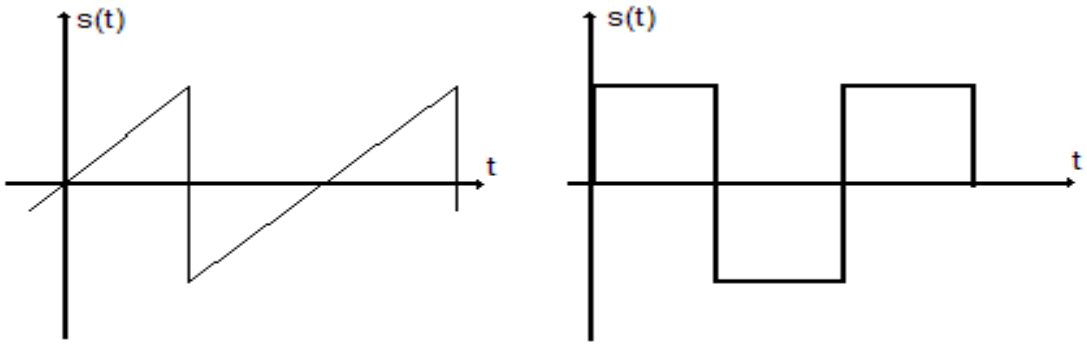


Figura (1.3 - 5) Diente de sierra y pulso cuadrado.

PERIODO

Periodo es el tiempo que transcurre para que un fenómeno u onda ocurra una sola vez. Debe notarse que no es necesario empezar a contabilizar el periodo (T) desde el momento en el cual la señal toma un valor nulo, sino que se puede empezar a partir de cualquier instante.

$$T = \frac{1}{f}$$

donde: $T = Periodo[s]$ $f = frecuencia[Hz]$

FRECUENCIA

Este término es empleado para indicar el número de veces que se repite en un segundo algún fenómeno. Por tanto, la frecuencia es el número de ciclos (oscilaciones) que una onda efectúa en un segundo; se mide en hercios (ciclos por segundo). Ver la figura (1.3 - 6).

$$f = \frac{1}{T}$$

donde: $f = frecuencia[Hz]$

$T = Periodo[s]$

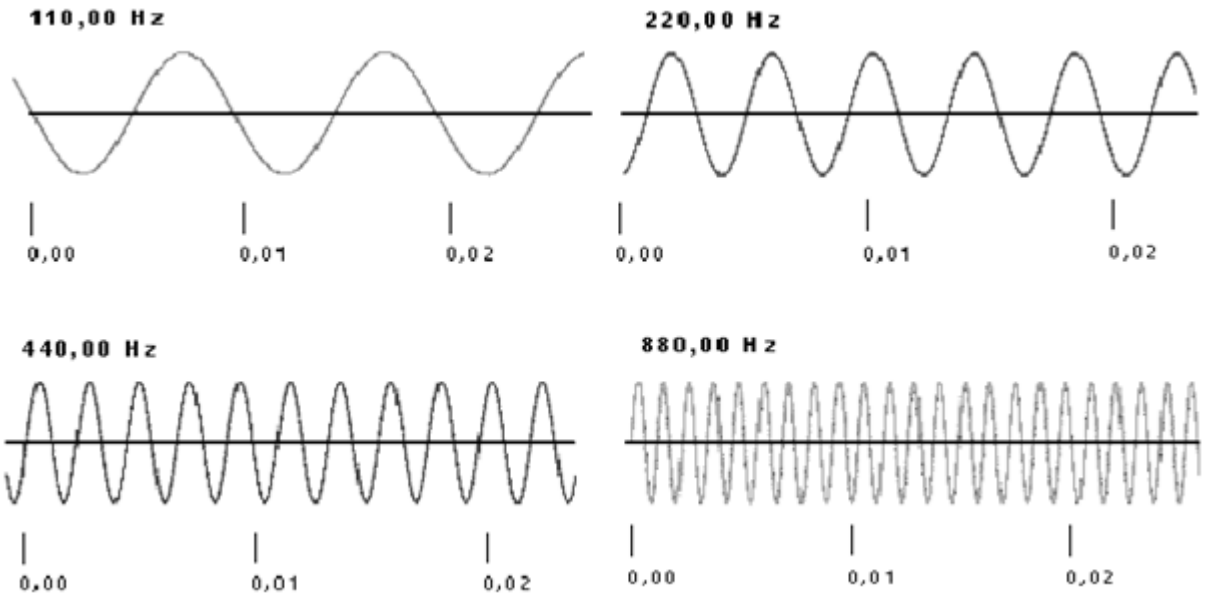


Figura (1.3 - 6) Graficas de frecuencias.

AMPLITUD

Amplitud es el valor tomado desde el origen, hasta la cresta, como en figura (1.3 - 7).

VALOR MÁXIMO

El valor máximo se refiere al valor mayor, considerando solo el medio ciclo positivo de la onda senoidal. En pocas palabras es el valor que tiene la cresta de la onda. Véase la figura (1.3 - 7).

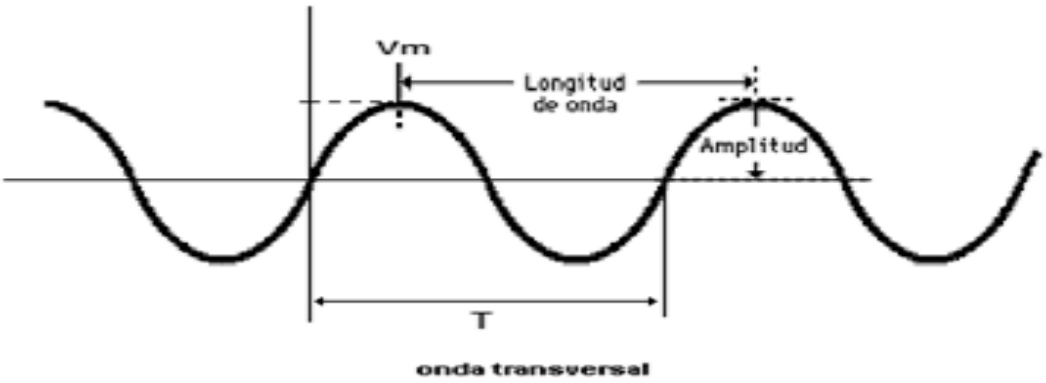


Figura (1.3 - 7) Función senoidal, señal analógica periódica.

VALOR EFICAZ

Es el valor que obtendríamos al integrar la parte positiva de una señal senoidal, por ejemplificar una señal; es un promedio del valor máximo y demás valores en la señal. También se puede definir como el valor que debería tener una señal constante para disipar, en un intervalo de tiempo igual a un periodo o, igual cantidad de energía que la señal en cuestión. La relación existente es que el valor eficaz es un valor constante que trata de mantenerse, y se calcula tomando el valor máximo (cresta o pico). Matemáticamente se define como:

$$V_{ef} = \frac{V_{máx}}{\sqrt{2}}$$

donde:

Vmáx = Valor máximo

Vef = Valor eficaz

1.3.2. CONCEPTOS ELECTRICIDAD

ELEMENTOS ACTIVOS

Un elemento activo es capaz de suministrar o generar energía. Los elementos activos son fuentes potenciales de energía. Como ejemplos de elementos activos están las baterías y los generadores. Esta expresión indica que la energía puede ser transferida, ya sea imponiendo la tensión, o bien fijando la corriente. En consecuencia habrá dos variables posibles, aquellos que imponen la tensión y aquellos que imponen la corriente.

ELEMENTOS PASIVOS

Un elemento pasivo absorbe energía. Se dice que es un elemento pasivo si la energía total que se le suministre del resto del circuito es siempre no negativa (cero a positiva). Aquellos elementos los cuales consumen, almacenan, intercambian energía ó la ceden al medio, se les llama elementos pasivos. Observando detalladamente los procesos energéticos que se desarrollan en los tramos pasivos, existen procesos irreversibles, reversibles asociados al campo eléctrico, o reversibles asociados al campo magnético.

Los procesos irreversibles son fenómenos de intercambio energético que, corresponden a la transformación de energía electromagnética a otras formas de energía (térmica, mecánica, lumínica, química, acústica, etc.).

Los procesos reversibles están asociados al campo eléctrico y su almacenamiento de energía en forma potencial. También al campo magnético y almacenan energía en forma electrocinética.

SIMBOLOGÍA

Para los diagramas de circuitos, es importante conocer estas prácticas representaciones de algunos elementos. Ver la figura (1.3-8).

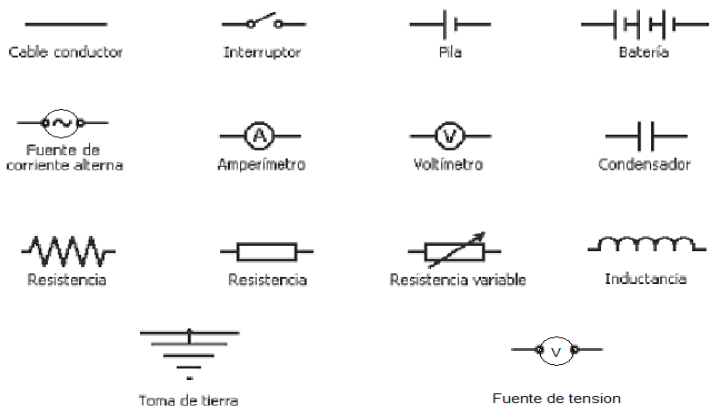


Figura (1.3 - 8) Símbolos de elementos pasivos y activos.

TENSIÓN Y CORRIENTE

La unidad de diferencia de potencial se llama volt, y es la diferencia de potencial entre dos puntos, cuando la unidad de carga, o sea un coulomb, al llevarse de un punto a otro, requiere de un trabajo de 1 joule. Debido a esta diferencia, la corriente de electrones fluye a través del conductor, del más negativo al menos negativo o más positivo. Se supone que la fuente de voltaje proporciona energía con un voltaje terminal específico.

Se llama flujo de corriente al movimiento continuo de los electrones libres. Así, cuanto mayor sea la corriente eléctrica, mayor será el número de electrones que pasan por un punto de referencia, en un intervalo de tiempo dado. La unidad de intensidad de corriente o flujo de corriente es el ampere, que se representa con una "A" y es igual al flujo de una carga de un coulomb por segundo, en un punto dado de un conductor.

$$1 ampere = \frac{1 coulomb}{1 segundo}$$

RESISTENCIA

El paso de electrones a través de un material no se logra sin que estos sufran choques con otras partículas atómicas. Es más, estas coaliciones no son elásticas y se pierde energía en cada una de ellas. La perdida por unidad de carga se interpreta como caída en el potencial a través del material. La cantidad de energía que pierden los electrones se relaciona con las propiedades físicas de una sustancia en particular.

LEY DE OHM

El flujo de corriente en ampere que circula por un circuito eléctrico cerrado, es directamente proporcional a la tensión o voltaje aplicado, e inversamente proporcional a la resistencia en ohm de la carga que tiene conectada.

La ley de O*hm* se puede escribir como:

$$v = Ri = R\frac{dq}{dt}$$ donde:

R=resistencia [Ω] ohm

i =corriente	[A] Ampere
q=carga	[c] coulomb
t=tiempo	[s] segundo
v=tensión	[v] volt

POLARIDAD

La polaridad es un concepto que es descubierto desde que nace el magnetismo, con la observación de la magnetita que atrae el hierro. En la convención de Franklin se decidió que la carga de la ebonita frotada con seda es negativa (-) y la carga de la seda positiva (+) (déficit de electrones). Pero en el análisis de un circuito, nos permite determinar el sentido de la corriente en una fuente de voltaje, y también nos permite determinar si hay consumo o suministro de voltaje, al determinar un sentido de corriente, aunque sea arbitrario. La polaridad de un elemento se determina colocando un signo positivo y uno negativo según, esté conectada la fuente.

SEMICONDUCTOR

Un semiconductor puede cambiar de estado, puede ser un conductor o un aislante, dependiendo su polarización. El ejemplo más sencillo es el diodo rectificador, que polarizado en directa puede ser un conductor, y polarizado en inversa actúa como un circuito abierto. La figura (1.3 - 9) muestra el voltaje [V_D], cuando está conduciendo en directa, y tiene una corriente distinta de cero.

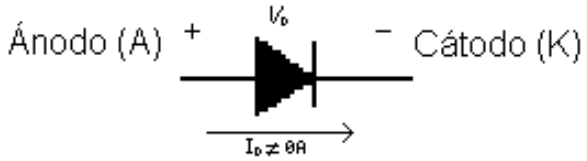


Figura (1.3 - 9).Símbolo del Diodo.

Los elementos semiconductores básicos son los Diodos, transistores, tiristores; Estos elementos se caracterizan por ser de estado sólido, lo que los hace elementos de conmutación muy rápida, ya que no tienen piezas móviles para su funcionamiento.

Los semiconductores con base de elemento silicio, germanio, se han vuelto muy populares, gracias a la diversidad en sus aplicaciones, además de ser muy económicos.

POLARIZACIÓN DIRECTA

Es cuando la polaridad del voltaje aplicado es igual al que existe en el interior de la componente y en la terminal conectada.

POLARIZACIÓN INVERSA

Es cuando la polaridad del voltaje es diferente a la polaridad que existe en la terminal seleccionada de la componente.

1.3.3. GALGAS EXTENSIOMÉTRICAS

Para este tema se debe de aclarar que no se hace un estudio profundo, ya que este puede ser un tema muy amplio, incluso para otro tema de tesis. Pero se toman los conceptos más importantes que aportan un mejor entendimiento en el desarrollo del proyecto.

El uso de las galgas extensiométricas está muy difundido como método instrumental para la:

- Medición de deformación
- Medición de esfuerzos
- Medición de fuerza/peso

Una Galga extensiométrica o String Gage es básicamente una resistencia eléctrica, ya que lo que se mide en ella es la variación de la resistencia de la galga cuando esta sufre una determinada deformación. Es decir, existe una variación directa entre la variación que sufre la resistencia y la variación de la deformación en la galga.

Al pegar una galga en la superficie en la cual se quiere analizar su deformación, se parte de la hipótesis de que el sensor experimenta la misma deformación que el material. El sensor consta de una base muy delgada no conductora, en la cual hay adherido un hilo metálico muy fino, de esta forma, la mayor parte de su longitud está distribuida paralelamente a una dirección determinada, como se puede ver en la figura (1.3 – 10).

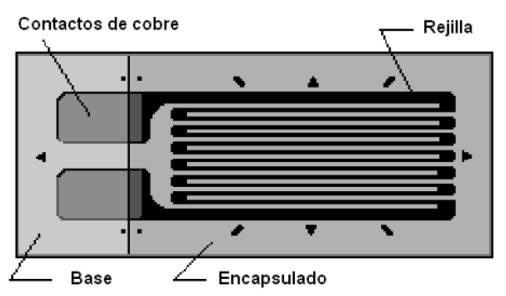


Figura (1.3 – 10) Galga extensiométrica.

El fino hilo, si conduce, es quien nos proporciona la resistencia que queremos medir, la cual varia con la deformación, esta viene dada por la ecuación:

$$R = \rho \frac{L}{A}$$

Donde: R = Resistencia Eléctrica
ρ = Resistividad
L = Longitud
A = Sección transversal

Vemos en esta ecuación que la resistencia medida es directamente proporcional a la longitud, es decir, la resistencia es mayor al alargar el hilo, lo cual se consigue cuando el material se deforma. Al estar la galga adherida al material, provoca esta variación de longitud, y con ello que la resistencia varíe. Ver la figura (1.3 - 11).

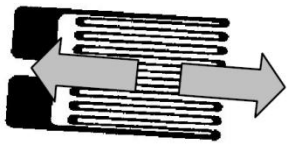


Figura (1.3 – 11) Deformación sobre la galga extensiométrica.

Existen diferentes tipos de galgas que son utilizadas conforme a sus características particulares, que son aplicables a diferentes sistemas. Estas son:

- De hilo metálico
- Laminares metálicas
- De metal depositado
- Semiconductores
- Tipo rosetas

Los materiales utilizados para su construcción son los siguientes:

- Constantan (Níquel-Cobre)
- Chromel (Níquel-Cromo)
- Aleaciones (Hierro-Cromo-Aluminio)
- Semiconductores (Silicio)

CARACTERÍSTICAS DE LAS GALGAS

Las características y propiedades que se deben tener en cuenta son:

1. *Anchura y Longitud:* Estos dos parámetros hay que tenerlos en cuenta cuando escogemos el sensor para adherirlo al material, por lo tanto se debe elegir la dimensión que más se adecue al tamaño del material.
2. *Peso:* El peso de una galga suele ser del orden de gramos y, en aplicaciones donde se necesita mucha precisión, puede influir a la medida de la deformación realizada.

1. *Tensión Obtenida:* Es el rango de variación de longitud de la galga, cuando esta se somete a una deformación. Este rango viene expresado en un tanto por ciento respecto a la longitud de la galga.

2. *Influencia de la Temperatura:* La temperatura puede afectar al funcionamiento de la galga, si esta varía durante una medida con bandas extensiométricas, la deformación real puede desviarse de la deformación medida. Para ello el fabricante de la galga proporciona dos curvas, para poder corregir los efectos dados por la variación de la temperatura.

3. *Resistencia de la Galga.* Es la resistencia de referencia que se da cuando el sensor no sufre ninguna deformación, es decir, el valor nominal de resistencia, suele venir acompañada por un porcentaje que indica su tolerancia.

4. *Factor de Galga:* Factor de galga es una constante K característica de cada galga. Este factor es común de muchos parámetros, pero especialmente de la aleación empleada en la fabricación. Viene acompañada de su tolerancia.

5. *Sensibilidad Transversal:* Las galgas están diseñadas para trabajar en una dirección determinada, sin embargo, si se producen deformaciones transversales, se puede dar una variación de resistencia. El fabricante proporciona este valor en forma de porcentaje, suele ser este menor del 1%.

6. *Material de Lámina:* Esta característica nos define el material del que está hecho el hilo conductor o el material semiconductor.

7. *Material de la Base:* Esta característica nos define el material del que está hecha la base no conductora de la galga.

8. *Linealidad, Histéresis, y Deriva:* La linealidad, histéresis y deriva dependen de diversos factores como son el nivel de deformaciones alcanzado, el material de soporte de la banda y la calidad y los materiales del pegado.

9. *Disipación de Calor:* Otro aspecto interesante al utilizar bandas extensiométricas es la disipación de calor, puesto que una banda extensiométrica es un elemento resistivo, formara parte de un circuito eléctrico y por tanto pasará una corriente eléctrica por la banda. Por lo tanto, hay que prestar especial cuidado en cuanto a que la potencia que consuma la banda, debido al paso de la corriente eléctrica, y que disipa en forma de calor, sea menor que la potencia que la banda es capaz de transmitir al material sobre el que se ha pegado. De esta forma se evita el sobrecalentamiento de la banda, que podría dar lugar a medidas erróneas o incluso a quemar la propia galga.

10. *Estabilidad:* Cuando se hacen medidas que duran tiempos largos o se utilizan bandas montadas en piezas con mucha antelación, las condiciones ambientales pueden degradar las propiedades de la banda, haciendo que el comportamiento de la galga se aleje de lo esperado o lleguen a deteriorarse.

11. *Comportamiento a la fatiga:* Como todos los materiales, las bandas tienen una vida limitada por la fatiga. Las bandas estándar son capaces de aguantar unos 105 ciclos. Cuando se requiere una mayor durabilidad en fatiga existen bandas especiales para tales fines.

MEDICIONES CON GALGAS EXTENSIOMÉTRICAS

Las galgas extensiométricas son capaces de medir deformaciones inapreciables a simple vista, esto es, una pequeña variación de resistencia, por lo cual el circuito debe ser muy sensible. Para el acondicionamiento de las galgas, el circuito utilizado es el puente de Wheatstone, que debido a sus características lo convierten en el circuito ideal para estos casos. Existen tres tipos de montajes básicos que son con una, dos ó cuatro galgas.

PUENTE WHEATSTONE

En la Figura (1.3 – 12) vemos a Rx, es la resistencia cuyo valor queremos determinar, R1, R2 y R3 son resistencias de valores conocidos, además, la resistencia R2 es ajustable. Si la relación de las dos resistencias del brazo conocido (R1/R2) es igual a la relación de las dos del brazo desconocido (Rx/R3), el voltaje entre los dos puntos medios será nulo y por tanto no circulará corriente alguna entre esos dos puntos C y B.

Para efectuar la medida, lo que se hace es variar la resistencia R2 hasta alcanzar el punto de equilibrio. La detección de corriente nula se puede hacer con gran precisión mediante un galvanómetro entre los puntos C y B.

La dirección de la corriente, en caso de desequilibrio, indica si R2 es demasiado alta o demasiado baja.

Cuando el puente está construido de forma que R3 es igual a R2, Rx es igual a R1 en condición de equilibrio (corriente nula por el galvanómetro).

En condición de equilibrio siempre se cumple que:

$$Rx = \frac{R1 \times R3}{R2}$$

Si los valores de R1, R2 y R3 se conocen con mucha precisión, el valor de Rx puede ser determinado igualmente con precisión. Pequeños cambios en el valor de Rx romperán el equilibrio y serán claramente detectados por la indicación del galvanómetro.

De forma alternativa, si los valores de R1, R2 y R3 son conocidos y R2 no es ajustable, la corriente que fluye a través del galvanómetro puede ser utilizada para calcular el valor de Rx, siendo este procedimiento más rápido que el ajustar a cero la corriente a través del medidor.

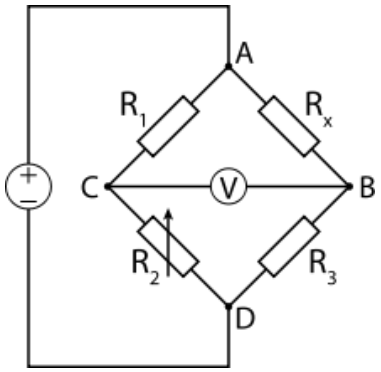


FIGURA (1.3 – 12) Puente Wheatstone.

CONFIGURACIONES DEL PUENTE WHEATSTONE

Dependiendo del tipo y complejidad de la medición que se esté efectuando, existen configuraciones diferentes del puente Wheatstone.

CONFIGURACIÓN DE UN CUARTO DE PUENTE

Cuando se utiliza solo una galga y resistores complementando el puente, la configuración se denomina cuarto de puente. En esta configuración los cables conductores que conectan la galga con el puente deben ser del mismo calibre y, si se puede, deben de trenzarse para evitar diferencias de temperatura entre ellos. La figura (1.3 – 13) muestra la configuración.

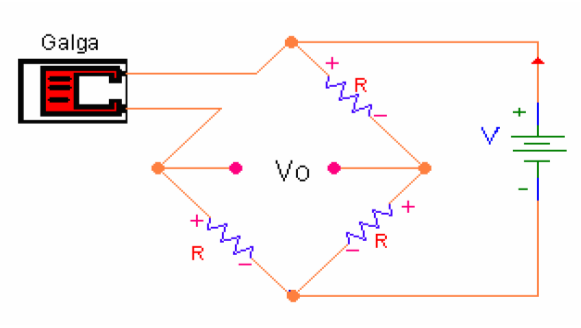


Figura (1.3 – 13) Configuración de un cuarto de puente Wheatstone.

CONFIGURACIÓN DE MEDIO PUENTE

Esta configuración posee un voltaje que es lineal, y dobla aproximadamente la salida del circuito de cuarto de puente, aquí ambas galgas están activas, solo que la segunda galga se coloca transversalmente a la primera galga. Para que esta configuración funcione adecuadamente, la resistencia de una galga debe incrementarse mientras que la otra debe disminuir. La figura (1.3 – 14) muestra esta configuración.

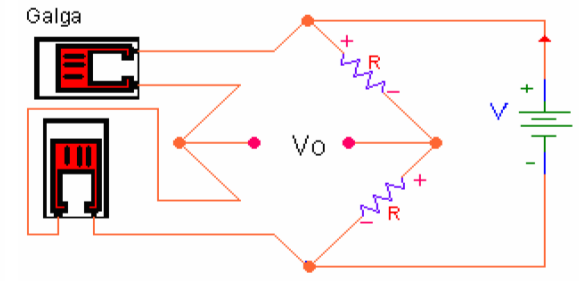


Figura (1.3 – 14) Configuración de medio puente de Wheatstone.

CONFIGURACIÓN DE PUENTE COMPLETO

En esta configuración los cuatro brazos del puente son utilizados, y la sensibilidad del puente aumenta considerablemente, siendo el voltaje de salida cuatro veces el de una configuración de cuarto de puente. Esta configuración se usa cuando el punto donde se efectúa la medición esta retirado de los instrumentos de medición, y también cuando las condiciones ambientales son sumamente cambiantes. Su uso principal es en transductores donde se requiere leer unidades diferentes a la deformación, tales como: presión, carga, etc. La figura (1.3 – 15) muestra esta configuración.

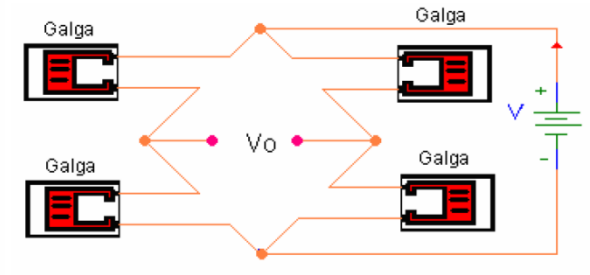


Figura (1.3 – 15) Configuración de puente completo de Wheatstone.

Las galgas no poseen polaridad, aunque, dependiendo de cuál de las tres configuraciones anteriores se emplee, existe un gran número de conexiones diferentes que se pueden tener para realizar el hardware de medición.

1.4. MICROCONTROLADORES

Los Microcontroladores son circuitos de alta escala de integración, pero se diferencian de otros circuitos especialmente porque se trata de una computadora en un solo circuito. Anteriormente, los circuitos integrados tenían una función específica (circuito a la medida), significa que solo realizaban una operación o tarea para la cual eran diseñados, de lo que se deriva que si se quería realizar alguna otra tarea, se debía diseñar un nuevo circuito, que cumpliera con las nuevas necesidades. Esto también implica un gasto, no solo en el tiempo para un nuevo diseño, también implica que para tareas más complicadas de control se requería un amplio diseño, una estructura robusta, poco flexible que, en caso de un error en el diseño, será muy complicado el depurarlo, corregir el error o cambiar piezas. El microcontrolador tiene esa flexibilidad que le hace falta a un circuito a la medida. El microcontrolador puede ser reprogramado para realizar tareas distintas (una sola función a la vez). Por eso se considera al microcontrolador como de propósito general, y a un circuito a la medida como de propósito específico. Cabe aclarar que cuando se programa un microcontrolador con la intención que realice una tarea, este se convierte en un circuito de propósito específico.

En definición, un microcontrolador es un circuito integrado que internamente, al igual que una computadora, tiene un microprocesador (CPU) o unidad central de proceso, memoria de programa, memoria RAM, oscilador, puertos etc. Es de propósito general y se puede programar para que realice alguna función de control.

Existen diversos Microcontroladores en el mercado, pero en los últimos años se han vuelto muy populares los Microcontroladores PIC *(Peripheral Interface Controller)* de la empresa *Microchip Technology Incorporated.* Esto gracias a sus diversas características, accesibilidad, bajo precio, diversos tamaños, calidad, bajo consumo, modelos, y en especial, la abundante información expuesta en Internet; los hacen una de las mejores propuestas para la solución de tareas de control.

En este trabajo se hace uso de un microcontrolador en especial, el PIC18F4550, entonces describiremos las características generales; contiene muchos módulos que no usaremos en este proyecto, pero se hará mención para describir el microcontrolador.

1.4.1. ARQUITECTURA INTERNA

Es necesario discernir algunos conceptos, que se aplican a casi todos los Microcontroladores PIC. Son muy importantes ya que facilitan el entendimiento de su funcionamiento.

MICROCONTROLADOR Y MICROPROCESADOR

Un microcontrolador es una microcomputadora. Para ser considerado microcomputadora, debe contar un mínimo de recursos.

Los requerimientos son: Un microprocesador (CPU [Unidad Central de Proceso]), memoria, oscilador, periféricos y puertos. Esto significa que un microcontrolador, en un solo circuito integrado, contiene una estructura con estos elementos y más. Esto implica que un diseño, basado en un microcontrolador, es menos robusto que el de un microprocesador, ya que al microprocesador se le tienen que agregar más recursos, como memorias, módulos de entrada y salida, etc. Además esto hace al sistema del microprocesador más costoso y poco fiable. Resumiendo, un microcontrolador tiene muchas ventajas, como tamaño reducido, costo bajo, fiable, y es bastante flexible, ya que cuenta con bastantes recursos para sustentar diversos tipos de tareas, a pesar de ser un elemento cerrado (un solo circuito Integrado).

ARQUITECTURA HARVARD

La arquitectura es la disposición en la estructura del microcontrolador. En esencia, tiene los requerimientos mínimos de una computadora, y se caracteriza por tener memorias independientes. Significa que cada memoria cuenta con su propio bus de datos, de dirección y de control, lo que implica que sea una ventaja, ya que puede tener ambas en funcionamiento, haciendo al sistema más veloz y eficiente. Ver la figura (1.4-1).

Existe otra arquitectura, más tradicional (arquitectura Von Neumann), en la cual se comparten las líneas de dirección, de datos y de control. En donde la ventaja es un diseño más simple y un costo más bajo. Ver la figura (1.4-2).

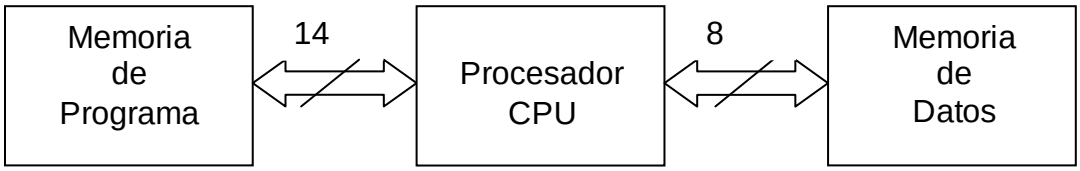


Figura (1.4 - 1): Arquitectura Harvard.

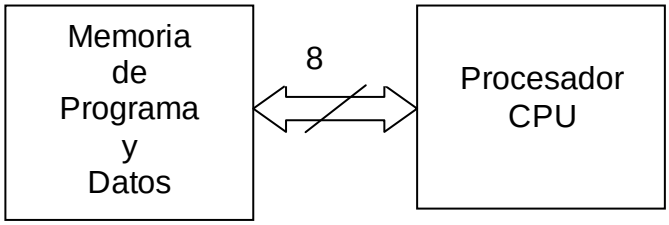


Figura (1.4 - 2): Arquitectura Von Neuman.

PROCESADOR SEGMENTADO

Este sistema va acompañado de una arquitectura Harvard, se caracteriza por hacer dos operaciones simultáneamente; búsqueda de una nueva instrucción y ejecución de la última instrucción. En esta forma se puede optimizar tiempo, ejecutando una instrucción por ciclo máquina. Ver la tabla (1.4-1).

Código	1erCM	2doCM	3erCM	4toCM	5toCM
Instrucción1	Búsqueda 1				
Instrucción2		Búsqueda 2 Ejecuta 1			
Instrucción3			Búsqueda 3 Ejecuta 2		
Instrucción4				Búsqueda 4 Ejecuta 3	
					Ejecuta 4

Tabla (1.4 - 1) Segmentación ó Pipe Line.

CICLO MÁQUINA

Otra característica del microcontrolador es que cada instrucción, se ejecuta en un ciclo máquina que, necesita 4 pulsos de reloj.

Esto nos lleva a la siguiente conclusión:

El tiempo que tarda el microcontrolador en ejecutar una sola instrucción esta dado por la siguiente fórmula:

$$Tiempo\ [s] = 4\left(\frac{1}{f}\right)$$

Donde:

- $[s]$ = segundos
- f = frecuencia $[Hz]$
- 4 = Número de pulsos de reloj

PROCESADOR RISC

Esto se refiere a la cantidad de instrucciones que soporta el procesador, y se clasifican a continuación:

RISC (*Reduced Instruction Set Computer*). Es el Juego de Instrucciones reducidas de Computadora. Son instrucciones Simples o muy elementales, se ejecutan regularmente en un ciclo máquina. Los Microcontroladores PIC cuentan con esta característica, y nuestro modelo PIC18F4550 tiene teste juego de instrucciones.

CISC (Complex Instruction Set Computer). Es el juego de Instrucciones complejas de computadora. Este tipo de juego de instrucciones suele ser muy amplio con más de 200 instrucciones, muchas de ellas son muy complejas, potentes y suelen requerir muchos ciclos de reloj.

SISC (Specific Instruction Set Computer). Juego de instrucciones específicas de computadora. Se refiere a instrucciones para tareas específicas, como procesamiento de audio o video, etc.

ARQUITECTURA ORTOGONAL

En este tipo de arquitectura, tras la ejecución de una instrucción, se puede llevar a un destino diferente, ya se al registro de trabajo, o alguna dirección de la memoria RAM (Memoria de acceso aleatorio); con esto se optimiza el tiempo y se reduce el número de instrucciones necesarias en la ejecución del programa. Ver la figura (1.4 - 3).

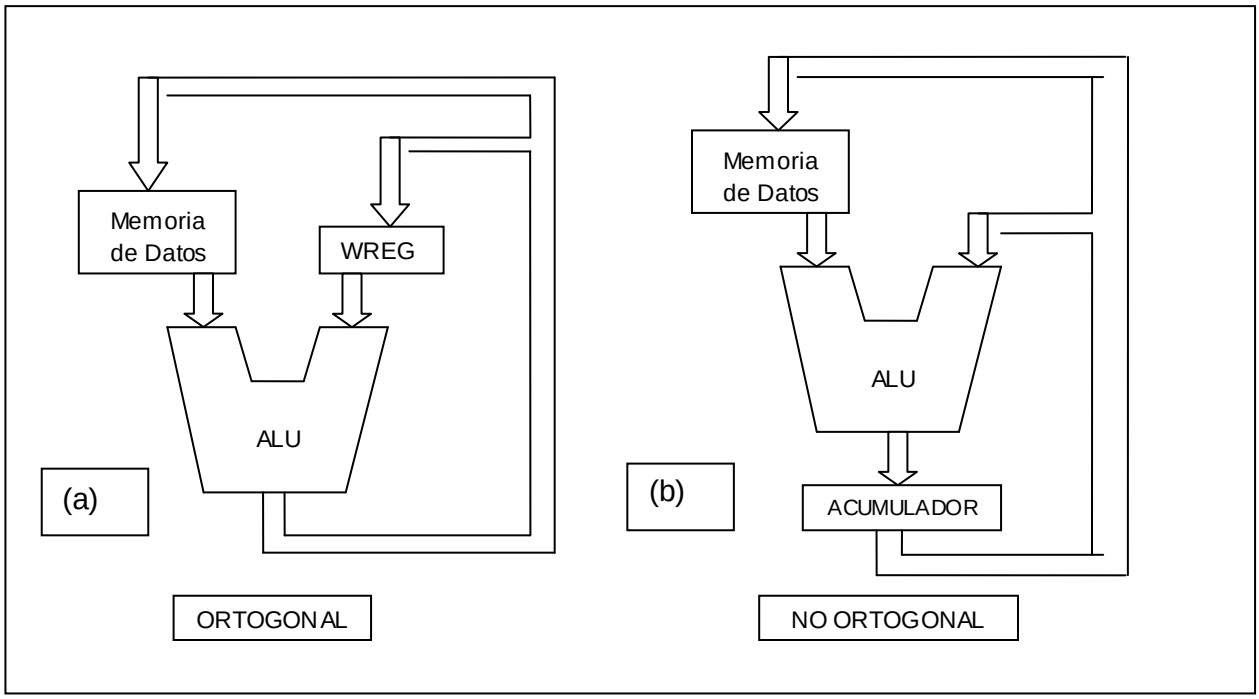


Figura (1.4 - 3): (a) arquitectura del PIC y (b) arquitectura tradicional.

Anteriormente, el resultado de una operación se llevaba directamente al registro de trabajo, llamado acumulador, y después se requería otra instrucción para cargar ese resultado en otro registro. Ahora con esta ventaja se reduce el código de un programa.

1.4.2. GAMAS DE LOS PIC

Para el diseño de un proyecto se tiene que pensar en el microcontrolador que mejor satisfaga las necesidades de dicho proyecto. Estos dispositivos se clasifican como familia o gama enana, baja, media y alta:

GAMA ENANA PIC12CXXX

Se trata de un grupo de PIC de reciente aparición que ha acaparado la atención del mercado. Su principal característica es su reducido tamaño, al disponer todos sus componentes de 8 patitas. Se alimentan con un voltaje de corriente continua comprendido entre 2,5V y 5,5V, y consumen menos de 2mA cuando trabajan a 5V y 4MHz. El formato de sus instrucciones puede ser de 12 o de 14 bits y su repertorio es de 33 o 35 instrucciones, respectivamente.

Aunque los PIC enanos sólo tienen 8 patitas, pueden destinar hasta 6 como líneas de E/S (Entrada/Salida) para los periféricos, porque disponen de un oscilador interno R-C (Resistencia-Capacitancia).

En la Tabla (1.4 -2) se presentan las principales características de los modelos de esta subfamilia, que el fabricante tiene la intención de potenciar en un futuro próximo. Los modelos 12C5xx pertenecen a la gama baja, siendo el tamaño de las instrucciones de 12 bits, mientras que los 12C6xx son de la gama media y sus instrucciones tienen 14 bits.

Los modelos 12F6xx poseen memoria Flash para el programa y EEPROM para los datos.

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS	FRECUENCIA MÁXIMA	LINEAS I/O	TEMPORIZADORES	PINES
PIC12C508	512x12	25x8	4 MHz	6	TMR0 + WDT	8
PIC12C509	1024x12	41x8	4MHz	6	TMR0 + WDT	8
PIC12C670	512x14	80x8	4MHz	6	TMR0 + WDT	8
PIC12C671	1024x14	128x8	4MHz	6	TMR0 + WDT	8
PIC12C672	2048x14	128x8	4MHz	6	TMR0 + WDT	8

Tabla (1.4 - 2) Algunos modelos de la gama enana.

GAMA BAJA PIC16C5X

Se trata de una serie de PIC de recursos limitados, pero con una de la mejores relaciones costo/prestaciones. Sus versiones están encapsuladas con 18 y 28 pines, y pueden alimentarse a partir de una tensión de 2,5 V, lo que les hace ideales en las aplicaciones que funcionan con pilas, teniendo en cuenta su bajo consumo (menos de 2mA a 5V y 4MHz). Tienen un repertorio de 33 instrucciones cuyo formato consta de 12 bits. No admiten ningún tipo de interrupción y la Pila sólo dispone de dos niveles. Ver la tabla (1.4 - 3).

Al igual que todos los miembros de la familia PIC16/17, los componentes de la gama baja se caracterizan por poseer los siguientes recursos:

- Sistema “POR” (Power On Reset)
- Perro guardián “WDT” (Watchdog)
- Código de protección
- Líneas de E/S de alta corriente
- Modo de reposo “sleep” (Bajo consumo

MODELO	MEMORIA PROGRAMA EPROM	ROM	MEMORIA DATOS	FRECUENCIA MÁXIMA	LINEAS I/O	TEMPORIZADORES	PINES
PIC16C52	384	no	25x8	4MHz	4	TMR + WDT	18
PIC16C54	512	no	25x8	20MHz	12	TMR + WDT	18
PIC16CRS4A	no	512	25	20MHz	12	TMR + WDT	18
PIC16C57	2K	no	72	20MHz	20	TMR + WDT	28
PIC16CRS8A	no	2K	73	20MHz	12	TMR + WDT	18

TABLA (1.4 - 3) Algunos modelos de la gama baja.

GAMA MEDIA PIC16”X”XXX

Es la gama más variada y completa de los PIC. Abarca modelos con encapsulado desde 18 patitas hasta 68, cubriendo varias opciones que integran abundantes periféricos. Dentro de esta gama se halla el «fabuloso PIC16X84» y sus variantes.

En esta gama sus componentes añaden nuevas prestaciones a las que poseían los de la gama baja, haciéndoles más adecuados en las aplicaciones complejas. Admiten interrupciones, poseen comparadores de magnitudes analógicas, convertidores A/D (analógico-digital), puertos serie y diversos temporizadores.

El repertorio de instrucciones es de 35, de 14 bits cada una y compatible con el de la gama baja. Sus distintos modelos contienen todos los recursos que se precisan en las aplicaciones de los microcontroladores de 8 bits. También dispone de interrupciones y una Pila de 8 niveles que permite el anidamiento de subrutinas. En la Tabla (1.4 – 4) se presentan las principales características de los modelos de esta familia.

Encuadrado en la gama media también se halla la versión PIC14C000, que soporta el diseño de controladores inteligentes para cargadores de baterías, pilas pequeñas, fuentes de alimentación ininterrumpibles y cualquier sistema de adquisición y procesamiento de señales que requiera gestión de la energía de alimentación. Los PIC 14C000 admiten cualquier tecnología de las baterías, como Li-Ion, NiMH, NiCd, Ph y Zinc.

El temporizador TMR1 que hay en esta gama tiene un circuito oscilador que puede trabajar asíncronamente, y que puede incrementarse aunque el microcontrolador se halle en el modo de reposo ("sleep"), posibilitando la implementación de un reloj en tiempo real.

Las líneas de E/S (entrada-salida) presentan una carga "pull-up" activada por software.

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS RAM | EEPROM	REGISTROS ESPECÍFICOS	INTERRUP-CIONES	LINEAS I/O	TEMPORI-ZADORES	PINES
PIC16F84	1Kx14FLASH	68 | 64	11	4	13	TMR0+WDT	18
PIC16F83	512x14FLASH	36 | 64	11	4	13	TMR0+WDT	18
PIC16C84	1Kx14EEPROM	36 | 64	11	4	13	TMR0+WDT	18
PIC16CR84	1Kx14 ROM	68 | 64	11	4	13	TMR0+WDT	18
PIC16CR83	512x14 ROM	36 | 64	11	4	13	TMR0+WDT	18

TABLA (1.4 - 4) Algunos modelos de la gama media.

GAMA ALTA PIC17"X"XXX

Se alcanzan las 58 instrucciones de 16 bits en el repertorio, y sus modelos disponen de un sistema de gestión de interrupciones vectorizadas muy potente. También incluyen variados controladores de periféricos, puertas de comunicación serie y paralelo con elementos externos, un multiplicador hardware de gran velocidad, y mayores capacidades de memoria, que alcanza los 8 k palabras en la memoria de instrucciones y 454 bytes en la memoria de datos.

Quizás la característica más destacable de los componentes de esta gama es su arquitectura abierta, que consiste en la posibilidad de ampliación del microcontrolador con elementos externos. Para este fin, las patitas sacan al exterior las líneas de los buses de datos, direcciones y control, a las que se conectan memorias o controladores de periféricos. Esta facultad obliga a estos componentes a tener un elevado número de patitas comprendido entre 40 y 44. Esta filosofía de construcción del sistema es la que se empleaba en los microprocesadores y no suele ser una práctica habitual cuando se emplean microcontroladores.

GAMA MEJORADA

En esta gama entra el microcontrolador que vamos a utilizar, en donde lo más destacable es su tecnología nano watt, referente a un consumo muy bajo de energía, y los elementos necesarios para la comunicación USB 2.0, como el SIE "Serial Interface Engine" (Motor de Interface Serial) y un transceptor USB, además de muchos otros módulos para otros tipos de comunicación, convertidor A/D, varios TIMER (Temporizadores), vectores de interrupción con nivel de prioridad, etc. Existen modelos incluso más avanzados que estos, con arquitectura de 32 bits para la Memoria de programa y 16 para el bus de datos, en fin, una gama de nuevos modelos.

1.5. LENGUAJES DE PROGRAMACIÓN

Tener buenas costumbres nos ayuda a mejorar como individuos, y la programación no es la excepción. Esta sección habla solo un poco de las buenas costumbres que se pueden convertir en los medios para alcanzar nuestros objetivos.

1.5.1. PROGRAMACIÓN

La técnica que se tomará en cuenta es la programación estructurada, que proviene de los principios de diseño modular.

PRINCIPIO DE DISEÑO MODULAR

Las partes altamente relacionadas deben pertenecer a la misma pieza del sistema.

Las partes no relacionadas deben residir en otras entidades del sistema.

En resumen, lo que tiene relación debe relacionarse, y aquello que no presente afinidad alguna, apartarse.

CAJA NEGRA

Una verdadera caja negra es un sistema que puede ser utilizado en su totalidad sin tener conocimiento de lo que hay en su interior.

MÓDULO

Es un segmento de programa que posee un conjunto de entradas finito, un conjunto finito de salidas y que ejecuta una función. Ver la figura (1.5 - 1).

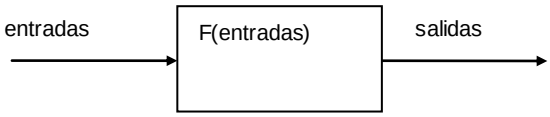


Figura (1.5 – 1) Diagrama a bloques del Módulo.

OBJETIVOS DE LA PROGRAMACIÓN ESTRUCTURADA

La programación está enfocada a las estructuras de control de un programa. La técnica primaria consiste en la disminución de salto incondicional y su remplazo por sentencias bien estructuradas de bifurcación y control. La programación estructurada es un caso especial de la programación modular. El diseño de un programa estructurado se realiza construyendo bloques tan pequeños que pueden ser codificados fácilmente, esto se logra hasta que se alcanza el nivel de módulos atómicos, es decir, sentencias individuales. Ver la figura (1.5 - 2).

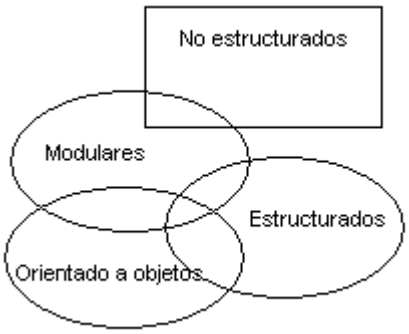


Figura (1.5 - 2). La programación estructurada en el universo de los programas.

TEOREMA DE LA ESTRUCTURA

El teorema de la estructura fue enunciado por Bohm y Jacopini. En él se establece que se requiere de tres bloques básicos para construir cualquier programa.

- Bloque de proceso
- Decisión binaria
- Ciclo de repetición

PROGRAMA PROPIO

Se llama programa propio a aquel que cumple con los siguientes requisitos:

- Tiene un solo punto de entrada hasta arriba
- Se lee de arriba hacia abajo
- Tiene un solo punto de salida hacia abajo

ESTRUCTURAS DE CONTROL

BLOQUE DE PROCESO

Generalmente interpretado como la estructura que se ejecuta en secuencia (de arriba hacia abajo siguiendo el concepto de programa propio). Una característica esencial del bloque de proceso es que no necesariamente puede ser una sola sentencia, sino un conjunto de ellas, cuya ejecución sea secuencial. Incluso un conjunto de cualquiera de las estructuras a continuación presentadas puede ser un bloque de proceso.

DESICIÓN BINARIA

La estructura de decisión binaria más sencilla es la de una sola rama. A esta estructura se le conoce con el nombre de SI-ENTONCES (IF-THEN). En esta estructura una condición es valuada, si la condición resulta verdadera, entonces se ejecuta el bloque de proceso, si la condición es valuada como falso, no se ejecuta ninguna sentencia. Ver la figura (1.5 - 3).

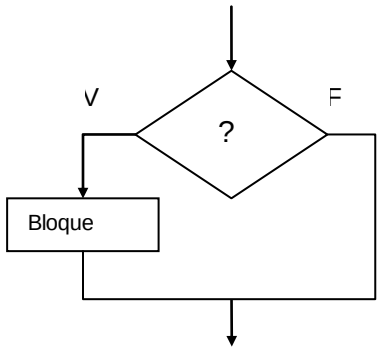


Figura (1.5 - 3) Estructura IF-THEN (SI-ENTONCES) de una sola rama.

El pseudocódigo es:

```
SI (condición es verdadera) ENTONCES
        EJECUTAR Bloque de proceso
FIN SI
```

DECISIÓN BINARIA DE DOS RAMAS

Es similar a la anterior, si la condición valuada es verdadera, se ejecuta el bloque verdadero, si la condición valuada es falsa se ejecuta el boque falso. Ver la figura (1.5 - 4).

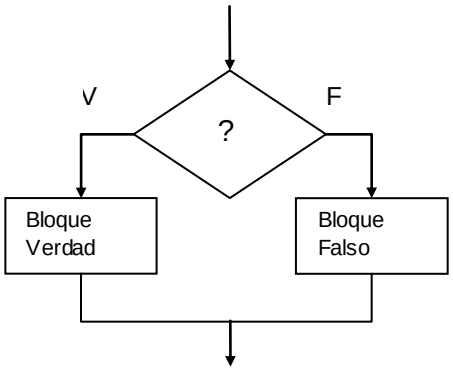


Figura (1.5 - 4) Estructura IF-THEN (SI-ENTONCES) de dos ramas.

El pseudocódigo es:

```
SI (condición es verdadera) ENTONCES
        EJECUTAR bloque verdadero
SINO
        EJECUTAR bloque falso
FIN SI
```

ESTRUCTURAS DE REPETICIÓN

Estructura HAZ-MIENTRAS (DO-WHILE). Si la condición que se evalúa a la entrada es verdadera, entonces se ejecuta un bloque de proceso definido dentro de un bucle en el que, cada vez que termina de ejecutarse el bloque de proceso, se evaluará la condición nuevamente, y la sentencia termina hasta que la condición valuada sea falsa. Ver la figura (1.5 - 5).

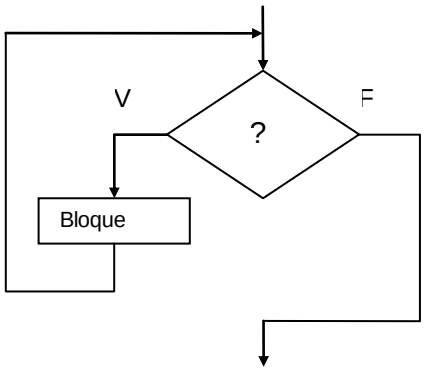


Figura (1.5 - 5) Estructura DO-WHILE (HAZ-MIENTRAS).

El pseudocódigo es:

```
MIENTRAS (condición sea verdadera)
        EJECUTAR bloque
FIN MIENTRAS
```

Estructura REPITE-HASTA (REPEAT-UNTIL). La diferencia radical, con respecto de la anterior estructura, es que a la entrada de la estructura, se ejecuta primero el bloque de proceso y no la condición, además de que sale del bucle solo si la condición valuada es verdadera, a diferencia de la anterior estructura que sale si es falsa. Ver la figura (1.5 - 6).

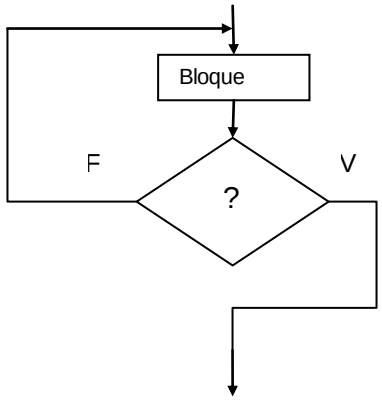


Figura (1.5 - 6) Estructura REPEAT-UNTIL (REPITE HASTA).

El pseudocódigo es:

```
REPITE
        EJECUTAR bloque de proceso
HASTA (condición sea verdadera)
Otra forma similar en pseudocódigo es:
HAZ
        EJECUTAR bloque de proceso
MIENTRAS (condición verdadera)
```

LA ESTRUCTURA PARA

Esta se deriva de las estructuras de repetición. Se caracteriza por incluir una inicialización de variable y un proceso iterativo. Este bloque se repite n número de veces. Ver la figura (1.5-7).

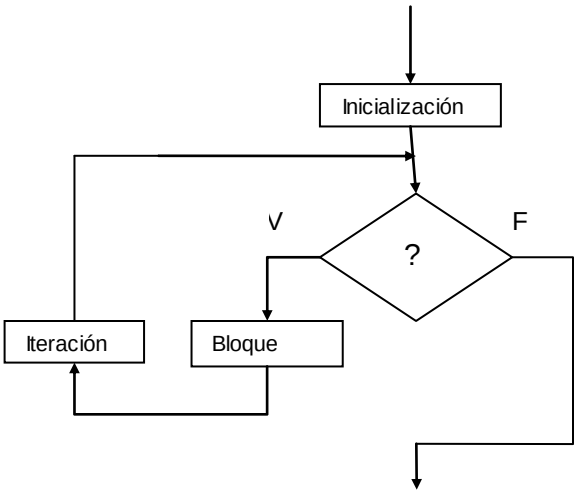


Figura (1.4 – 7) Estructura FOR (PARA).

El pseudocódigo es:

```
PARA (Inicialización; Condición; Proceso Iterativo)
        BLOQUE
FINPARA
```

LA SELECCIÓN MÚLTIPLE

La estructura de selección múltiple es construida a través de estructuras anidadas de selección binaria. La selección múltiple es conocida como estructura CASO (CASE). Véase la figura (1.5 - 8).

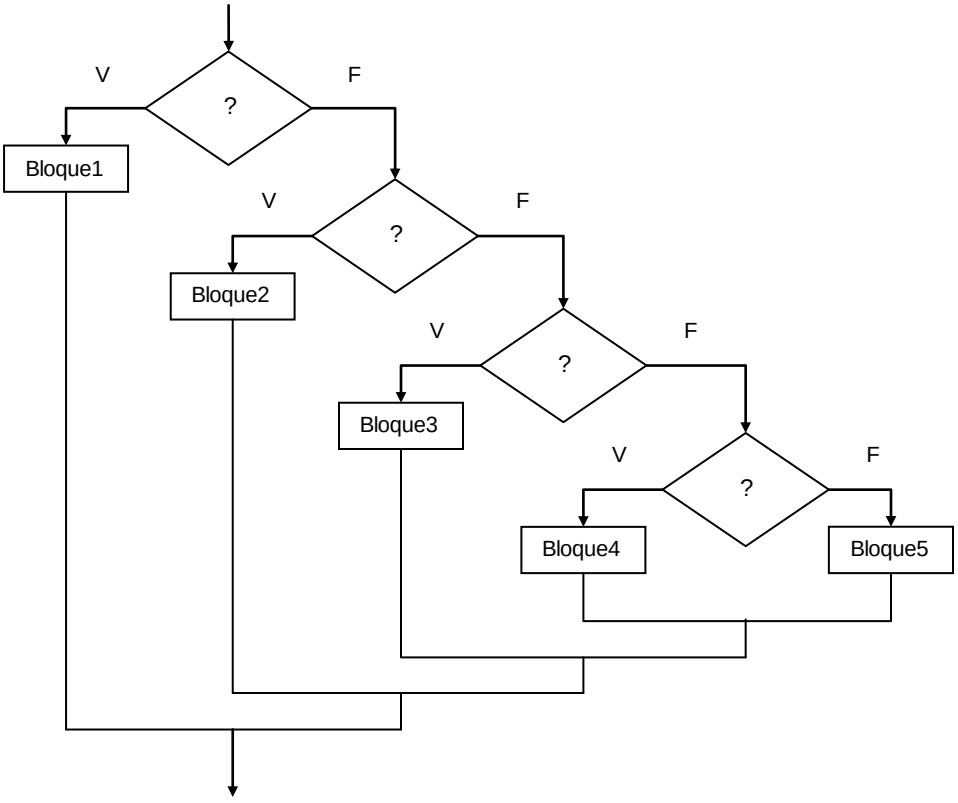


Figura (1.5 - 8) Estructura CASE, derivada de estructuras IF-THEN

En pseudocódigo es:

```
HAZ CASO DE opción
        CASO opción = 1
                EJECUTAR bloque1
        CASO opción = 2
                EJECUTAR bloque2
        CASO opción = 3
                EJECUTAR bloque 3
        CASO opción = 4
                EJECUTAR bloque4
        CASO default
                EJECUTAR bloque5
        FIN CASO
```

CONTENEDORES

Al realizar un programa reservamos localidades de memoria, en donde operaremos o guardaremos los resultados de operaciones. Estas localidades de memoria que reservamos son las variables que vamos a necesitar en la ejecución de un programa, entonces, al escribir un pseudocódigo, también es recomendable escribir las variables. En pseudocódigo podemos escribirlas de la siguiente manera considerando un ejemplo que contiene dos variables:

```
CONTENEDORES
        Contenedor A
        Contenedor B
FINCONTENEDORES
```

En la elaboración de algún programa, realizar los siguientes pasos nos acerca de forma adecuada a la solución que necesita nuestro problema. Planteamiento del problema, lista de pasos para la solución (borrador), algoritmo bien estructurado y prueba de escritorio, esto es lo más recomendable en la solución de un problema, ya que al terminar nuestro pseudocódigo nos sirve para empezar a codificar en el lenguaje que necesitemos.

1.5.2. LENGUAJE ENSAMBLADOR

Conocemos, de forma superficial, las características de un microcontrolador PIC. Ahora sabemos que tiene una arquitectura Harvard, lo que significa en esencia que, tiene memorias independientes. En particular, el PIC que vamos a trabajar se basa en una arquitectura de 8 bits (byte) para trabajar con datos (RAM), y 16 bits (word) para las instrucciones que forman parte del código fuente.

Recordemos que una computadora trabaja señales eléctricas, por esta razón, la señal más simple para trabajar es una señal digital de unos y ceros. Entonces una computadora trabaja con lenguaje máquina, es decir, con señales de unos y ceros. Esta forma de lenguaje es la más primitiva de todas, ya que indicar instrucciones de unos y ceros resulta muy tedioso y complicado. De esta necesidad de tener un lenguaje que se acercara más al de una persona surgió el lenguaje ensamblador, donde una sección de código de unos y ceros es convertido en una palabra que represente el significado de esa instrucción de unos y ceros. A esta palabra equivalente a una sección de unos y ceros se le llama “mnemónico”.

Utilizando, la primera instrucción del juego de instrucciones, vemos de qué trata un mnemónico.

La instrucción ADDWF NUM_A, W, R equivale a esta palabra de 16 bits <15:0>, construido de la siguiente forma:

Del bit 15 al bit 10 (<15:10>) es un código fijo que corresponde a esa instrucción en particular, el bit 9 (<9>) puede tomar dos valores, 1 o 0. El bit 8 (<8>) igual puede ser 1 o 0 y por último del bit 7 al 0 (<7:0>) también varía dependiendo la dirección del registro. Ver el siguiente ejemplo:

15 10 9 8 7 0

Código						d	a	f(FILE#)							
0	0	1	0	0	1	d	a	f	f	f	f	f	f	f	f

Donde:

d = 0; el resultado se envía al WREG (Registro de trabajo)

d = 1; el resultado se envía a FILE que es un registro de RAM indicado por su dirección

a = 0; obliga al microcontrolador a usar el Área “Access Bank” (área de la RAM en particular)

a = 1; puede seleccionar con BSR (selector de banco de memoria)

f = <7:0>; Esta sección es donde se indica una dirección de memoria RAM, y no el contenido de esa memoria

El ejemplo anterior es para entender qué es el lenguaje ensamblador, tomando como base una instrucción del PIC4550.

Entremos de lleno en el tema, para hablar de hacer un programa en lenguaje ensamblador. El programa ensamblador que se usa para los Microcontroladores PIC es el MPASM, que está incluido en el MPLAB IDE de Microchip (Entorno de Desarrollo Integrado).

Al código que va en la memoria de programa del microcontrolador se le llama código fuente, y está compuesto por muchas líneas de instrucciones en un orden lógico y de lista.

En el entorno MPASM se trabaja de la manera siguiente:

1er Columna	2da Columna	3er Columna	4ta Columna
Etiquetas	Código de Operación	Operandos	Notas

Se deben separar las columnas por espacios o tabulaciones, en donde lo más recomendable son uno o dos tabulaciones entre cada columna.

LAS ETIQUETAS

Una etiqueta es una dirección de una instrucción en el programa y sirve para identificar dicha instrucción, para poder recurrir a saltos en el programa que nos lleven a esa sección del código. La ventaja es que podemos escribirlas como una palabra con caracteres alfanuméricos, de esta manera resulta mucho más fácil asignar la referencia en las instrucciones de salto. En el MPASM las etiquetas deben seguir las siguientes normas:

- Deben empezar con alguna letra y posteriormente se pueden escribir más letras o números incluyendo el símbolo “guión bajo” (_).
- Deben ir pegadas a la izquierda sin ningún espacio, o el ensamblador MPASM lo interpreta como una instrucción.
- Tampoco deben utilizarse palaras reservadas, como algunas directivas, nombres de registros especiales o bits especiales.
- Por último, no se debe asignar la misma etiqueta a dos o más instrucciones.

CÓDIGO DE OPERACIÓN

El código de operación es una parte del código en la instrucción; específicamente es la parte constante que indica qué operación es traducida directamente por el ensamblador.

OPERANDOS

Los operandos son el resto de la instrucción; esta parte de la instrucción es variable, ya que es donde se indican las características en particular que queremos que efectúe nuestra instrucción. Los operandos pueden ser literales (constantes), direcciones de registros, bits. Son muy variados y aumentan con la complejidad del microcontrolador.

NOTAS

Estas van en la última columna y deben ir precedidas por un punto y coma (;). Esta parte no va incluida dentro del código de programa que debe llevar el microcontrolador. La utilidad de esta columna radica en la conciencia del programador, dándole la síntesis y propiedad necesaria para que se entienda de forma clara el proceso que se lleva a cabo.

TIPOS DE SINTAXIS EN SISTEMAS DE NUMERACIÓN Y CÓDIGO ASCII

En la tabla (1.5 - 1) del apéndice A, se muestran los sistemas de numeración y el código ASCII que soporta el programa ensamblador MPASM, así como los tipos de sintaxis que pueden declararse o utilizar en el entorno. Se escriben en la tercer Columna, ya que forman parte de los operandos, o suelen serlo si se trata de una literal.

DESCRIPCIÓN DEL CAMPO DE CÓDIGO

Para poder pasar al juego de instrucciones, conviene revisar estas descripciones, para comprender la nomenclatura que se expresa en la tabla de instrucciones. La descripción del campo de código se muestra en el Apéndice A en la tabla (1.5 – 2).

SIMBOLOGÍA UTILIZADA EN LA DESCRIPCIÓN

Para poder entender alguna instrucción, los manuales de microchip usan símbolos para simplificar y representar como operan las instrucciones. Más bien son símbolos didácticos. La simbología es mostrada en el Apéndice A en la Tabla (1.5 – 3).

JUEGO DE INSTRUCCIONES

Es indispensable tener a la mano el set (juego) de instrucciones (mnemónicos) que soporta el microcontrolador PIC4550. Que como se puede observar son secciones de código con un ancho de 16 bits, mejor conocida como palabra o Word. Este juego es mostrado en el Apéndice A.

Secciones:

Instrucciones de Carga o Direccionamiento en la Tabla (1.5 – 4).

Instrucciones Aritméticas en la Tabla (1.5 – 5).

Instrucciones Lógicas en la Tabla (1.5 – 6).

Instrucciones de Decisión con Saltos en la Tabla (1.5 – 7).

Instrucciones Especiales y de Salto en la Tabla (1.5 – 8).

MPLAB IDE

El MPLAB Integrated Development Environment (Entorno de desarrollo Integrado) es un conjunto de herramientas libres e integradas para el desarrollo de aplicaciones empleando microcontroladores PIC®. MPLAB IDE se ejecuta como una aplicación de 32 bits en el sistema operativo Windows®, incluye una gran cantidad de componentes software para el desarrollo y depuración rápida de aplicaciones. Moverse entre las herramientas es fácil y la mejora de simulador para la programación y depuración se realiza en un instante, debido a que MPLAB IDE tiene la misma interfaz de usuario para todas las herramientas.

El MPLAB incluye:

- Editor de texto
- Ensamblador MPASM
- Simulador MPLAB SIM
- Organizador de proyectos

Este programa es gratuito y se puede obtener en la página del fabricante www.microchip.com.

ENSAMBLADOR MPASM

Este es un programa ensamblador que se utiliza dentro del entorno del MPLAB de microchip Este programa es uno de los más utilizados. Se puede obtener gratis en la página web del fabricante www.microchip.com.

DIRECTIVAS

El ensamblador necesita directivas para ensamblar un programa, estas directivas son comandos insertados en el programa que controla el proceso de ensamblado. Estas directivas no forman parte del juego de instrucciones del microcontrolador, esto significa que no tienen traducción al código máquina. Un ejemplo sencillo de una directiva a la que estamos acostumbrados al programar es END, esta directiva es la más intuitiva y le dice al compilador donde termina nuestro programa. Enunciaremos las directivas más utilizadas.

LIST

Descripción:

La directiva LIST tiene efecto sobre el proceso de ensamblado y sobre el formato del fichero listable. El fichero listable es un archivo que contiene los nombres de los registros de función específica con su dirección y con respecto del microcontrolador utilizado, así como el formato de los bits de cada uno de los SFR (Registros de Función Específica) que se necesiten. El ejemplo de uso de la directiva LIST es como sigue, obsérvese que se coloca en la segunda columna.

Ejemplo:

```
LIST P=P18F4550                ; PIC18F4550 como procesador utilizado
```

INCLUDE

Descripción:

El archivo que se especifica en la directiva se lee como código fuente. Se permiten seis niveles de anidamiento. Si se especifica totalmente la ruta, solo se busca en esa ruta. Si no se indica la ruta, el orden de búsqueda es: el directorio activo actual, después el directorio de archivo fuente y por último el directorio ejecutable de MPASM.

Ejemplos:

```
INCLUDE<P18F4550.INC>                ;Define el archivo donde están definidos todos
                                     ;los registros del PIC18F4550
INCLUDE"P18F4550.INC"                ;También se puede definir de esta forma
INCLUDE<c:\TESIS\programa1\led.INC>  ;Busca led.INC en la ruta especificada
INCLUDE<led.INC>                     ;Define led.INC sin trayectoria
```

CONFIG

Para la definición de los bits de la palabra de configuración del microcontrolador con el valor descrito. Hay que notar que no se está escribiendo __CONFIG (con dos guion bajo antes), esto es porque para la familia del PIC18F4550 la directiva va sin los guion bajo, solo para las familias de la gama anterior se sigue utilizando con los guiones. También cabe señalar que, al escribir esta directiva en el editor de texto, el ensamblador no la pintará de color (predefinido azul), como en las demás directivas, esto no nos debe afectar creyendo que no la reconoce. También, como son muchos bits de configuración, no alcanza una sola línea, así que se debe escribir varias.

Ejemplo:

```
CONFIG FOSC = XTPLL_XT, PLLDIV = 1, CPUDIV = OSC1_PLL2
CONFIG WDT = OFF, WDTPS = 1,MCLRE = ON,PBADEN = OFF
```

CBLOCK y ENDC

La directiva CBLOCK asigna direcciones a una lista de etiquetas (variables), estas direcciones deben ser de GPR (registros de propósito general). Se debe especificar la dirección de la que parte, para asignar direcciones a los demás registros en orden ascendente.

La directiva ENDC indica que ahí termina nuestra lista de contenedores.

Ejemplo:

```
CBLOCK    0x00      ;Al primer contenedor se le asigna la dirección 0x00
                    ;de GPR
NumeroA             ;NumeroA = 0x00
NumeroB             ;NumeroB = 0x01
ENDC                ;Fin de la lista de contenedores
```

NOTA: Se puede utilizar varios CBLOCK en el código fuente y sin indicar la dirección, pues el segundo CBLOCK continuará la asignación teniendo en cuenta el bloque anterior de CBLOCK, ENDC.

#DEFINE

Esta directiva define una cadena de substitución de texto. Se escribe desde la primer columna.

Ejemplo:

```
#DEFINE    LEDROJO    PORTB,1      ;Escribir LEDROJO, será igual que PORTB,1
#DEFINE    LEDVERDE   PORTB,2      ;Led de encendido
```

EQU

Esta directiva permite asignar un valor (constante) a un identificador (nombre). El ejemplo común sería PI EQU 3.14, pero solo se permite números enteros con un rango de 0 a 255.

Ejemplo:

```
Instruccion1 EQU    b'10010111'
```

ORG

Generalmente se requiere cuando se utilizan vectores de interrupción. Los vectores de interrupción se pueden definir como aquellas direcciones especiales en la memoria del programa, en las cuales al ocurrir un evento de interrupción, el contador de programa hará un salto a esa dirección. En algunos casos la directiva ORG, se utiliza para indicar donde es el inicio del programa, esto ocurre cuando compilamos dos archivos para ensamblar un código único.

Ejemplo:

```
ORG 0x00                ;Dirección 0x00 de la memoria de programa,
goto        inicio      ;se considera un vector, ya que es la primer
                        ;instrucción y se ejecuta tras cada reinicio del microcontrolador.
ORG 0x04                :Vector de interrupción
goto        Interrupcion
```

END

Esta directiva indica el final del programa y es obligatoria

Ejemplo:

```
Principio
    ;código fuente
    END             ;Fin del programa
```

PASOS PARA ELABORAR UN PROGRAMA

Al elaborar un programa primero debemos saber cómo utilizar el programa, ya sea crear, guardar y hasta simularlo, como es en el caso del MPLAB. El programa propuesto es:

El ejemplo más recurrente y sencillo para microcontroladores es el de encender y apagar un led.

1 Descargar el programa gratuito MPLAB IDE de la página oficial, que en la creación de este trabajo va para la versión 8.5.

2 Si es posible, también es bueno descargar el manual de uso y/o instalación.

3 Seguir los pasos para su correcta instalación en el sistema operativo WINDOWS.

4 Crear una carpeta para poner nuestros programas, siempre es mejor. Se sugiere en el disco duro para referirnos a nuestros archivos, utilizando una dirección más corta, aunque también se deja a la preferencia. La única restricción es que la trayectoria absoluta, o path del fichero, no debe exceder los 62 caracteres.

Ejemplo: “C:TESIS\Codificador”

5 Abrir el programa dándole doble click al icono del programa. 

6 Crear un proyecto. Al crear un proyecto, es util el organizador de proyectos, que es el que nos permite asignar o quitar los archivos que tengamos, según nos convenga.

Existen varias formas de crear proyectos, la más facil es utilizando la opción Proyect Wizard…, que está en la barra de menu en el boton Project.

Aparece el Project Wizard que nos guia para crear el proyecto.

Primero le damos un click en siguiente.

En el paso 1 aparece una ventana, le indicamos el dispositivo a usar (PIC18F4550), y damos click.

En el paso 2 aparecen nuevas opciones, las dejamos como estan, son las siguientes:

En Active Toolsuite: Microchip MPASM Toolsuite

en Toolsuite Contents: MPASM Assambler

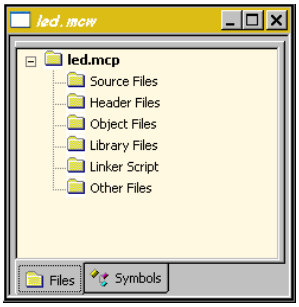
En Location también se deja como está, ejemplo C:\Archivosdeprograma\Microchip\MPASM Suite\MPASMWIN.exe y le damos click en siguiente.

Aparece el paso tres, damos click donde dice Browse… aparece una pantalla que dice save project as (salvar proyecto como), le damos click en la ventana y elegimos C:\TESIS (carpeta TESIS en el disco duro), despues, crear una carpeta con el nombre de nuestro proyecto, esto se hace dando click con el boton secundario en un area en blanco de esa misma ficha y eligiendo la opcion nuevo>carpeta. Al crear la carpeta le damos click para abrirla, le damos un nombre al proyecto y click en guardar.

Al terminar debe quedar en la ventana una ruta similar C:\TESIS\LED\led.mcp, le damos click en siguiente.

En el paso 4 podemos agregar archivos previamente creados. Como no es el caso le damos click en siguiente.

Por último, aparece una ventana con las características de nuestro proyecto y le damos click en finalizar. Como resultado aparece el organizador de proyectos, que es una ventana con dos fichas. La primera ficha llamada “Files”, con un arbol donde la carpeta principal es la llamada led.mcp y la segunda ficha llamada “Symbols”. Ver la figura (1.5-9).



Figura(1.5 - 9) Organizador de proyectos.

7 Abrimos el editor de texto con Ctrl+N ó en la barra del menu Seleccionamos File>New ó click en el icono con figura de hoja.

Nos vamos a la barra de menu y elegimos “File>Save As…”, aparece una ventana, damos el nombre de nuestro archivo, tambien le podemos poner el mismo nombre pero con la extension .ASM (Led.ASM). Damos click en el cuadrito que dice Add File To Poject (agregar archivo al proyecto), es importante asegurarse que el File (archivoLed.ASM) este direccionado en su respectiva carpeta donde se encuentra el proyecto). A consecuencia de haberlo hecho bien, en la carpeta Source Files (Archivos fuente) del arbol del organizador de proyectos se mostrara el archivo guardado (archivoLed.ASM).

8 Ahora podemos escribir nuestro código fuente, utilizando las reglas del MPASM.

Aunque es un ejemplo sencillo, se recomienda construirlo, con sus respectivos comentarios, notas, etc. Como es un ejemplo didactico, contiene comentarios extra de los que es posible prescindir.

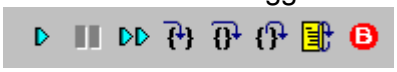
```
;*************************************************
;UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
;FACULTAD DE ESTUDIOS SUPERIORES
;"ARAGÓN"
;
;INGENIERÍA MECÁNICA ELÉCTRICA
;;
;NOMBRE DEL PROYECTO
;ENCENDIDO DE UN LED; PIC18F4550
;
;DESCRIPCIÓN
;Programa básico para la identificacion de una estructura básica.
;Ajuste del oscilador.
;Ajuste de los bits de configuración.
;Uso del juego de instrucciones (palabra de 16 bits).
;de código fuente en un pic de la familia P18F4550.
;Orden en la estructura de cualquier programa.
;*************************************************
;Columna1       Columna2        Columna3        Columna4
;Etiquetas      CódigosOperandos        Comentarios

;CABECERA
;En esta sección deben ir incluidas las siguientes directivas.
                LIST    P=P18F4550              ;Selección del procesador.
                INCLUDE<P18F4550.INC>           ;Archivo que contiene las definiciones de los
                                                ;operandos utilizados.

;Tambien los bits de configuración. Se recomienda ver el archivo P18F4550.INC incluido en las
;carpetas del MPASM, regularmente esta en la siguiente ruta:
;"C:\Archivos de programa\Microchip\MPASM Suite"
; En este archivo se muestran las opciones que tenemos y como deben declararse.

                CONFIG PLLDIV = 3       ;Dividido entre 3 (Oscilador externo de 12MHz).
                CONFIG CPUDIV = OSC4_PLL6 ;96MHz del PLL, dividido entre 6 (16MHz).
                CONFIG USBDIV = 2       ;El reloj fuente del USB viene del PLL 96Hz entre 2.
                CONFIG FOSC = HS        ;Oscilador HS.
                CONFIG FCMEN = OFF      ;Habilitado  reloj monitor Fail-Safe.
                CONFIG IESO = OFF       ;Deshabilitad conmutacion oscilador interno/externo.
                CONFIG PWRT = ON        ;Habilitado el Timer Power-up.
                CONFIG BOR = OFF        ;Deshabilitado el Reinicio Brown-out.
                CONFIG VREGEN = OFF     ;Deshabilitado el regulador de voltaje USB.
                CONFIG WDT = OFF        ;Deshabilitado el WatchDog Timer.
                CONFIG MCLRE = ON       ;MCLR habilitado; pin de entrada RE3 deshabilitado.
                CONFIG LPT1OSC = OFF    ;Timer1 para operar a a mayor potencia.
                CONFIG PBADEN = OFF     ;Pines PORTB<4:0> como I/O digital.
                CONFIG CCP2MX = ON      ;CCP2 entrada/salida es multiplexado con RC1.
                CONFIG LVP = OFF        ;Fuente ICSP dshabilitada.
                CONFIG ICPRT = OFF      ;Depuración/programación en circuito del ICPORT.
                CONFIG XINST = OFF      ;Juego de intrucciones extendida deshabilitado.
                CONFIG DEBUG = OFF      ;Deshabilita depurador; RB6 y 7 como I/O general.
```

```
;CONTENEDORES
;En este programa no es necesario, pero reservaremos una localidad.
            CBLOCK      0x00        ;Primer diección del acces banck, propuesta.
            iNumero_n               ;El programa ensamblador entiende que al
                                    ;referirnos a iNumero_n se trata de la dirección
                                    ;0x00 del accses banck de los registros GPR (Registros
                                    ;de Propósito General).
            ENDC                    ;Termina de asignar variables.

;PROGRAMA PRINCIPAL

            ORG         0x00        ;INICIAR el programa en la direccion 0x00 de la
                                    ;memoria de programa.

;CONFIGURACIÓN DE PUERTOS
;La configuración de puertos es en donde indicamos el tipo y/o dirección que tendrán nuestros
;puertos, lo mas recomendable es consultar la sección de puertos en nuestro manual. En resumen
;se indica nuestras señales de entrada y salida, ya sean niveles lógicos o analógicos etc.

;Primero asegurar puertos como I/O digitales cancelando el convertidor A/D y el módulo comparador.
                                    ;CONFIGURAR el A/D para I/O digitales.
            movlw       0x0F        ;CARGAR cte. 0x0F al WREG.
            movwf       ADCON1      ;COPIAR WREG en el registro ADCON1.
                                    ;CONFIGURAR los comparadores para I/O digitales.
            movwf       0x07        ;CARGAR cte. 0x07 al WREG.
            movwf       CMCON;COPIAR WREG en el registro CMCON.
                                    ;Dirección de los  PUERTOS A y B.
;Los puertos estan configurados por default como entrada.
                                    ;CONFIGURAR PUERTOB como salida.
            clrf        TRISB       ;Los bits de los PIN del PUERTOB que contienen 0
                                    ;seran configurados como salida.
;PROGRAMA
Inicio:                             ;La etiqueta, no necesita los ":" pero se pueden
                                ;agregar.
                                    ;LEER PUERTO A.
            movf        PORTA,W     ;COPIAR el contenido del PUERTOA en WREG.

            movwf       iNumero_n   ;COPIAR WREG en el registro iNumero_n.

                                    ;APLICAR mascara (solo acepta nibble bajo).
            movlw       0x0F        ;CARGAR cte. 0x0F en WREG.
            andwf       iNumero_n,F ;AND WREG con el registro iNumero_n.

                                    ;APLICAR Complemento a dos (c'2) a iNumero_n.
            comf        iNumero_n,F ;APLICAR Complemento a uno a iNumero_n
                                    ;el resultado queda en F (iNumero_n).
            incf        iNumero_n,F ;SUMAR 1 a iNumero_n, el resultado queda en F.

            movff       iNumero_n,PORTB     ;MOVER iNumero_n al PUERTOB.

            goto        Inicio      ;SALTAR Etiqueta Inicio.

                                    ;FINALIZAR.
            END                     ;Directiva que indica el FIN del programa.
```

---

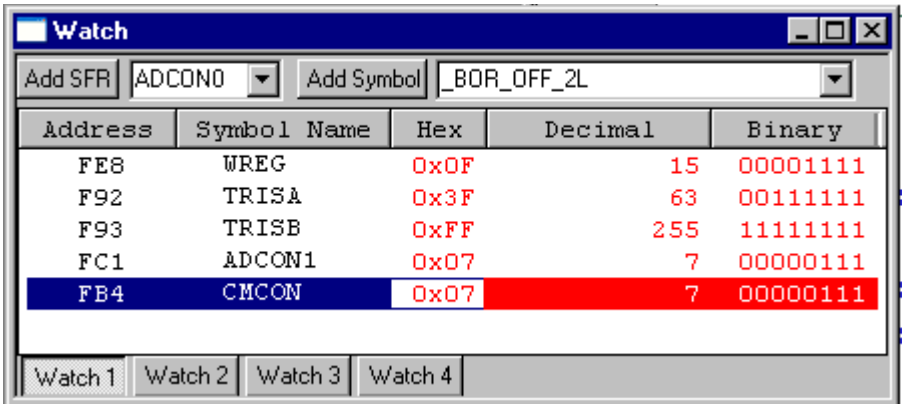
9 Ahora a compilar. Si nuestro código está mal, deberemos corregirlo guiandonos con la ventana de salida Output, que nos indica errores; normalmente violaciones a las reglas del ensamblador con el mensaje en rojo BUILD FAILED. Cuando compila correctamente, el mensaje obtenido en la ventana Output será de otro color (azul):

```
MPLINK 4.30.01, Linker
Copyright (c) 2009 Microchip Technology Inc.
Errors    : 0
```

10 El paso siguiente es simularlo con la herramienta MPLAB SIM. Primero abrimos la herramienta de simulacion, eligiendo desde la barra del menu: “Debugger/Select Tool/MPLAB SIM” y aparecerá el menu de botones siguiente: 

- Para correr el programa
- Para pausar el programa o simulación
- Simula paso a paso pero en modo continuo
- Paso a paso
- Paso a paso se detiene en un break point
- Ejecuta un bloque (subrutina) hasta encontrar un break point
- Reinicio
- Ubica un break point (punto de ruptura)

Ahora, eligiendo desde la barra del menu “View>Watch”, saldra la ventana de watch, que sirve para ver los registros de la memoria RAM. La manera mas facil de mostrarlos es escribiendo el nombre del registro en las celdas del watch y dar enter. Ver la figura (1.5 - 10).



Figura(1.5 - 10) Ventana de simulación watch

11 Para poder simular las entradas es necesario utilizar la herraminta de estimulos, que se encuentra en la barra del menu, seleccionando “Debugger/Stimulus”.

Dar clik en la ficha de Asynch (asincrono), en la columna de Pin/SFR elegimos los RA0, RA1, RA2, RA3 y en la columna de acción elegimos una condición. Debe de quedar semejante a esta ventana (Figura (1.5 - 11)). Por último, en el tiempo de simulación debemos dar click en los botones de la columna Fire, para que se activen nuestras señales y podamos analizar nuestra simulación.

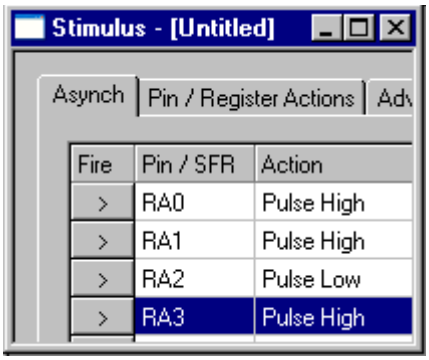


Figura (1.5 - 11) Ventana de simulacion Stimulus

Al terminar la simulación, si queramos cerrar el programa, nos preguntara si queremos salvar el libro de trabajo stimulus. Y si queremos que cada vez que abramos el proyecto, el workbook este ahí, damos click en si, damos un nombre y guardamos.

El MPLAB IDE cuenta con otras herramientas de simulacion, donde podemos ver la memoria de programa, la memoria EEPROM, los registros de proposito específico o general, la pila (Stack), etc. En resumen, son ventanas que muestran estos registros y, al mismo tiempo que simulamos, estos muestran los cambios. Todas estas herramientas se pueden elegir seleccionando desde la barra del menu en la opción View.

1.5.3. LENGUAJE DE ALTO NIVEL

Por esto se puede definir como: “Lenguaje de programación de alto nivel que se parece al ensamblador”.

El lenguaje C tiene características muy peculiares: es posible manipular los registros del microprocesador, puertos y muchas otras características del hardware. El lenguaje C es para programadores y sorprende que no contiene operaciones para trabajar directamente con elementos compuestos, tales como cadenas de caracteres, conjuntos, listas, arreglos o vectores considerados como un todo.

El libro base del lenguaje C es “El lenguaje de programación C”, escrito por Kernighan y Ritchie. C no cuenta con operaciones de entrada-salida, no existen proposiciones READ y WRITE, ni métodos propios para el acceso a archivos. C solo ofrece proposiciones (sentencias) sencillas de control de flujo, secuenciales, de selección, de iteración, bloques y subprogramas. C es relativamente pequeño y puede aprenderse fácilmente.

Existen muchos tipos de compiladores de C, pero el que se utilizará es el de CCS (Custom Computer Services), fabricante del PIC C Compiler, que ofrece su versión de compilador para sistema operativo Windows, e incluso Linux. Este compilador soporta las familias de

microcontroladores PIC10, PIC12, PIC14, PIC16, PIC18 y ahora soporta los chips PIC24. El PIC C ofrece numerosas librerías con rutinas listas para correrse en programas, por ejemplo periféricos para hardware incrementando su nivel de productividad en el desarrollo. Esta última característica es la que nos va a permitir utilizar el módulo USB del PIC18F4550. De esta forma podemos concentrarnos en las demás características propias del diseño de nuestra función.

El compilador PIC C contiene múltiples funciones que podemos utilizar en nuestro diseño, ya que cuenta con muchas librerías destinadas a facilitar la configuración y control de nuestro PIC. Pero antes debemos conocer características generales y esenciales que requiere un código en C, independientemente del tipo de compilador.

TIPOS DE DATOS

En C podemos almacenar los datos en variables. Las variables pueden ser de distinto tipo dependiendo del tipo de dato que se quiera almacenar. No es lo mismo guardar un número que un nombre. Una **declaración de tipo o nombre de variable** es la definición del tipo de datos que va a contener una variable. A las variables no se les puede asignar cualquier nombre. Las reglas son:

- Se pueden poner letras de la “a” a la “z” excepto la “ñ”, números y el símbolo “_”.
- No se pueden poner otros símbolos, por ejemplo caracteres especiales como “?”, etc.
- El nombre puede tener números, pero nunca al inicio del nombre.
- Tampoco se pueden usar palabras reservadas para uso exclusivo del compilador.

Los Tipos de datos son mostrados en la Tabla (1.5 – 9).

TIPO	TAMAÑO EN BITS	SIN SIGNO	CON SIGNO
int1	número de 1 bit	0 a 1	no
int8	número de 8 bit	0 a 255	-128 a 127
int16	número de 16 bit	0 a 65 635	-32768 a 32767
int32	número de 32 bit	0 a 4294967295	-2147483648 a 2147483647
float32	bit 32 flotante	$-1.5x10^{46}$ a $3.4x10^{36}$	

Tabla (1.5 - 9) Tipos de datos

Al escribir los tipos de datos en C estándar, el compilador PIC C los interpreta de la forma siguiente. Ver la tabla (1.5 - 10).

TIPO C ESTANDARD	TIPO C DEFAULT
short	int1
char	unsigned int8
int	int8
long	int16
long long	int32
float	float32

Tabla (1.5 - 10) Tipos de datos, equivalencias con C estándar.

En el tipo flotante se pueden almacenar números decimales.

Dicha declaración en forma general:

tipo lista_de_variables

Donde:

tipo	es un tipo de datos validos en C.
lista_de_variables	consiste en uno o más identificadores (nombres), separados por comas.

OPERADORES

Un operador es un símbolo que representa una operación (de asignación, de relación, lógico, aritméticos y de manipulación de bits), entre variables, caracteres o constantes dentro de un programa. Ver la tabla (1.5 - 11).

OPERADOR	NOMBRE
lógicos	
+	suma
-	resta
*	multiplicación
/	división
%	división en módulo (residuo)
--	decremento
++	incremento
lógicos	
>	mayor que
<	menor que
>=	mayor o igual que
<=	menor o igual que
==	igual
!=	distinto de
!	no
||	O
&&	Y
&	dirección

Tabla (1.5 - 11) Tipos de operadores más utilizados.

La precedencia es el orden en que el programa ejecutará las operaciones, al igual que en la aritmética. Ver la tabla (1.5 – 12).

OPERADOR	PRECEDENCIA
Aritméticos	
( )	Mayor
++ --	
*** / %**	
+ -	
=	Menor
Lógicos	
!	Mayor
> < >= <=	
== !=	
&$	
||	Menor

Tabla (1.5 - 12) Orden de importancia entre los operadores.

LAS CONSTANTES

Una constante es un valor numérico, ya sea entero o decimal normalmente, que a diferencia de una variable, este se mantiene igual. Es importante saber los tipos de expresiones que son validas en el compilador. Ver la tabla (1.5 – 13) del Apéndice A.

EXPRESIÓN	TIPO
123	decimal
0123	octal
0x123	hexadecimal
0b010010	binario
‘x’	carácter
‘/010’	octal
‘/xA5’	hexadecimal
‘/c’	carácter
“abcdef”	cadena de carácter

Tabla (1.5 - 13) Las constantes y expresiones permitidas en PIC C.

ENTRADA Y SALIDA DE DATOS

C no posee instrucciones para la entrada y salida de datos. Estas operaciones se realizan mediante funciones de biblioteca y proporcionan un amplio repertorio de opciones para el programador, permitiéndole escoger la opción que más se adapta al problema que se desea solucionar. Para la salida de datos, ver la tabla (1.5 - 14).

CÓDIGO	FORMATO
%c	Caracter
%d	Enteros decimales con signo (±)
%i	Enteros decimales con signo (±)
%e	Notación científica (e minúscula)
%E	Notación científica (E mayúscula)
%e	Coma flotante
%g	Usar %e ó %f el que resulte más corto
%G	Usar %E ó %f el que resulte más corto
%o	Octal sin signo
%s	Cadena de caracteres (arreglo)
%u	Enteros decimales sin signo
%x	Hexadecimales sin signo (letras minúsculas)
%X	Hexadecimales sin signo (letras mayúsculas)
%p	Mostrar puntero
%n	El argumento asociado es un puntero, a entero al que no se asigna el número de caracteres escritos
%%	Imprimir el signo %

Tabla (1.5 - 14) Códigos de Formato.

Algunos símbolos no están incluidos en el teclado, además el compilador puede confundirlos con los requerimientos de la función como las comillas (“). Para esto se utilizan el código de barras invertidas. Ver la tabla (1.5 – 15).

CÓDIGO	SIGNIFIVADO
\b	Espacio atrás
\f	Salto de página
\n	Salto de línea
\r	Retorno de carro
\t	Tabulador
\”	Comillas
\’	Apostrofo
\0	Nulo
\\	Barra invertida
\r	Tabulador vertical
\a	Alerta (sonido en C estandar)

Tabla (1.5 - 15) Código de barras invertidas.

Para los programas tradicionales en C, la función printf() resulta primordial cuando se trata mostrar los datos en pantalla. Para los microcontroladores no es la excepción ya que existen librerías que se encargan de imprimir nuestros datos en un LCD, o podría ser por interface serial RS 232 a otro programa como hyperterminal. Por esta razón es importante conocer los códigos de formato.

La forma general para imprimir un dato es la siguiente:

```
#include <librería.h>
printf ("cadena de control", lista de argumentos);
```

donde:

cadena de control: es una cadena con los códigos que controlarán la forma como se desplegarán los resultados en el dispositivo de salida.

lista de argumentos: es la lista con las variables o las expresiones que serán desplegadas.

La directiva #include indica al compilador cual archivo de cabecera es necesario para compilar correctamente los programas.

DEFINICION DE VARIABLES

Ya conocemos los tipos de códigos más utilizados, ahora debemos saber que, antes de utilizar una variable, esta debe ser declarada. Existen dos posibilidades: Una es declararla como global y la otra es declararla como local. Global es aquella que puede ser utilizada en todo el programa en general, podemos decir que se declara fuera de la función main. Y local, la que se declara dentro de una función, o aquella que solo puede ser utilizada por una función en particular.

VARIABLE GLOBAL

```
#include <18F4550.h>
Int8 x;
void main ()
{
}
```

VARIABLE LOCAL

```
#include <18F4550.h>
void main ()
{
int8 x;
}
```

DEFINICIÓN DE UNA FUNCIÓN

Las funciones devuelven y reciben datos de determinado tipo. Si los tipos no coinciden y no son declarados explícitamente, C podría compilar exitosamente, pero ejecutar erróneamente. Una función debe tener el siguiente formato:

```
tipo_de_variable        nombre_de_la_funcion        (lista_de_parámetros)
        {
        definición de variables;
        cuerpo de la función;
        return 0;
        }
```

Donde:

tipo_de_variable: Cuando una función de ejecuta y termina debe devolver un valor. Este valor puede ser cualquiera de los tipos de variable mencionados anteriormente (int, char, etc.). Tibien podemos usar el tipo void. Este nos permite devolver cualquier tipo de variable o ninguna. El valor que devuelve la función suele ser el resultado de las operaciones que realizan en la función.

nombre de la función: en este caso se utilizan la mismas reglas que para un nombre de variable.

lista de parámetros: Son variables que se pasan como datos a una función. Deben ir separados por una coma y cada variable debe ir con su tipo de dato.

definición de variables: Dentro de la función podemos definir variables que solo tendrán validez dentro de su propia función. Si declaramos una variable en una función, no podemos usarla otra vez, se dice que la variable muere al terminar la función.

cuerpo de la función: Se entiende que es ahí en donde va el código de la función.

return: La función return, devuelve un valor. El dato que se pone después de return es el dato que se devuelve. Puede ser una constante o una variable. Debe ser del mismo tipo que tipo_de_variable.

Las funciones al igual que las variables deben definirse antes de ser llamadas. Estas se deben definir antes del main (nombre de la función principal).

CONSTANTES

Las constantes son los datos que no cambian en la ejecución del programa. Las constantes se pueden declarar usando la directiva #define.

Forma general:

```
#define     nombre_de_constante     valor_de_constante
```

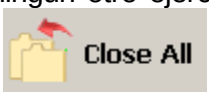
Ejemplo:

```
#define     PI      3.1416
```

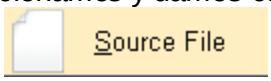
Como puede notarse, el lenguaje C puede resultar un poco difícil al principio, sobre todo si no se está familiarizado con la programación. En este trabajo solo requeriremos de los conceptos básicos.

GUARDAR Y COMPILAR UN PROGRAMA EN PIC C

Es bastante sencillo, solo hay que seguir los siguientes pasos:

1. Crear una carpeta para guardar nuestro proyecto.
2. Abrir el programa PIC C
3. Damos click, con el botón principal del mouse, en el icono superior izquierdo con forma de carpeta  , y nos aseguramos que no este activo ningún otro ejercicio dando click, con el botón principal del mouse, en el icono cerrar todo 

4. Creamos un nuevo proyecto dando click en el icono  , después seleccionando el icono nuevo 

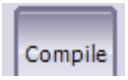
y por ultimo seleccionamos y damos click, con el botón primario del mouse, en archivo de código fuente 

. Aparecerá la ventanita conocida de guardar como, y le indicamos en donde lo queremos guardar, le ponemos un nombre y damos click en el botón guardar.
5. Escribimos nuestro código fuente, como en el ejemplo siguiente:

El siguiente ejemplo de un programa en C para microcontrolador ayudará a aterrizar algunos de los conceptos.

```
//************************************************//
//UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
//FACULTAD DE ESTUDIOS SUPERIORES
//"ARAGÓN"
//
//INGENIERÍA MECÁNICA ELÉCTRICA
//PRÁCTICAS DE LABORATORIO
//PROGRAMACIÓN EN LENGUAJE "C"
//ORIENTADO A MICROCONTROLADORES
//
// Ortega Nava Carlos Fernando
//
//NOMBRE
//INTERRUPCIÓN EXTERNA (RB0); PIC16F84A
//
//DESCRIPCIÓN
//El programa no realiza ninguna operación importante
//se queda atorado en un bucle, pero atiende el
//servicio de interrupción al estimular PIN_RB0.
//************************************************//
// Como se puede observar, las diagonales sirven para indicar al compilador que se
```

```
//trata de comentarios y no de alguna instrucción

//CABECERA
//En esta sección van algunas directivas, que llaman a librerías que contienen funciones
//utilizadas en el programa.
#INCLUDE <16F84A.H>                    //Librería del PIC que se va a usar.
#FUSES XT,NOWDT,NOPROTECT              //Selección de los bits de configuración.
#USE DELAY(clock=4000000)              //Cristal de 4MHz (al recurrir a funciones de retardo,
                                       //esta librería se encarga de generar el código).
//Estas librerías ayudan a configurar los puertos como digitales.
#USE STANDARD_IO(A)                    //Librería de configuración del puerto A.
#USE STANDARD_IO(B)                    //Librería de configuración del puerto B.


//CONTENEDORES
//Aquí se pueden declarar variables globales.

//DECLARACIÓN DE FUNCIÓN
//Como se puede observar, solo declaro antes de main, para poder ser llamada por la
//estructura principal del programa.
void CONFIG_INT_EXT ();                //Rutina que configura la Interrupción externa.

//PROGRAMA PRINCIPAL
//main es una palabra reservada del compilador, significa principal e indica que se trata
//de la etructura principal de programa o que es ahí donde comienza el programa.
void main (void)
  {
        output_high(PIN_A0);           //Inicia con PIN_A0 como salida con pulso alto
         CONFIG_INT_EXT ();            //Configura interrupción externa (función prototipo)

        while(TRUE){}                  //Bucle infinito, sin ninguna operación interna
  }                                    //esperando la interrupción externa para hacer
                                       //el salto a la interrupción

//ESTRUCTURA DE FUNCIÓN PROTOTIPO
//Esta directiva permite establecer la función sin necesidad
//de declararla y debe ir junto a la función.
#INT_EXT                               //Interrupción Externa (PIN_RB0)
void INTERRUPCION_EXTERNA ()           //nótese que no lleva punto y coma ";"
  {                                    //por (PIN_RB0)
  output_toggle(PIN_A0);               //Alterna el estado del (PIN_RA0)
  }

void  CONFIG_INT_EXT ()                //Rutina que configura la Interrupción externa
  {
  enable_interrupts(INT_EXT);          //Habilita la interrupción externa
  ext_int_edge(L_to_H);                //Interrupción de flanco bajo a alto
  enable_interrupts(GLOBAL);           //Interrupción global habilitada
  }
;_____________________________________________________________________________
```

6. Para compilar seleccionamos, de la barra de menú, la ficha compile 

, después seleccionamos el path y dando click en el triangulo, nos ofrece tres tipos de compiladores, y seleccionamos el que es para la familia del P16F84A, “PCM 14bit”.
7. Por último compilamos el programa, dando click en el botón de esa misma ficha compile.


8. Listo, ahora podemos revisar en la carpeta donde guardamos el proyecto los archivos que nos generó. La extensión de estos archivos son: .bak, .cof, .esym, .lst, .sta, .tre, .c, .err, .pjt, .sym, y .hex .
9. El que nos sirve es el .hex (código hexadecimal) y lo podemos usar para grabar nuestro PIC.

1.6. PROGRAMACIÓN DEL MICROCONTROLADOR

En este caso se expone un ejemplo de cómo se realiza una grabación de microcontrolador. Y no es por fuerza que se tenga que seguir estos pasos, ya que existen varios tipos de programadores y software a disposición, que se encargan de hacer la misma tarea. Solo hay que revisar, en caso de hacerlo con otro equipo y software, si estos elementos soportan el microcontrolador con el que se trabaja.

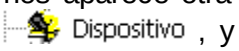
Para grabar un microcontrolador se debe contar con lo siguiente:

1. Una computadora personal, que cuente con puerto serial (DB-9).
2. Un cable de conectores (DB-9) macho y hembra.
3. Un grabador, también llamado programador, JDM. Existen diversos tipos, solo hay que cuidar que se pueda montar el tipo que elegimos.
4. Un software para esta tarea, yo recomiendo el WINPIC800, es fácil de instalar, no necesita parches y cuenta con drivers para diversos sistemas operativos, entre ellos XP, que es la plataforma donde se probó.

Una vez que contamos con la lista de material y software ya instalado, podemos proceder a grabar.

Los siguientes pasos, solo cuando es la primera vez que vamos a programar.

1. Abrir el programa WINPIC800.
2. En el caso que lo hayamos instalado en otro idioma, hay que buscar en la barra del menú, la ficha referente a idioma (la segunda a partir de la izquierda), le damos click y seleccionamos la de español.
3. En la barra de menú, damos click en configuración y seleccionamos hardware. Aparecerá una ventanita, y en el path que dice selección del hardware, buscamos el llamado JDM Programmer, lo seleccionamos y damos click, el path que está debajo de este cambiará al puerto en función (COM1), puede variar dependiendo el número de puertos seriales disponibles. Damos click en el botón confirmar cambios y tenemos bien configurado el hardware.

4. Esta vez también es en configuración pero seleccionamos software, nos aparece otra ventanita con un árbol. Seleccionamos y damos click en dispositivo 

, y dando click en los cuadritos, seleccionamos ambas opciones y damos click en el botón confirmar cambios, es todo.

Los siguientes pasos son para programar una vez configurado el WINPIC800.

1. Montar correctamente el PIC en el programador.
2. Con ayuda del cable DB-9, lo conectamos a la PC.
3. Abrimos el programa WINPIC800 y al momento de abrirse, este deberá desplegar un mensaje diciendo que está detectado el microcontrolador y el tipo de microcontrolador.
4. Seleccionar y dar click a la pestaña archivo de la barra de herramientas, se despliega una ficha y damos click en donde dice abrir.
5. En la ventanita llamada abrir, indicamos donde está el archivo .hex que queremos programar en el PIC y lo seleccionamos, ya sea dando doble click sobre el archivo, o seleccionando con un click y después otro click en el botón abrir.
6. Por último, damos click en el botón programar todo  . Ahora podemos probar el PIC en nuestra protoboard.

2. Implementación del sistema de adquisición

2.1. METODOLOGÍA DEL DISEÑO

A la hora de diseñar circuitos, puede resultar complicado, y regularmente se debe a que pensamos en el sistema en general. Aunque hay quienes van más allá, no siempre se puede controlar todo un mundo de variables y conceptos en un solo circuito. Esta es la razón por la que mi investigación me llevó a estos párrafos. No se trata de hacer como dice en el texto, sino ver las cosas desde diferentes puntos, para que nos pueda ser más fácil abordar los problemas que se nos presentan. También se puede notar que tiene relación con la programación estructurada, así que sirve como complemento.

Esta sección aborda el tema del diseño de circuitos y sistemas integrados desde el punto de visión metodológico, es decir, se presentarán las técnicas y métodos más habituales, utilizados cuando se aborda el diseño de un circuito microelectrónico y las herramientas que el diseñador tiene a su disposición. El problema de un diseño de sistema integrado, como los que hoy en día podemos encontrar, es tan complejo que la primera metodología consiste en la estructuración y jerarquización del sistema, de forma que su diseño puede ser abordado por partes. Además, debido a la gran cantidad de variables que se deben controlar, es necesario enfrentarnos al diseño a diferentes niveles de abstracción, lo que nos permite reducir la cantidad de información que es necesario manejar en cada momento.

Antes de comenzar a abordar el diseño de un sistema, es necesario contar con una forma de expresarlo, que nos permita escribir sus especificaciones, describir su estructura, introducirlo en un equipo informático para su proceso automático, documentarlo, etc. La descripción de un sistema en general, y de un circuito o sistema microelectrónico en particular, se basa en tres procesos: La jerarquización, la abstracción y la representación. El concepto de jerarquización, consiste en la subdivisión del sistema en bloques de forma recursiva, para conseguir que el niel de complejidad de cada parte sea abordable, ya que, en la mayoría de los casos, tratar todo el sistema de forma unitaria es imposible. No obstante, para determinar algunos aspectos será necesario manejar bloques de complejidad considerable, y por ello que deberemos describir el sistema de forma que sea posible manejar la información justa y necesaria, y descartar los detalles que no necesitamos. En otros casos será necesario tener en cuenta todos los detalles y la información disponible, pero entonces deberemos concentrarnos por separado en pequeñas partes del sistema cuyo nivel de complejidad sea tratable. Este proceso es el que denominamos abstracción.

Todo sistema electrónico, y los circuitos integrados en particular, no son otra cosa, en definitiva, que sistemas físicos compuestos por distintos materiales, cuyas propiedades eléctricas (y en algunos casos mecánicas, químicas o térmicas) se utilizan para representar y procesar información. A pesar de ello, no sería viable abordar el diseño de uno de estos sistemas completos a nivel físico, dada la complejidad del aparato matemático que necesitaríamos y la cantidad de variables a tener en cuenta. No obstante, este tipo de metodología nos permite la representación más exacta posible del sistema. Para reducir la cantidad de información manejada se recurre al proceso de abstracción, por lo cual se reduce en un conjunto reducido de propiedades y elementos del sistema, mediante los cuales es factible abordar el problema de su diseño, especificación e implementación. Este proceso de abstracción se concreta en la estructuración a distintos niveles de abstracción del sistema (o sus partes), desde el nivel físico, el más fundamental, pero también en el que la representación del sistema es más exacta, hasta el nivel de arquitectura, en el que puede

llegar a describirse el sistema completo, pero en el que no se tiene información sobre muchas de las propiedades definidas en los niveles inferiores.

Entre el nivel físico y el nivel más alto de abstracción, al que llamaremos aquí nivel de arquitectura, existen una serie de niveles más o menos aceptados como típicos: *El nivel eléctrico y el nivel lógico.* Este último nivel, cuyo nombre proviene de los circuitos digitales, no es muy apropiado para el entorno analógico. Vamos a considerar en este texto que el nivel de abstracción equivalente, en los circuitos analógicos a nivel lógico sería, el *nivel de macromodelo.* Como vemos, cada nivel se caracteriza por una forma de describir los circuitos y un conjunto de variables. Independientemente del nivel al que se encuentre descrito el sistema, existen dos formas de representarlo, a las que llamaremos *representaciones o vistas*. La primera de estas dos formas se denomina *vista estructural,* en la que el sistema se describe utilizando la interconexión de bloques o componentes con funciones y propiedades conocidas, o bien, formados a su vez por bloques de niveles inferiores de jerarquía. La segunda es la vista *funcional o comportamental,* en la que el sistema se describe no solo por su estructura, sino por su función, utilizando ecuaciones matemáticas, curvas, algoritmos, tablas, etc. Cualquier elemento del sistema puede representarse de ambas formas (funcional o estructuralmente), y existen mecanismos para obtener una representación a partir de la otra. Podría hablarse de una tercera forma de describir del sistema a la que denominaremos *vista física o implementación*, que no es más que el aspecto real que tendrá el sistema o circuito una vez fabricado.

- *Abstracción:* procedimiento consistente en traducir un sistema desde un nivel inferior, más detallado, a un nivel superior, menos detallado. Aplicable en cualquier vista del diseño.
- *Refinamiento:* procedimiento opuesto a la abstracción, mediante el cual se obtiene una descripción más detallada y completa de un sistema a partir de su descripción en un nivel superior. En la vista estructural este procedimiento coincidiría con la descripción detallada de los niveles inferiores de jerarquía, partiendo de los superiores, aunque también es aplicable a las otras dos vistas del diseño.
- *Síntesis:* a cualquier nivel de abstracción; este procedimiento se utiliza para traducir una representación funcional en su equivalente estructural. Dado que en la mayoría de los casos las soluciones no son únicas, se deben aplicar unos criterios de síntesis que nos permitan optar por la solución que más nos interese en nuestra aplicación concreta.
- *Análisis:* es el proceso inverso a la síntesis, y consiste en encontrar una descripción funcional o abstracta para un sistema descrito estructuralmente. Su principal uso es el de comprobar que una estructura obtenida, mediante un proceso de diseño complejo, se comporta de acuerdo a la descripción funcional original del mismo (sus especificaciones).
- *Generación:* consiste en la obtención de la implementación real de una estructura determinada. En la mayoría de los casos supone descender un nivel de abstracción, y pasar de un una representación estructural a una física. Un ejemplo será la obtención de las máscaras de un circuito a partir de su esquema eléctrico, o la obtención de ese esquema eléctrico a partir de su descripción mediante ecuaciones booleanas.
- *Extracción:* proceso inverso al anterior. Dos ejemplos usuales de ese procedimiento son la extracción de componentes parásitos, a partir del diagrama de mascaras, que se añaden a la descripción de nivel eléctrico del circuito (su esquemático), y la extracción de retardos a partir del esquemático eléctrico para ser utilizados en el nivel lógico. Otro ejemplo sería la extracción de un modelo funcional, a partir del diagrama de bloques eléctrico, para una macrocelda analógica.

Considerando lo anterior, la representación del sistema en bloques se propon según la figura (2.1 - 1).

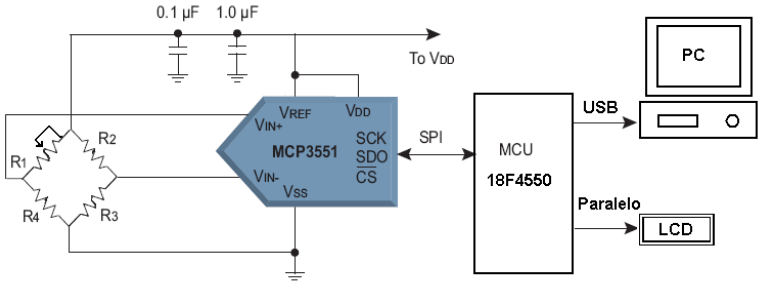


Figura (2.1 – 1) Jerarquización del sistema.

2.2. CARACTERÍSTICAS GENERALES DEL MICROCONTROLADOR PIC18F4550

Las características Técnicas del PIC18F4550 se encuentran en el Apéndice B. La elección entre los tipos de PIC (el 18F2450 y 18F4550) se realiza escencialmente porque es del que se encuentra más información en la red, y contiene más puertos digitales. Pero se puede trabajar indistintamente con cualquiera, ya que son muy similares, la diferencia más importante radica en el numero de puertos con el que dispone cada dispositivo, y el módulo convertidor A/D que no usamos. Así que para el diseño de un sistema más pequeño, el modelo 18F2450 es el más recomendable. Y como se mencionó anteriormente, la razón principal por la que se escojieron los microcontroladores PIC es por la gran cantidad de información disponible en la red y los software disponibles para trabajar con estos dispositivos, como es el C compiler.

CARACTERÍSTICAS GLOBALES

Esta familia de dispositivos ofrece las ventajas de todos los microcontroladores de la familia PIC18F como son: computadora avanzada, presentación en un económico precio, agregando un alto rendimiento y memoria de programa flash reforzada.

TECNOLOGÍA NANO WATT

Todos los dispositivos de la familia PIC18F incorporan características que pueden reducir significativamente el consumo de energía durante la operación, incluidos artículos clave:

- Modos de funcionamiento alternado: Para la generación de ciclos de reloj para el Timer1 u oscilador interno RC, el consumo de poder durante la ejecución del código puede reducirse como un 90%.
- Modos inactivos múltiples: El controlador puede trabajar con el CPU apagado pero los periféricos en funcionamiento. En estos estados, el poder de consumo puede ser reducido aun más, a tan solo un 4% del requerimiento normal de operación.

- Modo de cambio volado: El modo de manejo de poder es invocado por el código de usuario durante el funcionamiento, permitiendo al usuario incorporar ideas para ahorrar poder dentro del diseño de software.
- Consumo bajo en módulos clave: Los requerimientos de poder para ambos. Timer1 y Watchdog Timer, son reducidos.

UNIVERSAL SERIAL BUS (USB transmisión en serie universal)

Los dispositivos en la familia PIC18F4450 incorporan unos rasgos en el modulo de comunicaciones Universal Serial Bus, este es manejado con las especificaciones USB Revisión 2.0. El módulo soporta dos velocidades de comunicación, velocidad-baja y velocidad-total, que soportan diferentes tipos de transmisión. También se agrega en el chip un transceptor y regulador a 3.3V internos, pero puede usar un transceptor o regulador externo.

OPCIONES Y CARACTERISTICAS DEL OSCILADOR MULTPLE

Todos los dispositivos de la familia PIC 18F4050 ofrecen doce opciones de oscilador, permitiendo al usuario varias opciones en hardware de la aplicación en via de desarrollo.

Estas incluyen:

- Cuatro modos de cristal o resonadores cerámicos.
- Cuatro modos de reloj externo, ofreciendo la opción de uso de dos pin (entrada al oscilador y divide entre cuatro en la salida del reloj), o de un pin (entrada de oscilador, con el segundo pin reasignado como I/O general) .
- Una fuente INRTC (aproximadamente 31KHz, estable a temperaturas superiores y VDD). Esta opción deja libre un pin del oscilador para utilizarlo como I/O de propósito general.
- Un Phase Lock Loop (PLL: Cerradura de vuelta de fase) multiplicador de frecuencia, disponible para ambos modos de oscilador, oscilador externo y de gran velocidad que permiten un rango ancho de velocidades de reloj de 4 a 48MHz.
- Funcionamiento del reloj dual asíncrono, permitiendo al modulo USB operar a una alta frecuencia mientras el resto del microcontrolador es sincronizado por un oscilador de bajo poder.
  El oscilador interno proporciona una fuente de referencia estable, que da características adicionales de la familia para una operación compleja.
- Monitoreo del seguro de falla en el reloj: Esta opción constantemente monitorea la fuente de reloj principal, con un signo de referencia proporcionado por el reloj interno. Si una falla del reloj ocurre, el controlador cambia a un oscilador interno, y permite que continúe el funcionamiento a una baja velocidad, o el cierre de la aplicación.
- Dos velocidades de Arranque: Esta opción permite al oscilador interior funcionar como la fuente de reloj para el Power-on-Reset, o despertador para el modo sleep (dormido), hasta que la fuente de reloj esté disponible.

OTRAS CARACTERÍSTICAS ESPECIALES

- Durabilidad de la memoria: La durabilidad de las células de la memoria de programa flash que, promedia unos 100,000 ciclos de borrado y escritura.
- Reprogramación: Estos dispositivos pueden escribir en sus propios espacios de memoria bajo un control interno de software. Usando una rutina Bootloader, localizada en el Boot Block al inicio de la memoria de programa, es posible crear una aplicación que actualice al dispositivo en el campo.

- Juego de instrucciones extendida: La familia PIC18F, introduce una extensión opcional al juego de instrucciones del PIC18F4550, agrega 8 nuevas instrucciones y un modo de direccionamiento de desplazamiento del índice. Esta extensión, activada como una opción de configuración del dispositivo, ha sido específicamente diseñada para optimizar el código aplicado, originalmente diseñado para lenguajes de alto nivel como el C.
- Enhanced Addressable USART: Este modulo de comunicación serial es capáz de operar bajo la norma RS-232 y soporta el protocolo de bus LIN. Otros apoyos incluyen la detección automática de proporción del baudaje y una generación de proporción de baudaje a 16bit, para mejorar la resolución.
- Convertidor A/D 10bit: Este modulo incorpora un tiempo de adquisición programable, permitiendo poder seleccionar un canal e iniciar una conversión, sin esperar al periodo de muestreo y así reduciendo el código fuente.
- Puerto Especializado ICD/ICSP: Estos dispositivos introducen el uso e debugger y programan los pines que no son multiplexados con otras características del microcontrolador. Disponible en paquetes seleccionados, esta característica les permite a los usuarios desarrollar aplicaciones extensivas I/O que reteniendo la habilidad del programa

DETALLES INDIVIDUALES DE MIEMBROS DE LA FAMILIA

Los dispositivos en la familia PIC18F2450/4550 están disponibles en paquetes de 18-pin y 40/44-pin. Los dispositivos se diferencian de otros siguiendo:

1. Canales A/D (Analógico/Digital) (10 para dispositivos de 28-pin, 13 para dispositivos de 40/44-pin).
2. Puertos I/O (3 puertos bidireccionales y un puerto de entrada en dispositivos de 28-pin, 5 puertos bidireccionales en dispositivos de 40/44-pin).
   Todas las otras características para los dispositivos de esta familia son idénticas.
   Estos se describen en la tabla 1-3
   Los dispositivos miembros de la familia PIC18F2450/4550 están disponibles en ambos modos, normal y de bajo voltaje. Dispositivos estándar con memoria de programa flash durable, designados con una F en el numero de la parte (como PIC18F2450/4550), indica un VDD que opera en el rango de 4.2V a 5.5V. Designado con una L (como PIC18LF2450/4450) funciona en un rango amplio de voltaje VDD 2.0 a 5.5V. Ver la tabla (2.2 – 1) en el Apéndice B.

2.3. CARACTERÍSTICAS DEL MÓDULO LCD

Un módulo LCD (Liquid Cristal Display) es una pantalla que tiene la capacidad de mostrar caracteres alfanuméricos. Estos elementos vienen con toda la lógica de control preprogramada. El proceso de visualización es gobernado por un microcontrolador incorporado a la pantalla, incluso algunos manejan comunicación SPI y cuentan con led backligh con varios colores de opción para tener una mejor visualización de los caracteres. Y comparados con los display clásicos de 7 segmentos, estos consumen poca energía y son en definitiva más prácticos.

Existen muchos tipos de display LCD, algunos son de propósito específico, cuentan con imágenes, otros son gráficos, etc. Para ampliar nuestro panorama, en la página web "http://www.displaytech-us.com/" podemos consultar varios modelos de display, además de que podemos descargar la especificación del producto. Si queremos el data shett, podemos consultar la página web http://www.datasheetcatalog.com/, o recomiendo buscar directamente en algún explorador de internet.

Manejaremos un display clásico que despliega caracteres alfanuméricos, conocido como Display de 4x20. Figura (2.3 - 1).

Figura (2.3 – 1) Display LCD 4x20.

En la especificación del producto podemos consultar características como: especificación general, rangos máximos que soporta, características eléctricas, características electro ópticas, dimensiones que tiene etc., Esto nos serviría más si vamos a diseñar un producto, mientras que para el control se usa el data sheett.

Para el control del módulo LCD se debe empezar con la identificación de los Pines. Si observamos los pines del LCD, notamos que los pines 1, 2 y 3 sirven para polarizar y para el contraste. Los pin 4, 5, 6 son RS, RW y E respectivamente, los pin 7 al 14 que corresponden a <DB0:DB7> del bus de datos. Y los últimos dos, 15 y 16, para el led back light. Esto es importante porque así podemos reconocer los esquemas de tiempos. Ver la tabla (2.3 - 1).

Pin No.	Nombre	Señal	Descripción
1	V_{SS}	$0V$	Tierra
2	V_{DD}	$5V$	Tensión de Alimentación
3	V_O	$0 - 5V$	Tensión para ajustar el contraste. Maximo a 0V
4	RS	H/L	Register Select (Selección Registro)
5	R/W	H/L	Read/Write (Lectura/Escritura)
6	E	$H \rightarrow L$	Enable (Habilitador)
7	$DB0$	H/L	Bus de datos $DB7$ bit Busy flag (bit bandera), H.
8	$DB1$	H/L	
9	$DB2$	H/L	
12	$DB3$	H/L	
11	$DB4$	H/L	
12	$DB5$	H/L	
13	$DB6$	H/L	
14	$DB7$	H/L	
15	A	--	Ánodo
16	K	--	Cátodo

TABLA (2.3 - 1) Pines del dispositivo LCD.

Display Data RAM (DDRAM) -- Mostrar Datos RAM

Es un área de RAM que posee el módulo LCD, donde se almacenan los caracteres que se pueden representar. Su capacidad es de 80 bits, o 80 caracteres, 40 por cada línea, en donde, para un modelo de 2x16, solo se pueden visualizar 32 a la vez, pero con el display de 4x20 se pueden mostrar todos al mismo tiempo. Figura (2.3 – 2).

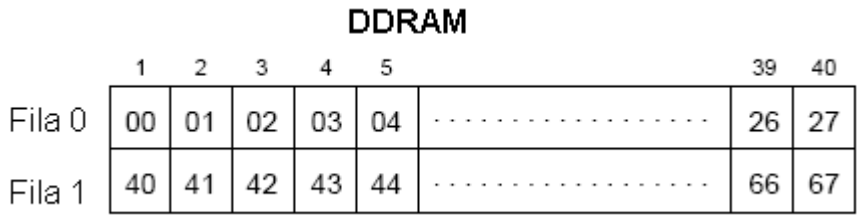


Figura (2.3 - 2) La DDRAM tiene un tamaño de 80 bytes.

Character Generator ROM (CGROM) -- Generador de caracteres ROM

El LCD posee una zona de memoria no volátil, donde se almacena una tabla con los caracteres que pueden ser visualizados. Cada uno de los caracteres tiene su equivalente en número binario. Por ejemplo, si se desea visualizar un carácter, el LCD debe recibir este valor binario por el bus de datos, el nibble alto corresponde a la columna y el nibble bajo corresponde a las filas. Ver la figura (2.3 – 3).

Figura (2.3 - 3) Correspondencia entre código y caracteres CGROM.

Character Generator RAM (CGRAM) -- Generador de caracteres

El usuario puede definir hasta 8 caracteres de 5x7 puntos, o 4 de 5x10. Se seleccionan y visualizan aplicando a la DDRAM cualquier valor entre *00* y *07* o *08* y *0Fh* como si tratase de un código ASCII.

Se definen introduciendo unos bytes en direcciones sucesivas de la CGRAM, cuyos patrones binarios definen al caracter, tal y como se muestra en la Figura (2.3 - 4).

Un carácter de 5x7 necesita de 8 octetos en la CGRAM para ser definido, uno de 5x10 necesita 16. La CGRAM es una memoria de 64 posiciones en total.

En el ejemplo de la figura (2.3 – 4), para definir la “R” de 5x7, se introducen 8 octetos en las 8 primeras posiciones (0 a la 7) de la CGRAM. Cada bit de uno de esos octetos, que valga nivel “1”, hará que su correspondiente pixel se active.

Como es el primer conjunto de 8 bytes, es decir, el primer carácter de la CGRAM, este se selecciona aplicando el código 00 en la DDRAM, como si fuera cualquier otro código ASCII.

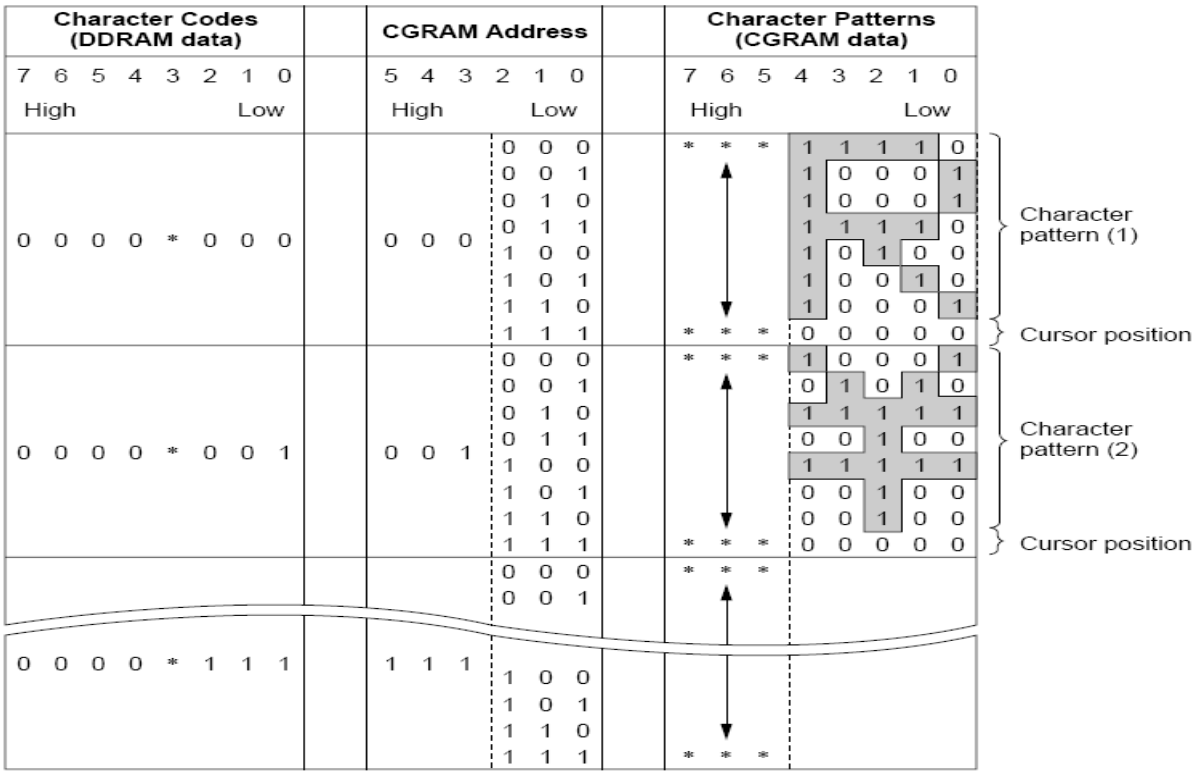


Figura (2.3 - 4) Relacion entre el direccionamiento de la CGRAM, Código de caracteres (CGROM) y patrones de caracteres (CGRAM).

JUEGO DE INSTRUCCIONES PARA EL LCD

Estas son las instrucciones que reconoce el LCD.

- **Clear Display (0000 0001).** *Borra el display y devuelve el cursor a la posición inicial (dirección 00h de la DDRAM).*
- **Return Home (0000 001x).** *Cursor a la dirección origen. Devuelve el cursor a la posición original de la DDRAM (dirección 00h), quedando intacto su contenido.*
- **Entry Mode Set (0000 01 I/D S).** *Mode Entrada. Establece las características de escritura de los datos Shift e Increment/Decrement:*
  - S=0. *La información visualizada en pantalla no se desplaza al escribir un nuevo carácter.*
  - S=1. *La información visualizada se desplaza al escribir un nuevo carácter. La pantalla se desplaza en el sentido indicado por el bit I/D cuando el cursor llega al filo de la pantalla.*
  - I/D=1. *Incremento automático de la posición del cursor. La posición de la DDRAM se incrementa automáticamente tras cada lectura o escritura de la misma.*
  - I/D=0. *Decremento de la posición del cursor. Se decrementa el puntero de la DDRAM.*
- **Display Control (0000 1 D C B).** *Control de pantalla:*
  - B=0. *Blink OFF, no hay efecto de parpadeo en el cursor.*
  - B=1. *Blink ON, efecto de parpadeo con un cursor rectangular.*
  - C=0. *Cursor OFF, el cursor no se visualiza.*
  - C=1. *Cursor ON, el cursor es visualizado.*
  - D=0. Display OFF, el display se apaga.
  - D=1. *Display ON, el display se enciende.*
- **Cursor and Display Shift (0001 S/C R/L xx).** *Control de los desplazamientos del cursor y de la pantalla:*
  - R/L=0. *Left. a la izquierda.*
  - R/L=1. *Right. a la derecha.*
  - S/C=0. *El efecto de desplazamiento se aplica solo sobre el cursor sin alterar el contenido de la DDRAM.*
  - S/C=1. *El efecto de desplazamiento se aplica sobre todo el display.*
- **Function Set (001DL N F x x).** *Características de control de hardware:*
  - F=0. *Font. caracteres de 5x7.*
  - F=1. *Font. Caracteres de 5x10.*
  - N=0. *Number Line. Pantalla de 1 Línea.*
  - N=1. *Number Line. Pantalla de 2 Líneas.*
  - DL=0. *Data Length. Comunicación con 4 bits. Indica al Display LCD que solamente se van a utilizar las líneas DB7, DB6, DB5 y DB4 para enviarle los datos y que se hará enviando primero el nibble alto y después el nibble bajo del dato.*
  - *DL=1. Data length. Comunicación con 8 bits.*
- **Set CGRAM Address.** *Se va a escribir sobre la dirección CGRAM señalada.*
- **Set DDRAM Address (1ddd dddd).** *Esta instrucción se utiliza para modificar el puntero de la DDRAM. Así por ejemplo, si la dirección es la 08h se escribirá en el centro de la primera línea.*
- **Read Busy Flag.** *Lee el BF (bit bandera) indicando si hay una operación interna en curso y lee, además, el contenido de la dirección DDRAM apuntada.*

El juego de instrucciones se puede resumir en la tabla (2.3 – 1). Para que estos modos de control funcionen, deben ser ingresados en secuencia, y en determinado tiempo, especialmente el ciclo de Enable. Ver la figura (2.3 – 5) y la tabla (2.3 – 2), para el modo escritura. Para leer el display, ver la figura (2.3 – 6) y la tabla (2.3 – 3).

El display LCD requiere de tiempo, después de recibir una instrucción. En ese momento, el display no puede aceptar ninguna más, entonces a esperar (1.64ms en modo comando o 40us en modo escritura), o leer el bit busy flag (bandera de ocupado). En la figura (2.3 – 7), muestra que, el bit busy flag se activa enviando ciclos enable mientras el display este trabajando.

MODOS	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
COMANDO										
Clear Display	0	0	0	0	0	0	0	0	0	1
Return home	0	0	0	0	0	0	0	0	1	x
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display Control	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	x	x
Function Set	0	0	0	0	1	DL	N	F	x	x
Set CGRAM Address	0	0	0	1	CGRAM Address					
Set DDRAM Address	0	0	1	DDRAM Address						
BUSY										
Read Busy Flag	0	1	BF	DDRAM Address						
ESCRITURA										
Write RAM	1	0	Write Data							
LECTURA										
Read RAM	1	1	Read Data							

Tabla (2.3 - 1) Instrucciones del Display LCD.

DIAGRAMAS DE TIEMPOS

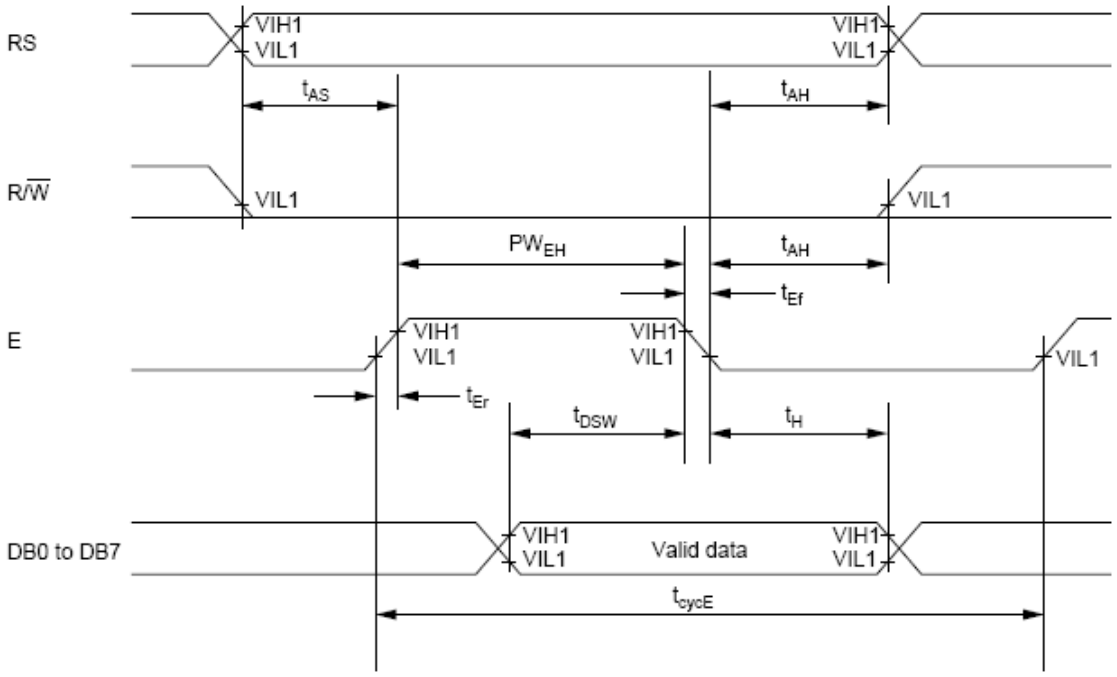
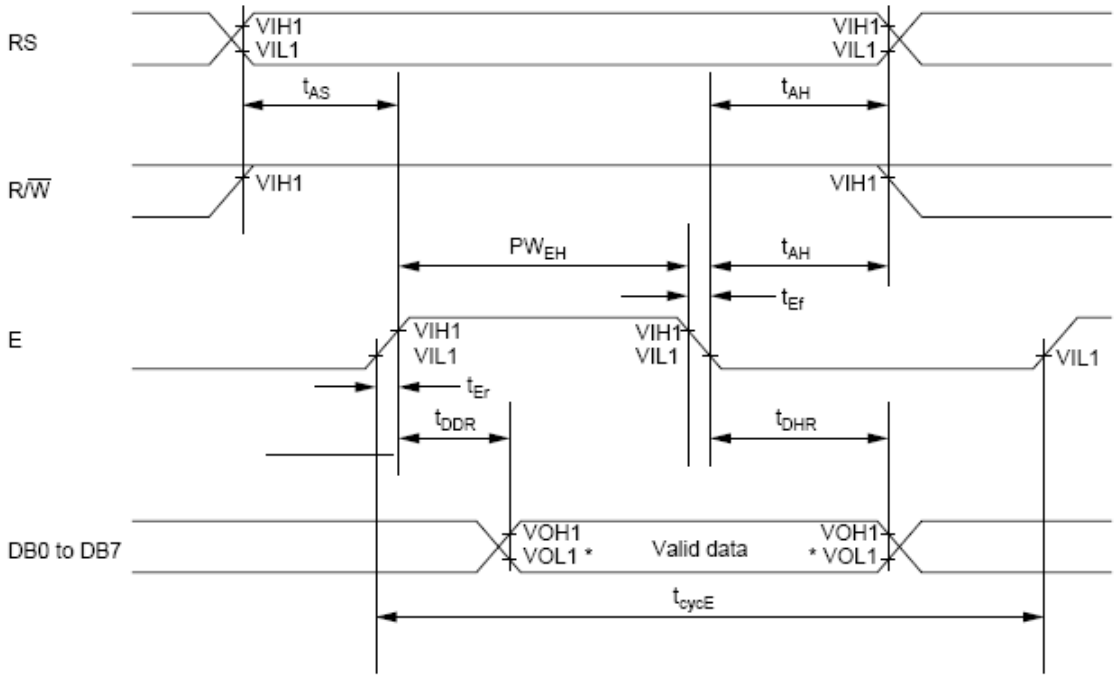


Figura (2.3 – 5) Operación Escritura.

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data set-up time	t_{DSW}	80	—	—	
Data hold time	t_{H}	10	—	—	

Tabla (2.3 – 2) Operación Escritura.



Note: * VOL1 is assumed to be 0.8 V at 2 MHz operation.

Figura (2.3 – 6) Operación Lectura.

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data delay time	t_{DDR}	—	—	160	
Data hold time	t_{DHR}	5	—	—	

Tabla (2.3 – 3) Operación Lectura.

INTERFACE DE 8 bit con busy bit

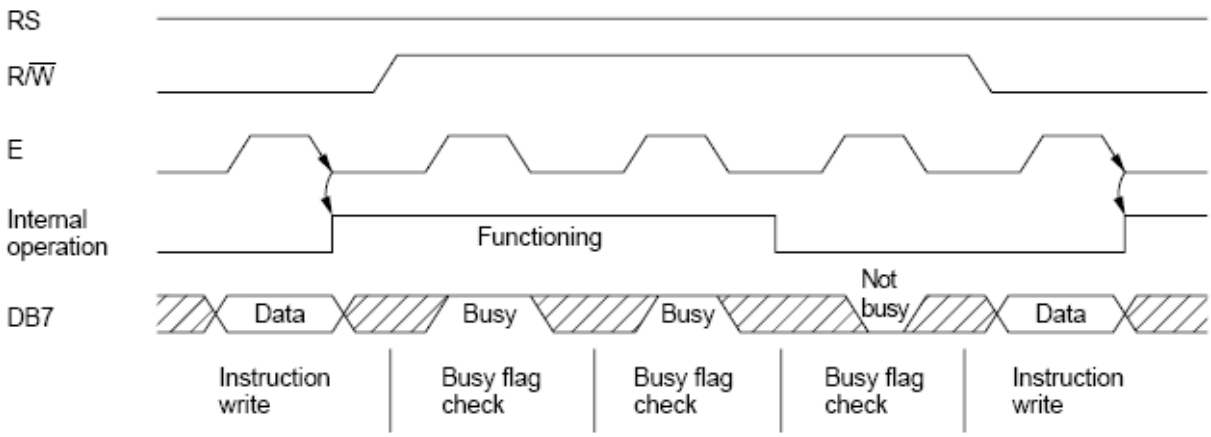


Figura (2.3 – 7) Operación Busy Flag.

2.4. CARACTERÍSTICAS DEL CONVERTIDOR A/D MCP3550/60

Actualmente existen muchos tipos de convertidores A/D con varias ventajas y desventajas entre unos y otros, incluso la versión anterior de este convertidor, el MCP3204, es un convertidor más rápido y fácil de controlar. Pero las características principales que hacen a este modelo mejor, es que tiene dos modos de operación ideales por si se desea tener ahorro de energía, también al ser de 22 bits, su resolución es mayor lo que lo convierte un dispositivo más preciso. También al contar con un filtro digital ($SINC^4$), hace al dispositivo más estable en cuanto a su operación, ya que mantiene su estabilidad y exactitud a diferentes temperaturas.

El manual del MCP3550-60 muestra sus características técnicas que están en el Apendice C:

DESCRIPCIÓN

El dispositivo MCP3550/1 consume de 2.5V a 5.0V, con un Convertidor Analógico-a-Digital (ADC) Delta-Sigma de 22-Bit. El dispositivo ofrece una salida con un ruido reducido a 2.5VRMS con un error total sin ajustar de 10ppm (partes por millón). Esta familia muestra 6ppm de error INL (No-Linealidad Integra), compensa un error de 3V y error Total de-Escala menor de 2ppm. El dispositivo MCP3550 proporciona una alta precisión y presenta un bajo ruido para aplicaciones donde el sensor realiza mediciones (Presión, Temperatura, Humedad). Con el Oscilador Interno y alta velocidad de muestreo, en una aplicación de Alta-Precisión, se requiere un mínimo de componentes.

Esta línea de este producto tiene todas sus entradas analógicas diferenciales, haciéndolo compatible con una gran variedad de sensores, control industrial o aplicaciones de procesos de control.

El dispositivo MCP3550 opera en un rango de -40°C a +125°C y está disponible en un espacio-reducido MSOP (Micro Small Outline Package) y paquetes SOIC (Small Outline Integrated Circuit). El diagrama de pines se encuentra en el Apéndice B en la figura (2.4 – 2) y la descripción de pines en la tabla (2.4 – 2) del mismo Apéndice.

VOLTAJE DE REFERENCIA (Vref) – VOLTAGE REFERENCE

El dispositivo MCP3550 acepta un voltaje de referencia desde 0.1V a VDD. Durante la conversión, el ruido en el resultado es producido por un ruido termal, que es independiente del voltaje de referencia. La generación de ruido no mejora significativamente por disminuir el voltaje de referencia VREF en el pin de entrada. Un voltaje de referencia reducido, mejorará significativamente la presentación de INL (No-Linealidad Integra); El INL de error máximo es proporcional a $VREF^2$.

ENTRADAS ANALÓGICAS (V_{IN}+, V_{IN}-) – ANALOG INPUTS

EL dispositivo MCP3550 tiene una entrada diferencial analógica en los pines VIN+ y VIN-. El voltaje diferencial que se convierte, esta definido por VIN = V_{IN}+ - V_{IN}- . El rango de voltaje diferencial, especificado para asegurar la precisión es de: –VREF a +VREF. Sin embargo, la conversión detendrá la producción valida y utilizable de códigos, con las entradas un un rango total por arriba del 12%. Este rango total es especificado claramente por dos bits de desbordamiento en el código de salida.

El rango de voltaje en estos pin de entrada abarca desde VSS -0.3V a VDD +0.3V. Cualquier voltaje sobre o debajo de este rango creará, a través de las corrientes de disparo, una Descarga Electrostática de Diodos (ESD). Esta corriente aumentará exponencialmente, degradando la precisión y presentando ruido en el dispositivo. El modo común de las entradas analógicas debe conectarse de tal manera que ambas entradas analógicas de rango diferencial y el rango absoluto de voltaje esté dentro del rango de operación definido, que especifica en la sección de características eléctricas.

VOLTAJE DE CONSUMO ($V_{DD,}$ V_{SS}) – SUPPLY VOLTAGE

VDD es el pin al que se conecta el suministro de poder para la circuitería analógica y digital del MCP3550. Este pin requiere un capacitor bypass de 0.1µF. El voltaje en este pin debe mantenerse en el rango de 2.7 a 5.0V para el funcionamiento especificado. VSS es el pin de tierra y la corriente de retorno para la, circuitería del MCP3550. Si un plano analógico de tierra esta disponible, se recomienda que este dispositivo se conecte al plano de tierra analógica de la Tabla de Circuito Impresa (PCB).

RELOJ DE SINCRONIZACIÓN (SCK) – SERIAL CLOCK

SCK Sincroniza la comunicación de datos con el dispositivo. El dispositivo opera ambos modos de SPI (Serial Peripheral Interface), modo 1,1 y modo 0,0. Los datos se intercambian fuera del dispositivo en flanco de bajada de SCK. El dato es tomado en flanco de subida del SCK. Durante un tiempo CS (Chip Select) alto, el pin SCK puede funcionar en los dos estados, alto o bajo.

SALIDA DE DATOS (SDO/RDY) – DATA OUTPUT

SDO/RDY es el pin de salida de datos para el dispositivo. Una vez que la conversión esta completa, este pin estará activo-bajo, actuando como una bandera lista. Los subsecuentes pulsos de reloj pondrán entonces los 24-bits del tamaño de palabra. (2 bits de desbordamiento y 22-bit de datos) en el bus de SPI por medio del pin SDO. El dato es sincronizado fuera del flanco de bajada del SCK.

SELECCION DEL CHIP (CS) - CHIP SELECT

CS es el control de toda la comunicación del dispositivo, y puede usarse para seleccionar múltiples dispositivos que comparten los mismos pin SCK y SDO/RDY. Este pin también controla conversiones internas, que comienzan en el flanco de bajada de CS. Con CS en estado alto, antes de la primera conversión interna el dispositivo entra en modo single conversión (conversión única). CS puede estar permanentemente en bajo para la operación de modo conversión continua (en modo conversión continua, conversiónes seguidas ocurrirán). SDO/SDY entra en estado de alta impedancia con CS en alto.

EL DISPOSITIVO EN GENERAL

El dispositivo MCP3550 22-bit delta-sigma ADC incluye entradas totalmente analógicas, un modulador delta-sigma de tercer-orden, un filtro modificado SINC de cuarto orden, oscilador interno de bajo-ruido, un circuito que monitorea la fuente de suministro y una de interface digital de tres lineas. Este dispositivo puede ser usado para medir a baja frecuencia, señales de nivel bajo como los que se encuentran en los transductores de presión, temperatura, de tensión, control industrial o procesos con aplicaciones de control. El rango de suministro de tensión es de 2.5V a 5.0V y el rango de temperatura de -40°C a 125°C. Un encendido reset (POR: Power On Reset) que controla el circuito para asegurar el suministro de voltaje apropiado durante el proceso de conversión. La fuente de reloj es internamente generada a $\pm$ 5% alrededor del rango de voltaje total de la fuente de alimentación y el rango de temperatura industrial.

INTERFACE SERIAL SPI

La comunicación serial entre el microcontrolador y el dispositivo MCP3550 se logra usando los pines CS, SCK y SDO/RDY. Hay dos modos de operación: Single conversión (conversión única) y Continuous Conversion (Conversión Continua). El CS controla la conversión inicial. Allí son 24 bits de tamaño de palabra: 22 bits de conversión de datos y dos bits de desbordamiento. El proceso de conversión tiene lugar vía el oscilador interno y el estado de esta conversión debe ser detectado. El método típico de conversión es mostrado en la Figura (2.4 -3). El estado de la conversión interna es dado por el pin SDO/RDY, disponible cuando CS esta en bajo. Hay un estado alto en el pin SDO/RDY mientras el dispositivo este ocupado convirtiendo, y habrá un estado bajo en el mismo pin, cuando el dato está listo para una transferencia, usando SCK. El SDO/RDY permanece en estado de alta impedancia cuando CS está en estado alto. El CS debe ser bajo cuando se toma el dato usando SCK and SDO/RDY.

El bit 22 es de desbordamiento Alto (OVH) cuando VIN > VREF -1 LSB, OVH alterna a un “1” lógico, detectando un desbordamiento alto en la entrada analógica de voltaje.

El bit 23 es de desbordamiento Bajo (OVL) cuando VIN > -VREF, OVL alterna a un “1” lógico, detectando un desbordamiento bajo en la entrada analógica de voltaje. El estado OVH = OVL = ”1” no está definido y debe ser considerado como una interrupción para la interface SPI significando una comunicación errónea.

Del bit 21 al bit 0 representa el código de salida de 22-bit en complemento a dos binario. El bit 21 es el bit de signo y es un “0” lógico cuando la entrada análoga diferencial es positiva y un “1” lógico cuando la entrada análoga diferencial es negativa. Del bit 20 al bit 0, del código de salida, es primero el MSB (More Significant Bit - bit Más Significativo) que es el bit 20 y LSB (Less Significant Bit - Bit Menos Significativo) que es el bit 0. Cuando el valor de la entrada análoga se comprende entre –VREF y VREF -1 LSB, los dos bits de desbordamiento

se ponen a “0” lógico. La relación entre el voltaje de entrada y el código de salida es mostrada en la figura (2.4 – 3).

El punto de saturación del modulador Delta-Sigma para la entrada analógica diferencial se localiza alrededor del 112% de VREF (a temperatura ambiente), significa que el modulador todavía entregará el código de salida exacto con un desbordamiento del 12% debajo o sobre el voltaje de referencia. A diferencia de los habituales dispositivos de 22-bit, el código de salida de 22-bit no cerrara a 0x1FFFFF para una señal de entrada positiva o 0x200000 para una señal de entrada negativa, se toma ventaja de las capacidades de desbordamiento del dispositivo. Esto puede ser práctico para un caso de operaciones con bucle cerrado. Si ocurre un desbordamiento, el código de salida comienza con un bit-23 en complemento a dos, donde la señal del bit será el bit OVL (bit-23). La figura (2.4 – 3) resume el formato de código de datos con el desbordamiento alto y bajo.

Analog Input Voltage	OVL	OVH	Digital Output Code																				Decimal Code	Hexa
	B23	B22	B21	B20	B19	B18	B17	B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0		*B[23:0]*
Vref +1 LSB	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2097153	600001
Vref	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2097152	600000
Vref - 1 LSB	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2097151	1FFFFF
2 LSB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	000002
1 LSB	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	000001
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	000000
-1 LSB	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	3FFFFF
-2 LSB	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	-2	3FFFFE
-Vref	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2097152	200000
-Vref -1LSB	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-2097153	9FFFFF
-Vref -2LSB	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	-2097154	9FFFFE

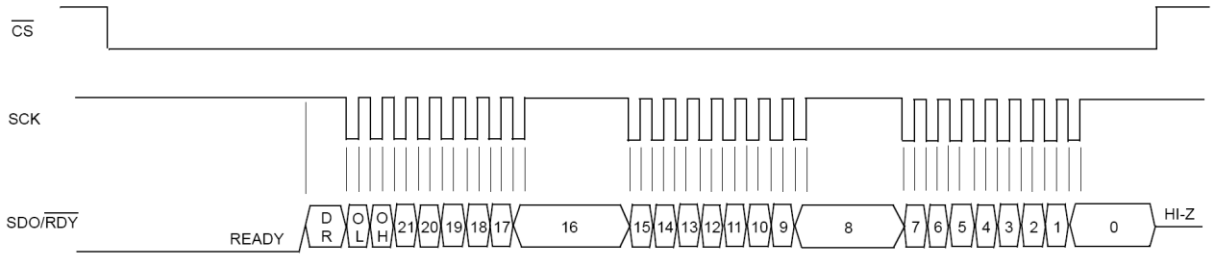


FIGURA (2.4 -3) Comunicación SPI y el Código de Salida Digital para Voltajes Analógicos de Entrada Específicos.

CONTROLANDO CONVERSIONES INTERNAS Y EL OSCILADOR INTERNO

Durante el modo shutdown (consumo bajo de energía), con un flanco de bajada de CS, comienza el proceso de conversión. Durante este proceso, el reloj oscilador interno, el modulador Delta-Sigma y el filtro SINC trabajan hasta que una conversión se completa. Este tiempo de conversión es Tconv y el diagrama de tiempo es mostrado en la Figura (2.4 – 4). Al final del tiempo Tconv, el filtro digital SINC se ajusta completamente y no está involucrada ninguna latencia en el MCP3550.

Los dos modos de conversión para el dispositivo MCP3550 son: el Single Conversion (conversión única) y el Continuous Conversion (conversión continua). En el modo Single Conversion una conversión consecutiva no comenzará automáticamente. En cambio, después de que una single conversion se completa y los cuatro filtros están fijos, el dispositivo pone los datos en el registro de salida y entra en modo shutdown.

En el modo conversión continua, automáticamente ocurrirán conversiones consecutivas. La conversión más reciente es la que estará en el registro de salida. Cuando el dispositivo entra en modo shutdown (consumo bajo de energía), hay un retraso power-up (consumo alto de energía, tras estar en modo shutdown), que debe observarse. Ovservar las figuras (2.4 – 4) y (2.4 – 5).

SINGLE CONVERSION MODE

Si un flanco de subida del Chip Select (CS) ocurre durante e Tconv, una conversión subsecuente no tendrá lugar y el dispositivo entrara en modo shutdown (consumo bajo de energía) después del Tconv completo. Esto es llamado modo Single Conversion. Esta operación es mostrada en la Figura (2.4 – 4). Note que un flanco bajo de CS durante la misma conversión detecta un flanco alto como en la figura (2.4 – 5), pero no comenzará una nueva conversión. Una vez que el dispositivo se halla puesto en modo Single Conversion, los datos deben ser sincronizados de fuera para que una nueva conversión tenga lugar. Un subsecuente flanco de bajada en CS durante Shutdown mode no iniciará una nueva conversión, a menos que los primeros datos de conversión hayan sido sincronizados fuera del dispositivo. Después de que el bit final ha sido sincronizado en los 25 pulsos de reloj, el pin SDO/RDY se pondrá activo alto.

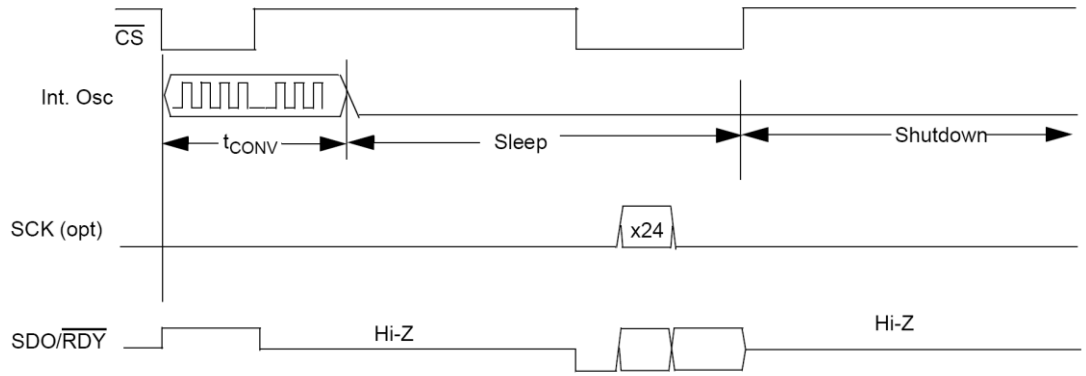


FIGURA (2.4 – 4) Single Conversion Mode.

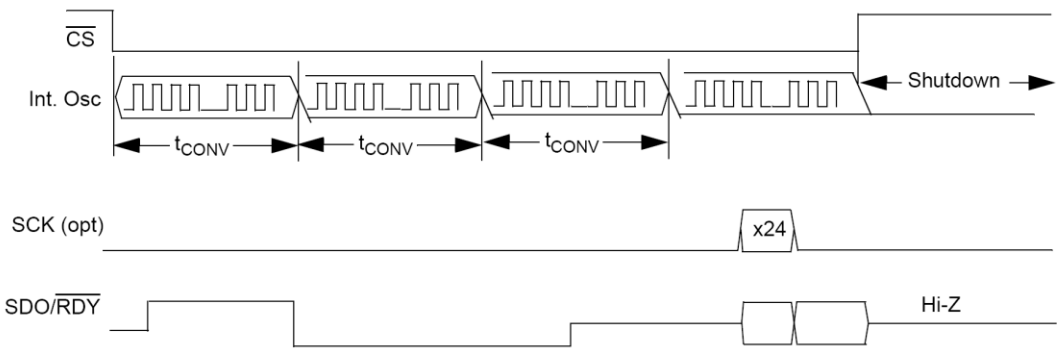


FIGURA (2.4 – 5) Continuous Conversion Mode.

FUNCION READY DEL PIN SDO/RDY EN MODO SINGLE CONVERSION

En cada flanco de bajada de CS durante la conversión interna, el estado de la conversión interna es tomado con el pin SDO/RDY para dar información de estar listo u ocupado. Un estado alto de que el dispositivo está realizando actualmente una conversión interna y los datos no pueden ser sacados con los pulsos de reloj. Un estado medio bajo de que el dispositivo a terminado su conversión y los datos están listos para la recuperación en flanco bajo de SCK. Esta operación es demostrada en la figura (2.4 – 6). Observá que el dispositivo se ha puesto en modo Single Conversión con el primer flanco alto de CS.

Nota: El estado ready es tomado en cada flanco bajo de CS y no actualizará si CS se mantiene bajo. CS debe ser alternado de alto a bajo.

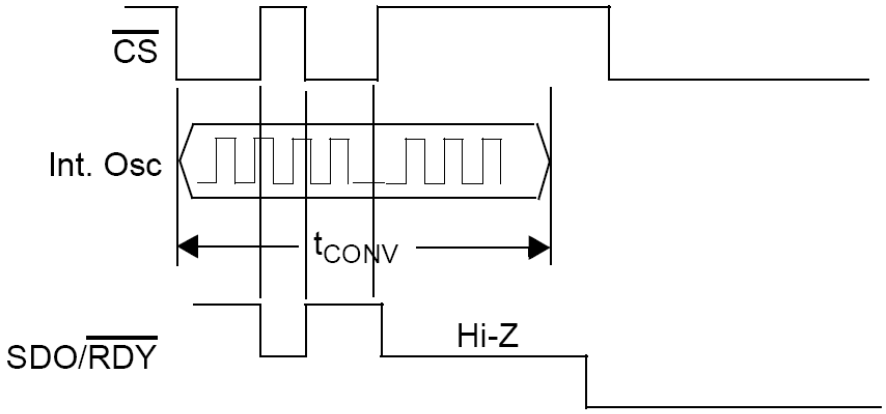


FIGURA (2.4 – 6) RDY Funcionando en Modo Single Conversion.

MODO CONTINUOUS CONVERSION

Si ningún flanco de subida de CS ocurre durante cualquier conversión dada como en la Figura (2.4 – 4), una conversión subsecuente tomará lugar y el contenido de la conversión anterior será sobrescrita. Esta operación es mostrada en la Figura (2.4 – 7). Una vez que la conversión de datos de salida a iniciado para ser sincronizado, el buffer de salida no se refresca hasta que los 24 bit son sincronizados (extraídos). Una lectura completa debe ocurrir para leer la siguiente conversión en este modo. Los datos de conversión subsecuente para ser leídos, solo con la más reciente conversión. El tiempo de conversión es arreglado y no puede ser acortado por el flanco alto de CS. Este flanco alto pondrá en parte el modo shutdown y todos los datos de la conversión se perderán.

La transferencia de datos del Filtro SINC para el buffer de salida es mostrado en la Figura (2.4 – 7). Si la conversión anterior de datos no es sincronizada fuera del dispositivo, se perderá y remplazará por una nueva conversión. Cuando el dispositivo está en modo de Conversión Continua, la conversión de datos más reciente siempre se presenta en el registro de salida para la recuperación de datos.

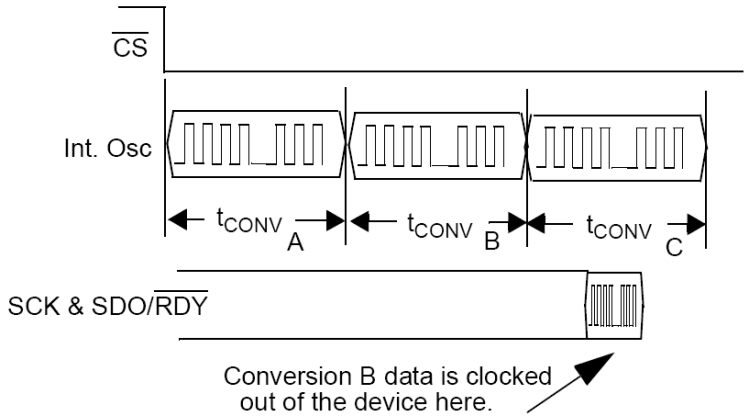


FIGURA (2.4 – 7) Modo Conversión Continua más actual de datos.

Si una conversión está en proceso, no puede terminarse con el flanco alto de CS. SDO/RDY primero debe transitar a un estado bajo, que indicará el fin de la conversión.

USANDO EL MCP3550 CON MICROCONTROLADOR (MCU) PUERTOS SPI

Se requiere que los puertos para SPI del microcontrolador sean configurados para salida de datos con pulso de reloj en flanco bajo y tomados en flanco de subida. Figura (2.4 – 8) dicta la operación mostrada en modo SPI 1,1, que requiere al SCK del Microcontrolador sin hacer nada en estado alto, mientras la Figura (2.4 – 9) muestra un caso similar del modo SPI 0,0 donde el reloj esta ocioso en estado bajo. La forma de onda en la figura son ejemplos de un Microcontrolador operando el puerto SPI en modo de 8-bit, y el dispositivo MCP3550 no requiere datos en grupos de 8-bit.

En modo SPI 1,1, el dato es leído usando solamente 24 pulsos de reloj para transferir 3 bytes. El bit de dato convertido debe leerse probando la línea SDO/RDY antes de un flanco bajo del reloj.

En el modo SPI 0,0, el dato se lee usando 24 pulsos de reloj para transferir cuatro bytes. Por favor note que los bits son incluidos en la transferencia como el primer bit en este modo.

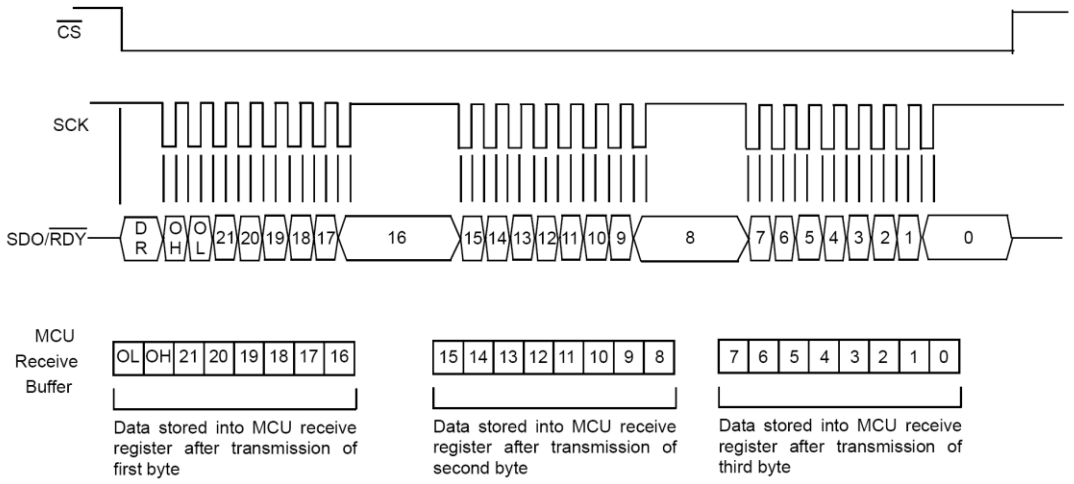


FIGURA (2.4 – 8) Comunicación SPI – Modo 1,1.

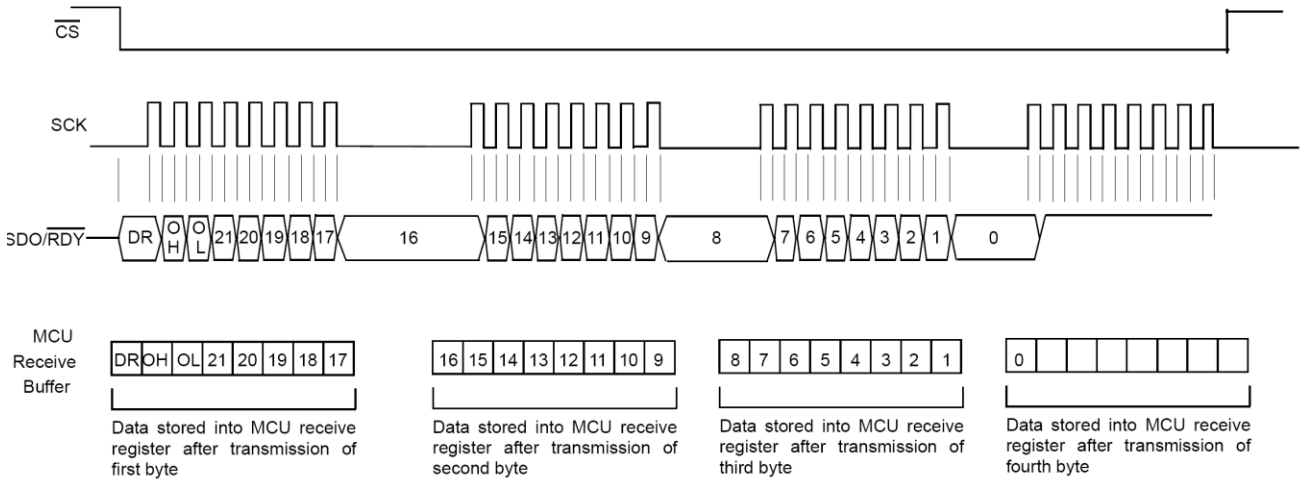


FIGURA (2.4 – 9) Comunicación SPI – Modo 0,0.

2.5. ELABORACIÓN DE CÓDIGOS

Al construir un proyecto, en donde ya se ha definido las características del hardware y se sabe, de forma general, el tipo de solución que se le va a dar, es porque ya se tiene bien definidas las partes ó módulos que conforman la función. Esto conlleva a que se debe saber el funcionamiento preciso de cada etapa o, cuando menos, nociones básicas, como es el caso del módulo USB.

Aunque no se domine el total funcionamiento de alguna etapa, se puede superar, ya que si se maneja una etapa como una caja negra, no significa que no se pueda tener los resultados esperados o no cumplir con el objetivo.

La información que contiene el libro USB COMPLETE, es muy amplia y habla del protocolo USB en general, por lo que el estudio de esta etapa es difícil. En internet, tampoco hay algo semejante a un código elaborado en ensamblador para el manejo del USB. Por esta razón se recurre al compilador PIC C, el cual contiene librerías para el control de dicho módulo.

Si bien queda claro que el PIC C es lo suficientemente práctico, poderoso, y su uso es relativamente sencillo, existen algunas desventajas que hacen de este, un poco inflexible en muchos casos. Una de las razones principales en las que PIC C resulta no favorable es que, como sabemos, PIC C es una versión de compilador de lenguaje de “alto nivel” en C. Esto implica que cuando nosotros, por ejemplo, escribimos operaciones matemáticas, el compilador genera las instrucciones en código ensamblador, esto quiere decir que nos ahorra trabajo, hacerlo en ensamblador puede resultar muy complicado, dependiendo del tipo de operación que se trate. La contraparte es que genera código extra, esto implica que la ejecución del código no resulte del todo eficiente, pues un programa más grande haría que la ejecución de una operación fuera más lenta y, por último, este código al ser más grande, implica que ocupa más espacio en la memoria de programa. El otro problema es que este compilador no puede simular las instrucciones, como en el caso de MPLAB IDE. Existe una versión de compilador C que permite simular dentro del entorno del MPLAB IDE, pero requiere de licencia. La ventaja con la que contamos con el compilador PIC C es que puede ser perfectamente compatible con el ensamblador del MPLAB IDE, si sabemos cómo.

Por estas razones se ha optado por no olvidar al código ensamblador, además de que será una buena referencia para aquellos que no saben cómo insertar código ensamblador en el PIC C, ya que en la red y en otros medios no viene suficiente documentación al respecto. En resumen, se utilizará lenguaje ensamblador para todo, excepto para el módulo USB. La ventaja que posee ensamblador, respecto al lenguaje de alto nivel, es que podemos controlar cada uno de los registros del microcontrolador y que podemos simular virtualmente su funcionamiento ahorrándonos mucho tiempo con pruebas físicas innecesarias, ya que algunos simuladores, como el PROTEUS o el PIC SIM, no siempre son confiables y también, a diferencia de las librerías de C, en donde ya vienen configurados los puertos para usar un dispositivo haciendo un poco inflexible el armado físico de nuestro dispositivo, como sucede con el módulo LCD, con ensamblador nos podemos dar gusto ajustando los pines como deseemos.

2.5.1. SUBRUTINAS DE RETARDO

Este proyecto tiene la necesidad de controlar de manera muy precisa los tiempos de ejecución de cada una de las operaciones, ya sea para el convertidor A/D, o el módulo LCD. Para esto, debemos definir la frecuencia a la que vamos a operar. Para esto se debe hacer un estudio consciente del oscilador que, cuenta con una arquitectura flexible. Esto quiere decir que puede operar a muy variadas frecuencias, dependiendo del tipo de configuración que se tenga.

Se propone un cristal de 12MHz. Con este dato, según tabla (2.5 – 1) proporcionada por el manual de microchip, nos dice que se trata de un oscilador tipo HS (High Speed).

La figura (2.5 – 1) muestra el tipo de conexión para oscilador externo. Esta configuración corresponde a los tipos XT, HS, o HSPLL y XTPLL.

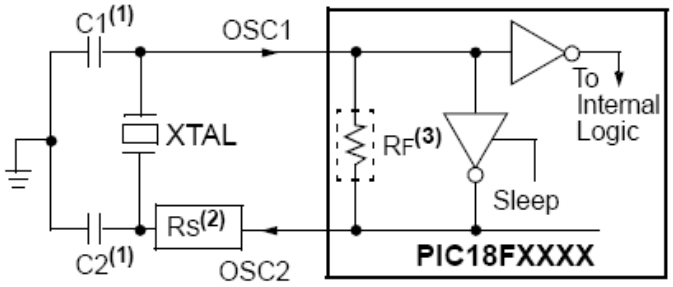


Figura (2.5 – 1) Conexión para oscilador XT, HS, o HSPLL y XTPLL.

Esta configuración es una de las más utilizadas, ya que la frecuencia de oscilación es la más estable con respecto de la configuración de oscilador interno.

El fabricante dice que puede requerir distintos valores del capacitor para producir un funcionamiento del oscilador aceptable. Ver la tabla (2.5 – 1). Se puede hacer una interpolación para estimar el valor que le corresponde al cristal de 12MHz, pero en la mayoría de los casos no está disponible ese valor en las tiendas, así que se propone 22pF que es un valor perfectamente comercial.

Tipo de Oscilador	Frecuencia del Cristal	Capacitores típicos: Valores probados	
		C1	C2
XT	4Mz	27pF	27pF
HS	4Mz	27pF	27pF
	8Mz	22pF	22pF
	20Mz	15pF	15pF

Tabla (2.5 – 1) Selección de capacitores para el oscilador de cristal.

Esta etapa es muy importante porque se define como se va a trabajar con el PIC. Ahora se es consiente de que los pines OSC1 y OSC2 se utilizarán para este propósito. También se utilizará el periférico USB, y consultando el manual según las opciones para operar con USB, todo se reduce a las siguientes opciones:

Frecuencia de entrada	División PLL (PLLDIV2:PLLDIV0)	Modo del reloj (FOSC3:FOSC0)	División del reloj MCU (CPUDIV1:CPUDIV0)	Frecuencia del microcontrolador
12MHz	÷3 (010)	HS,EC,ECIO	Ninguno (00)	12MHz
			÷2 (01)	6MHz
			÷3 (10)	4MHz
			÷4 (11)	3MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48MHz
			÷3 (01)	32MHz
			÷4 (10)	24MHz
			÷6 (11)	16MHz

Tabla (2.5 – 2) Opciones para la configuración del oscilador para operar con el USB.

Con esta tabla, intuitivamente, sabemos las opciones que necesitamos para la configuración del oscilador operando con el USB. Para entender lo que se esta haciendo, lo más recomendable es estudiar un poco el diagrama del reloj del PIC en la figura (2.5 - 2).

En la entrada del reloj de oscilación primario, tenemos un divisor de frecuencia PLL Prescaler, en donde la frecuencia es dividida por 1, 2, 3, 4, 5, 6, 10 y 12. Ahora con el MUX, usando PLLDIV, se escoge cual es la frecuencia a utilizar. Dependiendo de la frecuencia de nuestro cristal se elige la división cuyo resultado sea 4MHz que es la frecuencia que se debe tener a la salida del MUX.

Después del MUX, está un multiplicador de frecuencia que, con los 4 MHz inyectados, este nos generará 96MHz a su salida. Esta frecuencia de 96 MHz es reducida a la mitad (48MHz), para que pueda ser utilizada por el módulo USB. Para esto, debe configurar con un 1 a USBDIV que se debe usar la señal proveniente del PLL.

Según se observa en el diagrama de la figura (2.5 – 2), la señal de 96MHz también se conecta al PLL Postscaler, que es otro divisor que divide entre 2, 3, 4, 6. Estas señales van a otro MUX, usando el CPU DIV, se debe escoger una frecuencia para el PIC, estas frecuencias pueden ser 48MHz, 32MHz, 24MHz y 16MHz.

Para este proyecto utilizaremos 16MHz, ya que para trabajar con los dispositivos LCD y MCP3550 no es necesaria una frecuencia más alta. Una frecuencia más alta implicaría una evaluación cuidadosa en los diagramas de tiempo proporcionados por los fabricantes de los dispositivos MCP y LCD. Esto a razón de que aunque pongamos retardos, no significa que operen correctamente, ya que, tanto los pulsos o los flancos que requieren estos dispositivos que conectaremos, requieren de un tiempo mínimo, y una frecuencia demasiado alta, podría complicarlo todo.

Hay dos CPUDIV y el que utilizamos se selecciona con el switch FOSC3:FOSC0, que es de donde se obtiene la frecuencia en la ejecución de los programas.

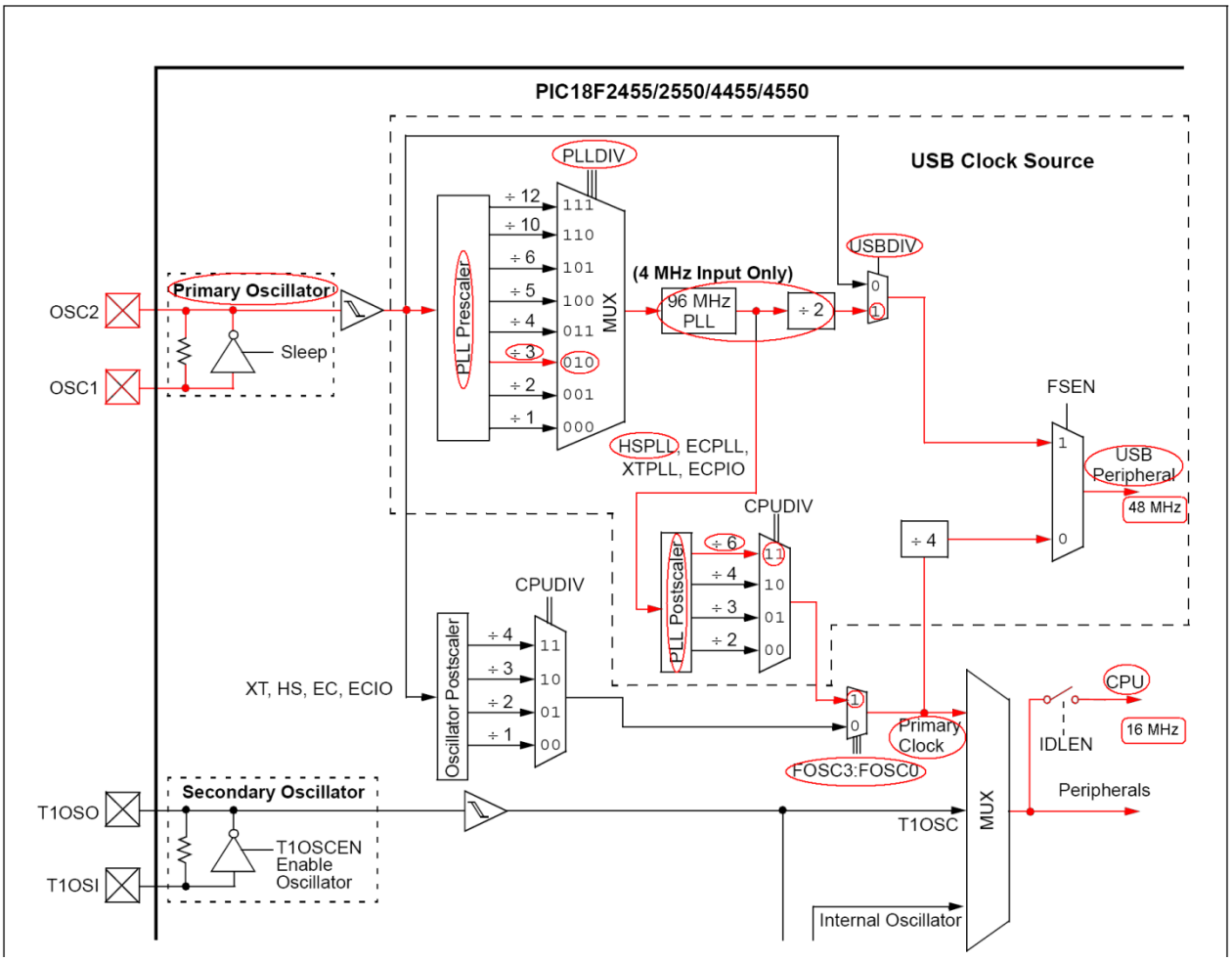


Figura (2.5 – 2) Diagrama a bloques del oscilador interno del microcontrolador.

Una vez decidido que el microcontrolador trabajará a una frecuencia de 16MHz, independiente a la del módulo USB de 48MHz, se procede a hacer el cálculo siguiente. Tomando como base la fórmula del ciclo máquina expuesta en el capítulo anterior, se calcula cuánto dura un ciclo máquina.

$$Tiempo[s] = 4\left(\frac{1}{16MHz}\right) = 4(0.0625 \times 10^{-6}) = 0.25 \times 10^{-6} = 0.25\mu s$$

$$\therefore 1cm = 0.25\mu s$$

Es muy importante crear el pseudocódigo y, aunque parece no tener relación con el ensamblador y solamente con C, se ve con este ejemplo, que se puede trabajar perfectamente con ambos.

Ejemplo de pseudocódigo para la creación de subrutinas de retardo:

```
;_____________________________________________________________________
CONTENEDORES
Conta_K
FIN CONTENEDORES

RETARDO_X𝜇𝑠                              ;Estructura1
        CARGAR          conta_K
        BUCLEBASICO
REGRESAR

RETARDO_Y𝜇𝑠                              ;Estructura2
        CARGAR          conta_K
        BUCLEBASICO
REGRESAR

BUCLEBASICO                             ;Estructura3
        HAZ
                conta_K = conta_K-1
        MIENTRAS  (conta_K = 0?)
REGRESAR
;_____________________________________________________________________
```

Los contadores son la base para hacer los retardos. Obsérvese como cada estructura es una subrutina, y cada una tiene su propia etiqueta con su nombre. Las primeras dos estructuras sirven solo para inicializar la variable conta_K, y la estructura 3 ejecuta el proceso con la variable. De esta forma podemos tener varios retardos, dependiendo de la constante que se cargue en conta_K. Este diseño tiene límites, ya que con una variable de 8 bits, solo se permite valores de 0 a 255, por lo tanto nos queda que el intervalo será $0 \leq conta\_K \leq 255$.

Por último, el código fuente en ensamblador queda como sigue:

```
;_____________________________________________________________________
;*************************************************
;UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
;FACULTAD DE ESTUDIOS SUPERIORES
;"ARAGÓN"
;
;INGENIERÍA MECÁNICA ELÉCTRICA
;
;NOMBRE
;Subrutinas de retardo para procesador de 16MHz
;
;DESCRIPCIÓN
;Con un procesador funcionando a 16MHz, un cm dura 0.25us
;para el modelo PIC16F4550.
;Este programa contiene una rutina para poder simular
;los retardos creados
;*************************************************
```

```
;CABECERA
    #INCLUDE <P18F4550.INC>

;CONTENEDORES
        CBLOCK  0x00     ;Dirección propuesta
        CONTA_K
        CONTA_L
        ENDC

;SIMULADOR
        call    retardo_1us
        call    retardo_2us
        call    retardo_3us
        call    retardo_4us
        call    retardo_5us
        call    retardo_10us
        call    retardo_20us
        call    retardo_30us
        call    retardo_40us
        call    retardo_50us
        call    retardo_100us
        sleep

;Estructuras para retardos de 1 a 100 us
;Estas estructuras requieren del  bucle básico
;
;Un retardo de 1us requiere de 4cm
retardo_1us                             ;2cm        LLAMADO
        return                          ;2cm        REGRESAR

;Un retardo de 2us requiere de 8cm
retardo_2us                             ;2cm        LLAMADO
        goto    ret2us1                 ;2cm
ret2us1 goto    ret2us2  ;2cm
ret2us2 return                          ;2cm        REGRESAR

;Para 3us requiere 12cm, entonces :
retardo_3us                             ;2cm        LLAMADO
        goto    ret3us1                 ;2cm
ret3us1 goto    ret3us2                 ;2cm
ret3us2 goto    ret3us3                 ;2cm
ret3us3 goto    ret3us4                 ;2cm
ret3us4 return                          ;2cm        REGRESAR

;Para 4us requiere 16cm, entonces :
;[2+1+1+BB+2] = 6+BB = 6+(4+3K) = 10+3K
;igualando la ecuación con 16cm
;10+3K=16; -> K=(16-10)/3 =2; -> K=2
;con k=2 ->16cm ->4us;
retardo_4us                             ;2cm    LLAMADO
        movlw   .2                      ;1cm    CARGAR K
        movwf   CONTA_K                 ;1cm
        call    BucleBasico             ;(BB)   LLAMAR BB
        return                          ;2cm    REGRESAR
```

```
;Para 5us requiere 20cm
;10+3K=20; -> K=(20-10)/3 =3.3; -> K=3
;con K=3 ->19cm ->4.75us; se compensa .25us con 1cm
retardo_5us                         ;2cm    LLAMADO
        movlw   .3                  ;1cm    CARGAR K
        movwf   CONTA_K             ;1cm
        call    BucleBasico         ;BBcm   LLAMAR BB
        nop                         ;Compensación
        return                      ;2cm            REGRESAR

;Para 10us requiere 40cm
;10+3K=40; -> K=(40-10)/3 =10; -> K=10
;con K=10 ->40cm ->10us;
retardo_10us                        ;2cm    LLAMADO
        movlw   .10                 ;1cm    CARGAR K
        movwf   CONTA_K             ;1cm
        call    BucleBasico         ;BBcm   LLAMAR BB
        return                      ;2cm    REGRESAR

;Para 20us requiere 80cm
;10+3K=80; -> K=(80-10)/3 =23.3; -> K=23
;con K=23 ->79cm ->19.75us; se compensa .25us con 1cm
retardo_20us                        ;2cm    LLAMADO
        movlw   .23                 ;1cm    CARGAR K
        movwf   CONTA_K             ;1cm
        call    BucleBasico         ;BBcm   LLAMAR BB
        nop                         ;compensación
        return                      ;2cm    REGRESAR

;Para 30us requiere 120cm
;10+3K=120; -> K=(120-10)/3 =36.67; -> K=36
;con K=36 -> 118cm -> 29.5us se compensa .5us con 2cm
retardo_30us                        ;2cm            LLAMADO
        movlw   .36                 ;1cm            CARGAR K
        movwf   CONTA_K             ;1cm
        call    BucleBasico         ;BBcm   LLAMAR BB
        nop                         ;compensación
        nop
        return                      ;2cm            REGRESAR

;Para 40us requiere 160cm
;10+3K=160; -> K=(160-10)/3 =50; -> K=50
;con K=50 -> 160cm -> 40us;
retardo_40us                        ;2cm    LLAMADO
        movlw   .50                 ;1cm    CARGAR K
        movwf   CONTA_K             ;1cm
        call    BucleBasico         ;BBcm   LLAMAR BB
        return                      ;2cm    REGRESAR
```

```
;Para 50us requiere 200cm
;10+3K=200; -> K=(200-10)/3 =63.3; -> K=63
;con K=63 -> 199cm -> 49.75us; se compensa .25us con 1cm
retardo_50us                        ;2cm    LLAMADO
        movlw   .63                 ;1cm    CARGAR K
        movwf   CONTA_K             ;1cm
        call    BucleBasico         ;BBcm   LLAMAR BB
        nop                         ;compensación
        return                      ;2cm    REGRESAR

;Para 100us requiere 400cm
;10+3K=400; -> K=(400-10)/3 =130; -> K=130
;con K=130 -> 400cm -> 100us
retardo_100us                       ;2cm    LLAMADO
        movlw   .130                ;1cm    CARGAR K
        movwf   CONTA_K             ;1cm
        call    BucleBasico         ;BBcm   LLAMAR BB
        return                      ;2cm    REGRESAR

;BUCLE BÁSICO
;su representación algebraica es la siguiente:
;[2+(k-1)+3+(k-1)2+2] = 7+(k-1)3 = 4+3K
;BucleBasico = BB = 4+3K
BucleBasico                         ;2cm    LLAMADO
                                    ;HAZ
        decfsz  CONTA_K             ;(k-1)cm+3cm    DECREMENTAR conta_k
        goto    BucleBasico         ;(k-1)2cm       MIENTRAS (conta_k=0?)
        return                      ;2cm            REGRESAR


END
```

;_____

Ahora se puede anidar más bucles para aumentar el tiempo de retardo, el pseudocódigo no cambia mucho. Y con la experiencia anterior, se crea las formulas para calcular el retardo, incluso con el pseudocódigo.

Ejemplo de pseudocódigo para la creación de subrutinas de retardo:

```
;_____________________________________________________________________________
;Para las estructuras 1 y 2, la representación algebraica queda:
;2+2+BA1+2=6+BA1 ->  Retardo_Zms=6+BA1
RETARDO_Zms                         ;2cm para el llamado        ;Estructura1
        CARGAR      L               ;2cm para inicializar conta_L
        BUCLEANIDADO1               ;BA1
REGRESAR                            ;2cm para el return

RETARDO_Nms                         ;2cm para el llamado        ;Estructura2
        CARGAR      L               ;2cm para inicializar conta_L
        BUCLEANIDADO1               ;BA1
REGRESAR                            ;2cm para el return
```

;Para BUCLEANIDADO1 la representación algebraica es la siguiente:
;$2 + 2L + 769L + (L - 1) + 3 + (L - 1)2 + 2$
;$7 + 771L + (L - 1)3 = 4 + 774L$
;BA1=4+774L

```
BUCLEANIDADO1                   ;2cm para el llamado;Estructura3
        HAZ
          Conta_K=255           ;2cm(L) se necesitan 2cm pero esta anidado con L
          BUCLEBASICO           ;BB(L)=(4+3K)(L)=(43+3(255))L=769L
          DECREMENTAR conta_L   ;(L-1)cm +3cm
          MIENTRAS (conta_L=0?) ;(L-1)2cm, este es para el goto
REGRESAR                        ;2cm para el return
```

;_____

Ahora solo se agrega el siguiente código al primer ejemplo. También se puede simular y comprobar que funcione correctamente.

;_____

```
;Las siguientes estructuras requieren del bucle básico y bucleanidado1
;y algunas requieren también de la compensación para que sean exactas
;
;Para 200us requiere 800cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 800cm
;10+774L=800 -> L=(800-10)/774=1.02 -> L=1
;L=1 -> 784cm -> 196us ; se compensa 4us con llamado al retardo4us
retardo_200us           ;2cm                    LLAMADO
        movlw   .1      ;1cm                    CARGAR L
        movwf   CONTA_L         ;1cm
        call    BucleAnidado1   ;BA1            LLAMAR BA1
        call    retardo_4us     ;compensación
        return          ;2cm                    REGRESAR

;Para 300us requiere 1200cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 1200cm
;10+774L=1200 -> L=(1200-10)/774=1.52 -> L=1
;L=1 -> 784cm -> 196us ; se compensa 104us con llamado al retardo4us y 100us
retardo_300us           ;2cm                    LLAMADO
        movlw   .1      ;1cm                    CARGAR L
        movwf   CONTA_L         ;1cm
        call    BucleAnidado1   ;BA1            LLAMAR BA1
        call    retardo_4us     ;compensación
        call    retardo_100us
        return          ;2cm                    REGRESAR

;Para 400us requiere 1600cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 1600cm
;10+774L=1600 -> L=(1600-10)/774=2.05 -> L=2
;L=2 -> 1558cm -> 389.5us ; se compensa 10.5us con llamado al retardo 10us y 2cm
retardo_400us           ;2cm                    LLAMADO
```

```
        movlw   .2          ;1cm                    CARGAR L
        movwf   CONTA_L     ;1cm
        call    BucleAnidado1   ;BA1                LLAMAR BA1
        call    retardo_10us    ;compensación
        nop
        nop
        return              ;2cm                    REGRESAR

;Para 500us requiere 2000cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 2000cm
;10+774L=2000 -> L=(2000-10)/774=2.57 -> L=2
;L=2 -> 1558cm -> 389.5us ; se compensa 110.5us con llamado al retardo 100, 10 us y, 2cm
retardo_500us       ;2cm                    LLAMADO
        movlw   .2          ;1cm                    CARGAR L
        movwf   CONTA_L     ;1cm
        call    BucleAnidado1   ;BA1                LLAMAR BA1
        call    retardo_100us   ;compensación
        call    retardo_10us
        nop
        nop
        return              ;2cm                    REGRESAR

;Para 1000us requiere 4000cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 2000cm
;10+774L=4000 -> L=(4000-10)/774=5.15 -> L=5
;L=5 -> 3880cm -> 970us ; se compensa 30us con llamado al retardo 30us
retardo_1ms         ;2cm                    LLAMADO
        movlw   .5          ;1cm                    CARGAR L
        movwf   CONTA_L     ;1cm
        call    BucleAnidado1   ;BA1                LLAMAR BA1
        call    retardo_30us    ;compensación
        return              ;2cm                    REGRESAR

;Para 2000us requiere 8000cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 8000cm
;10+774L=8000 -> L=(8000-10)/774=10.3 -> L=10
;L10 -> 7750cm -> 1937.5us ;  compensa 62.5us con llamado al retardo 50,10,2us y 2cm
retardo_2ms         ;2cm                    LLAMADO
        movlw   .10         ;1cm                    CARGAR L
        movwf   CONTA_L     ;1cm
        call    BucleAnidado1   ;BA1                LLAMAR BA1
        call    retardo_50us    ;compensación
        call    retardo_10us
        call    retardo_2us
        nop
        nop
        return              ;2cm                    REGRESAR

;Para 3000us requiere 12000cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 2000cm
```

```
;10+774L=12000 -> L=(12000-10)/774=15.49 -> L=15
;L=15 -> 11620cm -> 290s us; se compensa 95us con llamado al retardo 50,40,5
retardo_3ms             ;2cm                    LLAMADO
        movlw   .15     ;1cm                    CARGAR L
        movwf   CONTA_L         ;1cm
        call    BucleAnidado1   ;BA1            LLAMAR BA1
        call    retardo_50us    ;compensación
        call    retardo_40us
        call    retardo_5us
        return          ;2cm                    REGRESAR

;Para 4000us requiere 16000cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 2000cm
;10+774L=16000 -> L=(16000-10)/774=20.65 -> L=20
;L=20 -> 15490cm -> 3872.5us ; se compensa 127.5us con llamado al retardo 100,20,5,2us y 2cm
retardo_4ms             ;2cm                    LLAMADO
        movlw   .20     ;1cm                    CARGAR L
        movwf   CONTA_L         ;1cm
        call    BucleAnidado1   ;BA1            LLAMAR BA1
        call    retardo_100us   ;compensación
        call    retardo_20us
        call    retardo_5us
        call    retardo_2us
        nop
        nop
        return          ;2cm                    REGRESAR

;Para 5000us requiere 20000cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 20000cm
;10+774L=20000 -> L=(20000-10)/774=25.8 -> L=25
;L=25 -> 19360cm -> 4840us ; compensa 160us con llamado al retardo 100,50,10us
retardo_5ms             ;2cm                    LLAMADO
        movlw   .25     ;1cm                    CARGAR L
        movwf   CONTA_L         ;1cm
        call    BucleAnidado1   ;BA1            LLAMAR BA1
        call    retardo_100us   ;compensación
        call    retardo_50us
        call    retardo_10us
        return          ;2cm                    REGRESAR

;Para 10 000us requiere 40 000cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 40 000cm
;10+774L=40 000 -> L=(40 000-10)/774=51.67 -> L=51
;L=51 -> cm -> 39484us ; compensa 129us con llamado al retardo 100,20,5,4us
retardo_10ms            ;2cm                    LLAMADO
        movlw   .51     ;1cm                    CARGAR L
        movwf   CONTA_L         ;1cm
        call    BucleAnidado1   ;BA1            LLAMAR BA1
        call    retardo_100us   ;compensación
        call    retardo_20us
        call    retardo_5us
```

```
        call    retardo_4us
        return          ;2cm                    REGRESAR

;Para 20 000us requiere 80 000cm
;[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
;igualando la ecuación con 80 000cm
;10+774L=80 000 -> L=(80 000-10)/774=103.3 -> L=103
;L=103 -> 79732cm -> 19 933us ; se compensa 67us con llamado al retardo 50,10,5,2us
retardo_20ms            ;2cm                    LLAMADO
        movlw   .103    ;1cm                    CARGAR L
        movwf   CONTA_L         ;1cm
        call    BucleAnidado1   ;BA1            LLAMAR BA1
        call    retardo_50us    ;compensación
        call    retardo_10us
        call    retardo_5us
        call    retardo_2us
        return          ;2cm                    REGRESAR

;BUCLE ANIDADO1
;su representación algebraica es la siguiente:
;[2+1L+1L+BB(L)+(L-1)+3+(L-1)2+2] = 7+2L+BB(L)+(L-1)3
;7+2L+3L-3+BB(L) = 4+5L+BB(L); sustituyendo BB con K=255
;4+5L+(4+3(255))(L)=4+5L+769L = 4+774L
;BucleAnidado1 = BA1 = 4+774L
BucleAnidado1                   ;2cm            LLAMADO
                                                ;HAZ
        movlw   .255            ;1cm(L)          CARGAR conta_k
        movwf   CONTA_K                 ;1cm(L)
        call    BucleBasico     ;(BB)(L)         LLAMAR BucleBasico
        decfsz  CONTA_L         ;(L-1)cm + 3cm   DECREMENTAR conta_L
        goto    BucleAnidado1   ;(L-1)2cm               MIENTRAS (conta_L=0?)
        return                  ;2cm            REGRESAR
;_____________________________________________________________________________
```

Para simular los retardos se deben hacer los llamados siguientes, que se pueden colocar como rutina principal:

```
call    retardo_100us
call    retardo_200us
call    retardo_300us
call    retardo_400us
call    retardo_500us
call    retardo_1ms
call    retardo_2ms
call    retardo_3ms
call    retardo_4ms
call    retardo_5ms
call    retardo_10ms
call    retardo_20ms
```

La ventaja de este código en ensamblador es que tenemos la certeza de que el tiempo de espera es el exacto, siempre y cuando la frecuencia de operación sea la correcta.

2.5.2. CONTROL DEL MÓDULO LCD

Este modulo puede resultar un poco complejo, pero si estructuramos cada operación, resulta bien fácil.

El pseudocódigo queda así:

```
;_____________________________________________________________________
  CONFIGURACIÓN DE PUERTOS
        ASEGURAR puertos como digitales (Se debe deshabilitar algunos módulos)
        CONFIGURAR bus de control como salida (RS, RW, E)
        CONFIGURAR bus de datos como salida (BD0-DB7)
  FIN CONFIGURACION DE PUERTOS

; Modos de funcionamiento del LCD
  MODO COMANDO
        ENVIAR señal digital RS=0 y RW=0
  REGRESAR

  MODO DATO
        ENVIAR señal digital RS=1 y RW=0
  REGRESAR

  MODO BUSY
        ENVIAR señal digital RS=0 y RW=1
        CONFIGURAR DB7 (Busy bit) como entrada
        ENVIAR Enable (E=1)

        MIENTRAS (Busy=1?)
        FIN MIENTRAS

        ENVIAR Enable (E=0)
        CONFIGURAR DB7 (Busy bit) como salida
  REGRESAR

;Igual que en las memorias, se utiliza el enable para indicar la entrada de un dato. Solo que
;aquí es por flanco, y el fabricante indica que todo el pulso puede ser mínimo de 500ns. En
;resumen un pulso cuadrado.

  PULSO ENABLE
        ENVIAR Pulso cuadrado Enable
  REGRESAR

;Rutina que envía una dato al LCD
ENVIAR DATO
        ENVIAR constante al bus de datos
        PULSO ENABLE
REGRESAR

;La inicialización puede ser de dos formas, POWER ON o INICIALIZACIÓN POR INSTRUCCIONES.
;La segunda opción es la más segura cuando no hay suficiente energía, por lo que es ideal
;en caso de que se quiera alimentar con baterías.
```

```
INICIALIZACION LCD
        ESPERAR más de 15ms

        MODO COMANDO
        ENVIAR DATO 0011xxxx
        ESPERAR mas de 4.1ms

        ENVIAR  DATO 0011xxxx
        ESPERAR mas de 100us

        ENVIAR DATO0011xxxx
        ESPERAR mas de 100us

;Estos últimos envíos son los de configuración.
;Define la comunicación de 8bits, número de líneas y tamaño de fuente

        MODO COMANDO
        ENVIAR DATO 0011 1100
        ESPERAR         200us

;Enciende el display
        MODO COMANDO
        ENVIAR DATO 0000 1111

;Limpia el display
        MODO COMANDO
        ENVIAR DATO 0000 0001

;Información se desplaza y DDRAM aumenta
        MODO COMANDO
        ENVIAR DATO 0000 0111
REGRESAR

;Subrutina para imprimir un carácter, necesita un contenedor.
CONTENEDORES
        CARACTER_LCD
FIN CONTENEDORES

IMPRIME_LCD
        GUARDAR el carácter a enviar, en CARACTER_LCD
        MODO BUSY    ;Se mantendrá hasta mientras el LCD este ocupado
        MODO DATO
        RECUPERAR el dato a enviar (CARACTER_LCD)
        ENVIAR DATO
REGRESAR

;Las siguientes subrutinas son para el manejo del display y del cursor
;SUBRUTINA PARA POSICIONAR EL CURSOR DE LA DDRAM

CONTENEDORES
        Instrucción_LCD
FINCONTENEDORES
```

```
POSICION 1,1
        CARGAR Dirección                ;primer posición de la línea 1
        POSICIÓN DE LA DDRAM
        RETURN
POSICIÓN 2,1
        CARGAR Dirección                ;Primer posición de la línea 2
        POSICIÓN DE LA DDRAM
        RETURN
POSICIÓN3,1
        CARGAR DIRECCIÓN        ;Primer posición de la línea virtual3
        POSICIÓN DE LA DDRAM
        RETURN
POSICIÓN 4,1
        CARGAR DIRECCIÓN        ;Primer posición de la línea virtual 4
        POSICIÓN DE LA DDRAM
        RETURN

POSICIÓN DE LA DDRAM
        GUARDAR Dirección DDRAM ;Lo deja en el contenedor Instrucción_LCD
        MODO BUSY               ;Checar si está listo el LCD
        MODO COMANDO                    ;Activa este modo
        RECUPERAR Dirección DDRAM       ;Lo toma del contenedor Instrucción_LCD
        ENVIAR DATO
REGRESAR

;_____________________________________________________________________________
```

Aun que sea pseudocódigo, no está de más algunos comentarios para tratar de aclarar algún detalle.

En el programa principal, se usan las directivas, para definir las instrucciones con las que opera el LCD. Obsérvese la librería.

Hasta el momento, se había requerido indicar como deben de quedar los bits de configuración, esto es porque no había sido tan necesario, porque contamos con un buen simulador, que nos muestra lo que sucede instrucción por instrucción.

Para el caso del LCD, es indispensable utilizar un buen simulador o hacerlo de manera práctica. Cuando se realizo el código, se experimento con varios simuladores, y aveces funcionaba el código en un simulador y en otro no, sobre todo cuando se variaba la frecuencia de la simulación. En conclusión ninguno resultaba muy confiable, además de que en ninguno de los simuladores se podía variar el contraste del LCD. Para un ejercicio tan sencillo como desplegar un mensaje existieron tantas ventajas como desventajas, y si se es primerizo en algún simulador, no se ahorra nada de tiempo. Así que se dejará a un lado la simulación virtual y se salta directo a la práctica. Aunque no se cuente con un buen simulador para el LCD, se puede contar con el simulador MPLAB SIM. Siguiendo paso a paso la ejecución de las instrucciones, podemos estar muy cerca de que nuestro código opere correctamente en la realidad y haga funcionar el dispositivo LCD.

La librería LCD depende, en parte, del buen funcionamiento de los retardos creados. Para que un programa pueda hacer uso de otro programa o, mejor dicho, de una librería, se debe hacer lo siguiente:

Para el código de retardos anteriormente expuesto, se deben eliminar algunas líneas, como son: la línea de la directiva #INCLUDE, las líneas con call, la instrucción sleep, solamente el 0x00 después de la directiva CBLOCK, y el END.

Cambiar la extensión .ASM del archivo, por la extensión .INC. Por ejemplo RETARDOS.ASM por RETARDOS.INC

Poner este archivo ó librería en la carpeta del proyecto nuevo para el funcionamiento del LCD.

Por último, en el programa principal, al final se incluirá la línea #INCLUDE <RETARDOS.INC> pero antes del END

El siguiente ejemplo es un programa que manda imprimir un mensaje, y hace uso de librerías creadas previamente.

```
;_____________________________________________________________
;*********************************************
;UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
;FACULTAD DE ESTUDIOS SUPERIORES
;"ARAGÓN"
;
;INGENIERÍA MECÁNICA ELÉCTRICA
;;
;NOMBRE
;Mensaje en el LCD 4x20
;
;DESCRIPCIÓN
;Manda imprimir mensajes
;*********************************************

;CABECERA
        LIST    P=PIC18F4550            ;El PIC18F4550 como procesador utilizado
        #INCLUDE<P18F4550.INC>          ;Define el archivo donde están definidos los registros del PIC

;BITS DE CONFIGURACIÓN DEL PIC
        CONFIG PLLDIV = 3, CPUDIV = OSC4_PLL6, FOSC = HSPLL_HS, USBDIV = 2,FCMEN = OFF
        CONFIG IESO = OFF, PWRT = ON, BOR = OFF, VREGEN = ON,WDT = OFF, WDTPS = 1
        CONFIG MCLRE = ON, LPT1OSC = OFF, PBADEN = OFF
        CONFIG CCP2MX = ON, STVREN = OFF, LVP = OFF, ICPRT = OFF, XINST = OFF, DEBUG = OFF

        CBLOCK 0x00
        Caja
        ENDC

;INICIO

        org     0x00                    ;Dirección del comienzo del programa
        goto    InicioMensaje
```

```
;Programa

InicioMensaje
        call    ConfigPuertos_LCD       ;Configuración de puertos
        call    Inicializa_LCD          ;Inicialización por instrucciones del LCD

        movlw   'U'                     ;Carácter a imprimir cargado en WREG
        call            ImprimeLCD      ;Imprime el carácter previamente cargado en WREG
        movlw   'N'
        call            ImprimeLCD
        movlw   'I'
        call            ImprimeLCD
        movlw   'V'
        call            ImprimeLCD
        movlw   'E'
        call            ImprimeLCD
        movlw   'R'
        call            ImprimeLCD
        movlw   'S'
        call            ImprimeLCD
        movlw   'I'
        call            ImprimeLCD
        movlw   'D'
        call            ImprimeLCD
        movlw   'A'
        call            ImprimeLCD
        movlw   'D'
        call            ImprimeLCD

        call            Linea_2_1               ;Posiciona el cursor en la línea 2, columna 1 del LCD
        movlw   'N'
        call            ImprimeLCD
        movlw   'A'
        call            ImprimeLCD
        movlw   'C'
        call            ImprimeLCD
        movlw   'I'
        call            ImprimeLCD
        movlw   'O'
        call            ImprimeLCD
        movlw   'N'
        call            ImprimeLCD
        movlw   'A'
        call            ImprimeLCD
        movlw   'L'
        call            ImprimeLCD

        call            Linea_3_1               ;Posiciona el cursor en la línea 3, columna1
        movlw   'A'
        call            ImprimeLCD
        movlw   'U'
        call            ImprimeLCD
        movlw   'T'
        call            ImprimeLCD
```

```
        movlw   'O'
        call            ImprimeLCD
        movlw   'N'
        call            ImprimeLCD
        movlw   'O'
        call            ImprimeLCD
        movlw   'M'
        call            ImprimeLCD
        movlw   'A'
        call            ImprimeLCD

        call            Linea_4_1               ;Posiciona el cursor en la línea 4, columna 1
        movlw   'D'
        call            ImprimeLCD
        movlw   'E'
        call            ImprimeLCD
        movlw   ' '
        call            ImprimeLCD
        movlw   'M'
        call            ImprimeLCD
        movlw   'E'
        call            ImprimeLCD
        movlw   'X'
        call            ImprimeLCD
        movlw   'I'
        call            ImprimeLCD
        movlw   'C'
        call            ImprimeLCD
        movlw   'O'
        call            ImprimeLCD

        call            DerechaDisplay  ;Indica el tipo de desplazamiento que queremos

        movlw   .1                              ;Indica el número de veces
        call            DesplazarCuadros        ;Desplaza 1x4 veces a la derecha el display

        sleep                                   ;Manda a dormir al PIC

        #INCLUDE <LCD_LM016L.INC>       ;Librerías declaradas con la extensión .INC
        #INCLUDE <RETARDOS.INC>
        END
;_____________________________________________________________________
```

Librería LCD_LM016L.INC

```
;_____________________________________________________________
;====================================
;UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
;FACULTAD DE ESTUDIOS SUPERIORES
;"ARAGÓN"
;
;NOMBRE
;Librería para el LCD
;
;DESCRIPCIÓN:
;Este documento es una librería para la
;configuración y control del Display LM016L
;====================================

;Buffer utilizados
        CBLOCK
        CaracterLCD
        InstruccionLCD
        iCuatro
        iCuadros
        ENDC

;DECLARACIÓN DE LOS PUERTOS
;BUS DE CONTROL Y
;BUS DE DATOS
        #DEFINE EnaLCD          PORTD,7         ;Enable
        #DEFINE ReaWriLCD       PORTD,6         ;Read Write
        #DEFINE RegSelLCD       PORTD,5         ;Register Select
        #DEFINE BusyBitLCD      PORTB,7         ;Busy Bit
        #DEFINE BusDatLCD       PORTB           ;Bus Datos LCD

;Constantes para configurar los puertos
        #DEFINE TrisEnaLCD      TRISD,7
        #DEFINE TrisReaWriLCD   TRISD,6
        #DEFINE TrisRegSelLCD   TRISD,5
        #DEFINE TrisBusyBitLCD  TRISB,7
        #DEFINE TrisBusDatLCD   TRISB

;DECLARACIÓN DE CONSTANTES
;Estas constantes son una lista de la tabla de instrucciones del modulo LCD
;Comandos de Control                    ;Descripción
ClearDisplay    EQU     01h             ;Borra pantalla y cursor al origen (0DDRAM)
ReturnHome      EQU     02h             ;Cursor Origen (00h de la DDRAM)

;Sirven para la estructura de datos (modo entrada)
;Entry mode set (0000 0 1 I/D S)        ;Combinaciones  Descripción
EntryMode1      EQU     07h             ;I/D=1, S=1;    S=1 -> Informacion no se desplaza
EntryMode2      EQU     06h             ;I/D=1, S=0;    S=0-> Informacion si se desplaza
EntryMode3      EQU     05h             ;I/D=0, S=1;    I/D=1-> Posición DDRAM aumenta
EntryMode4      EQU     04h             ;I/D=0, S=0;    I/D=0->Posicion DDRAM disminuye
```

```
;Para el control del display
;Display Control (0000 1 D C B)          Combinaciones   Descripción
DisplayControl1  EQU   0Fh              ;D=1, B=1, C=1;   D=1->Display ON
DisplayControl2  EQU   0Eh              ;D=1, B=1, C=0;   D=0->Display OFF
DisplayControl3  EQU   0Dh              ;D=1, B=0, C=1;   C=1->Cursor ON
DisplayControl4  EQU   0Ch              ;D=1, B=0, C=0;   C=0->Cursor OFF
DisplayControl5  EQU   0Bh              ;D=0, B=1, C=1;   B=1->Blink ON
DisplayControl6  EQU   0Ah              ;D=0, B=1, C=0;   B=0->Blink OFF
DisplayControl7  EQU   09h              ;D=0, B=0, C=1
DisplayControl8  EQU   08h              ;D=0, B=0, C=0

;Para el control de desplazamientos
;Cursor and Display Shift (0001 S/C R/L X X)
                                        ;Combinaciones   Descripción
CursorDisplay1   EQU   1Ch              ;S/C=1, R/L=1;   S/D->Efecto desplazamiento sobre cursor
CursorDisplay2   EQU   1Bh              ;S/C=1, R/L=0;   S/D->Efecto desplazamiento sobre display
CursorDisplay3   EQU   1Ah              ;R/L=0, R/L=1;   R/L->Left
CursorDisplay4   EQU   19h              ;R/L=0, R/L=0;   R/L->Right

;Control de Hardware, solo se usan en la inicialización.
;Function Set (0 0 1 DL N F X X)        ;Combinaciones   Descripción
FunctionSet1     EQU   3Ch              ;DL=1, N=1, F=1; DL=1->Data Length Comunicacion 8bit
FunctionSet2     EQU   38h              ;DL=1, N=1, F=0; DL=0->Data Length Comunicacion 4bit
FunctionSet3     EQU   34h              ;DL=1, N=0, F=1; N=1->Number Line de 2, Pantalla de 2 linea
FunctionSet4     EQU   30h              ;DL=1, N=0, F=0; N=0->Number Line de 1, Pantalla de 1 linea
FunctionSet5     EQU   2Ch              ;DL=0, N=1, F=1;  F=1->Font Caracteres de 5x10 puntos
FunctionSet6     EQU   28h              ;DL=0, N=1, F=0; F=0->Font Caracteres de 5x7 puntos
FunctionSet7     EQU   24h              ;DL=0, N=0, F=1
FunctionSet8     EQU   20h              ;DL=0, N=0, F=0

;SUBRUTINAS
;Subrutina que configura e inicializa los puertos para el LCD
ConfigPuertos_LCD:
        movlw   0x0F                    ;Como digitales
        movwf   ADCON1

        movlw   0x07                    ; Configura los comparadores
        movwf   CMCON           ; como entrada digital

        bcf     TrisEnaLCD              ;Bus de control como salida<0:3> <PORTA>
        bcf     TrisReaWriLCD
        bcf     TrisRegSelLCD
        clrf    TrisBusDatLCD           ;Limpia el bus de datos

        bcf     EnaLCD          ;RS, R/W y E = 0 (Impide el funcionamiento)
        bcf     ReaWriLCD
        bcf     RegSelLCD
        return

;Subrutina de inicialización para configurar el Display LCD
;NOTA:Si las condiciones de suministro de poder para operar correctamente el circuito interno reset no se
;encuentran, la INICIALIZACION POR INSTRUCCIONES es necesaria.
```

```
Inicializa_LCD:                 ;EL ENCENDER (POWER ON)
        call    retardo_20ms    ;Esperar por más de 15ms después de Power ON
;_____________________________
        call    ModoComandoLCD  ;RS=0 y R/W=0
        movlw   FunctionSet2    ;(0011 xxxx)
        call    EnviaDatLCD
        call    retardo_5ms     ;Esperar por más de 4.1ms
;_____________________________
        movlw   FunctionSet4    ;(0011 xxxx)
        call    EnviaDatLCD
        call    retardo_200us   ;Esperar por más de 100us
;_____________________________
        movlw   FunctionSet4    ;(0011 xxxx)
        call    EnviaDatLCD
        call    retardo_200us
;_____________________________

        ;ULTIMOS ENVIOS
        call    ModoComandoLCD  ;Tipo de comunicación (8bits)
        movlw   FunctionSet1    ;El numero de lineas y tamaño de fuente no puede
        call    EnviaDatLCD     ;ser cambiada despues de este punto
        call    retardo_200us

        call    ModoComandoLCD
        movlw   DisplayControl1 ;Display ON
        call    EnviaDatLCD

        call    ModoComandoLCD
        movlw   ClearDisplay    ;ClearDisplay
        call    EnviaDatLCD

        call    ModoComandoLCD
        movlw   EntryMode1      ;EntryModeSet
        call    EnviaDatLCD
        return

;Modos de funcionamientoLCD
ModoComandoLCD:                 ;La operación en este modo tarda un máx de 1.6ms
        bcf     RegSelLCD       ;RS=0; Register Select
        bcf     ReaWriLCD       ;R/W=0; Read Write
        return

ModoDatoLCD:                    ;La operación en este modo tarda un máx 40us
        bsf     RegSelLCD       ;RS=1; Register Select
        bcf     ReaWriLCD       ;R/W=0; Read Write
        return

ModoBusyLCD                     ;Esperar a que BusyBitLCD (P14) sea igual a 0
        bcf     RegSelLCD       ;RS=0; Register Select
        bsf     ReaWriLCD       ;R/W=1; Read Write
        bsf     TrisBusyBitLCD  ;RB7 como entrada

        bsf     EnaLCD          ;FIJAR Enable
```

```
BucleBusyLCD:
        btfsc   BusyBitLCD              ;MIENTRAS (Busy=1)
        goto    BucleBusyLCD            ;FINMIENTRAS

        bcf     EnaLCD                  ;Desactiva la señal Enable
        bcf     TrisBusyBitLCD          ;RB7 como salida
        return

;Subrutina Enable
AplicaEnaLCD:                           ;Pulso Enable
        bsf     EnaLCD
        bcf     EnaLCD
        return

;Subrutina para Imprimir un caracter al LCD
ImprimeLCD:
        movwf   CaracterLCD             ;GUARDAR el caracter a enviar
        call    ModoBusyLCD             ;MIENTRAS (Busy=1)
                                        ;FINMIENTRAS
        call    ModoDatoLCD             ;Activar Modo Dato
        movf    CaracterLCD,w           ;Recuperar el dato a enviar
        call    EnviaDatLCD             ;Enviar dato por el bus de datos
        return

;Subrutina para enviar una constante al bus de datos
EnviaDatLCD:
        movwf   BusDatLCD
        call    AplicaEnaLCD
        return

;SUBRUTINA PARA POSICIONAR EL CURSOR DE LA DDRAM
Linea_1_1:
        movlw   0x80                    ;Primer posición de la linea1
        goto    PosicionDDRAM
Linea_2_1:
        movlw   0xC0                     ;Primer posición de la linea2
        goto    PosicionDDRAM
Linea_3_1:
        movlw   0x94                    ;Primer posición de la linea3
        goto    PosicionDDRAM
Linea_4_1:
        movlw   0xD4                    ;Primer posición de la linea4
        goto    PosicionDDRAM
PosicionDDRAM:
        movwf   InstruccionLCD          ;Guardar dirección DDRAM
        call    ModoBusyLCD             ;Checar si está listo el LCD
        call    ModoComandoLCD          ;Activar modo comando
        movf    InstruccionLCD,w ;Recuperar dirección
        call    EnviaDatLCD             ;Enviar dato por el bus de datos
        return
```

```
;CONTROL DE LOS DESPLAZAMIENTOS DEL CURSOR Y DE LA PANTALLA
;Carga la instrucción necesaria para preparar el tipo de ejecución en el desplazamiento
DerechaDisplay:
        movlw   CursorDisplay1
        movwf   InstruccionLCD
        return
IzquierdaDisplay:
        movlw   CursorDisplay2
        movwf   InstruccionLCD
        return
DerechaCursor:
        movlw   CursorDisplay3
        movwf   InstruccionLCD
        return
IzquierdaCursor:
        movlw   CursorDisplay4
        movwf   InstruccionLCD
        return

;REALIZA UN DESPLAZAMIENTO
Desplazar:
        call    ModoBusyLCD             ;Checar si está listo el LCD
        call    ModoComandoLCD          ;Activa modo comando
        movf    InstruccionLCD,w        ;Recuperar Instrucción
        call    EnviaDatLCD             ;Envia Instrucción por el bus de datos
        return

;REALIZA 4 DESPLAZAMIENTOS
Desplazar4:                             ;Desplaza un cuadro (4 desplazamientos)
        movlw   0x04
        movwf   iCuatro
BucleDes4:
        call    Desplazar
        decfsz  iCuatro,f
        goto    BucleDes4
        return

;DESPLAZA EL NUMERO DE ESPACIOS x 4 INDICADOS PREVIAMENTE (movlw n)
DesplazarCuadros:                       ;Desplaza n cuadros
        movwf   iCuadros
BucleDesCuadros:
        call    Desplazar4
        decfsz  iCuadros,f
        goto    BucleDesCuadros
        return
;_____________________________________________________________________________
```

2.5.3. CONTROL DEL CONVERTIDOR A/D MCP3550

En el primer capítulo se describe el funcionamiento. Todo se puede resumir en lo siguiente: Dos modos de operación y chequeo del estado de operación interno con busy flag, o simplemente uso de retardos (esperarnos entre cada conversión).

El siguiente pseudocódigo para el modo de operación single conversion, usando busy flag.

```
;_____________________________________________________________________
CONTENEDORES
        MCPByte_1
        MCPByte_2
        MCPByte_3
        MCPByte
        Contador_8
FIN CONTENEDORES

CONFIGURACION DE PUERTOS MCP
        ASEGURAR puertos como digitales (deshabilitar algunos módulos)
        CONFIGURAR CS y SCK como salida
        CONFIGURAR SDO/RDY como entrada

        ; Incluimos inicialización
        ENVIAR estado Idle y comunicación SPI 1,1 (señales digitales CS=1 y SCK=1)
REGRESAR

;Subrutina que extrae un Byte (Clocked)
CLKBYTE
        INICIALIZAR Contador_8 = 8

        HAZ
           ENVIAR pulso de reloj con CS comenzando con pulso bajo
           LIMPIAR carry (C=0)
           SI (RDY=1) ENTONCES
              C=1
           FIN SI
           ROTAR a la derecha MCPByte con carry
           Contador_8 = Contador_8 – 1
        MIENTRAS (contador_8 = 0)
REGRESAR

BUSY_MCP_SC
        MIENTRAS (RDY=1)
           ENVIAR pulso de reloj CS comenzando con el pulso alto
        FIN MIENTRAS
REGRESAR

ADQUISICIÓN_MCP_SC
        INICIAR conversión (CS=0)
        ESPERAR 200us

        CAMBIAR a modo Single Conversion ;Se interrumpe la 1er conversión con un pulso
                                         ;según indica el diagrama de tiempo.
```

```
        BUSY_MCP                        ;Esperamos a que este lista la conversión

        INDICAR que tomaremos la muestra (CS=0)

        CLKBYTE
        COPIAR variable MCPByte en MCPByte_3
        CLKBYTE
        COPIAR variable MCPByte en MCPByte_2
        CLKBYTE
        COPIAR variable MCPByte en MCPByte_1

        ENTRAR en modo shutdown (CS=1)
REGRESAR
;_____________________________________________________________________________
```

Si queremos usar el modo Continuous Conversion entonces, reutilizando pseudocódigo tenemos lo siguiente:

```
;_____________________________________________________________________________
ADQUISICION_MCP_CC
        INICIAR conversión (CS=0)

        BUSY_MCP_CC                     ;Esperamos a que este lista la conversión

        CLKBYTE
        COPIAR variable MCPByte en MCPByte_3
        CLKBYTE
        COPIAR variable MCPByte en MCPByte_2
        CLKBYTE
        COPIAR variable MCPByte en MCPByte_1
REGRESAR

BUSY_MCP_CC
        MIENTRAS (RDY=1)
        FIN MIENTRAS
REGRESAR
;_____________________________________________________________________________
```

La diferencia radica en que solo se indica el inicio de la operación, no se necesitan tantos flancos en CS. Por último, BUSY también es diferente, ya que en este modo CC, el busy actualiza solo, sin necesidad de mandarle pulsos.

El código en ensamblador para Single Conversion queda:

```
;_____________________________________________________________________
;*********************************************
;UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
;FACULTAD DE ESYUDIOS SUPERIORES
;"ARAGÓN"
;
;INGENIERÍA MECÁNICA ELÉCTRICA
;PROGRAMACIÓN EN ENSAMBLADOR
;Familia 18F4550
;
;CONVERTIDOR ADC SERIAL MCP3550-60
;
;DESCRIPCIÓN
;El MCP3550-60 es un convertidor analógico digital
;serial, con una resolución de 22bits.
;El código de salida dígital
;Byte_3          Byte_2          Byte_1
;<23:16>         <15:8>          <7:0>
;
;-------------------------------------------------------
;CABECERA
        LIST    P=PIC18F4550
        #INCLUDE <P18F4550.INC>

;CONTENEDORES
;Variables Bits
        #DEFINE CS      PORTE,0,A       ;Chip Select
        #DEFINE SCK     PORTE,1,A       ;Serial Clock
        #DEFINE RDY     PORTE,2,A       ;Data Ready

;Variables Bytes
        CBLOCK          0x00
        MCPByte_1
        MCPByte_2
        MCPByte_3
        MCPByte
        Contador_8
        ENDC

;CONFIGURACIÓN DE PUERTOS PARA MCP3550
MCPConfiguracion
        movlw  0Fh                      ; CONFIGURAR A/D
        movwf  ADCON1                   ;para entradas digital
;Automáticamente elige con accses banck donde está el registro TRISE
        movlw  b'00001100'              ;CONFIGURAR CS y SCK -> Salida; SDO/RDY-> Entrada
        movwf  TRISE,A                  ;<0:OUT>, <1:OUT>, <2:IN>
                                        ;PREPARAR señales
```

```
        bsf     CS                  ;CS->High (Estado Idle)
        bsf     SCK                 ;SCK->High(Comunicación SPI 1,1)
        return

MCPAdquisicion
        bcf     CS                  ;INICIAR conversión (CS->Low)
        call    retardo_200us       ;ESPERAR tcsl (Tiempo mínimo para CS low)
        bsf     CS                  ;ACTIVAR Single-Conversion-Mode (CS->High)
                                    ;ESPERAR  tconv=Tiempo de conversion
        call    MCPBussy            ;       CHECAR bandera busy
        bcf     CS                  ;INDICAR que tomaremos la conversión

        call    CLKByte             ;TOMAR Byte
        movff   MCPByte,MCPByte_3   ;CARGAR variable (MCPByte_1=MCPByte)
        call    CLKByte             ;       TOMAR Byte_2
        movff   MCPByte,MCPByte_2
        call    CLKByte             ;       TOMAR Byte_3
        movff   MCPByte,MCPByte_1
        return                      ;FINSI

;Subrutina que sincroniza un byte
CLKByte
        movlw   .8                  ;INICIALIZAR
        movwf   Contador_8,A        ;Contador_8=8

BucleByte                           ;pulso de reloj
                                    ;HAZ
        bcf     SCK                 ;CS->Low
        bsf     SCK                 ;CS->High
        bcf     STATUS,C,A          ;Carry=0
        btfsc   RDY                 ;SI (RDY=1)ENTONCES
        bsf     STATUS,C,A          ;       Carry=1
        rlcf    MCPByte,f,A         ;FINSI
                                            ;ROTAR a la derecha MCPByte
        decfsz  Contador_8,f,A      ;Contador=Contador-1
        goto    BucleByte           ;MIENTRAS (Contador_8=0)
        return

MCPBussy                            ;       HAZ
        bsf     CS                  ;               CS->High
        bcf     CS                  ;               CS->Low
        btfsc   RDY                 ;       MIENTRAS (RDY=0)
        goto    MCPBussy

        #INCLUDE <RETARDOS.INC>
        END
;_____________________________________________________________
```

2.5.4. CODIFICADOR BCD

Lleva este nombre porque se trata de un programa que se encarga de convertir una trama de 22 bits, en complemento a dos, a código BCD. En el capitulo fundamentos están los conceptos que se ocupan en este código.

La elaboración de este código resultó sencillo, a pesar de que la arquitectura del microcontrolador solo almacena bytes, y esto resulta un poco complejo a la hora de relacionar los tres bytes en una sola trama. Para esto se utiliza la misma técnica de estructuración, elaborando los pseudocódigos.

Pero primero se hace un análisis del problema, en donde el objetivo es transformar la trama de 3bytes en código BCD, y para esto lo mejor es recurrir a la tabla de código de salida en complemento a dos del MCP3550. Analizando la tabla se observa que existen diversas condiciones, lo primero a notar en la trama es que los bits 23 (OVL)y 22 (OVH) son bits de desbordamiento, estos indican que el voltaje entre las terminales $V_{IN}+$ y $V_{IN}-$ excede el voltaje de referencia, por lo tanto, el resto de código de salida no corresponde al valor de entrada. La siguiente condición es que existen números positivos y números negativos en complemento a dos (indicado por el bit de signo), esto quiere decir que los números positivos no se les aplica la operación complemento a dos y a los números negativos si, entonces deberán procesarse independientemente. La tercer condición a observar es que la trama a transformar no está concatenada o unida, esto quiere decir que para transformar una trama de 22 bits se debe encontrar la manera de relacionar los tres bytes y con un orden; con orden se debe entender que el byte 3 es más significativo y el byte 1 es menos significativo.

```
;_____________________________________________________________________________
CONTENEDORES
        Byte3_BCD           ;Contienen la trama en complemento a 2
        Byte2_BCD
        Byte1_BCD
        SIGNO               ;Contiene la constante que indica el signo (+ ó -)
        UNIDADES            ;Contienen la cantidad que representa el valor BCD
        DECENAS
        CENTENAS
        UNIDAD DE MILLAR
        DECENA DE MILLAR
        CENTENA DE MILLAR
        UNIDAD DE MILLON
FIN CONTENEDORES

;Esta parte selecciona el tipo de proceso que se le va a dar a la trama, la llamaremos filtro y
;consta de dos estructuras de decisión binaria de dos ramas.
FILTRO_NUM_BCD
        SI (OVL o OVH =1) ENTONCES
           INDICAR valor excedido
        SINO
           SI (SIGNO = 1) ENTONCES
              NUMEROS NEGATIVOS
           SINO
              NUMEROS POSITIVOS
           FINSI
        FINSI
```

```
;Una estructura para los números positivos y una estructura para los números positivos

NUMEROS POSITIVOS
        INDICAR signo positivo      ;cargar la constante que representara el signo positivo
        CODIFICAR TRAMA BCD
REGRESAR

NUMEROS NEGATIVOS
        INDICAR signo negativo      ;cargar la constante que representa el signo negativo
        COMPLEMENTO A DOS           ;Transformamos el número con esta operación
        CODIFICAR TRAMA BCD
REGRESAR

;Esta subrutina codifica un solo byte a BCD, para esto, resta diez al byte e incrementa los
;contenedores según corresponda con el conteo decimal. Esta rutina la forman tres
;estructuras, la primera es una estructura mientras, la segunda un bloque simple y la tercera
;utiliza estructuras de decisión binaria de una sola rama anidadas tipo CASE.

CODIFICA BCD
        MIENTRAS (Byte > 9)
           RESTA_BCD
        FIN MIENTRAS
        COPIAR el contenido de byte en UNIDADES
REGRESAR

RESTA_BCD
        Byte = Byte – 10
        LLENAR los CONTENEDORES_BCD
REGRESAR

CONTENEDORES_BCD
        DECENAS++
        SI (DECENAS=10) ENTONCES
           DECENAS=0

           CENTENAS++
           SI (CENTENAS=10) ENTONCES
              CENTENAS=0

              UNIDAD DE MILLAR++
              SI (UNIDAD DE MILLAR=10) ENTONCES
                 UNIDAD DE MILLAR=0

                 DECENA DE MILLAR++
                 SI (DECENA DE MILLAR=10) ENTONCES
                    DECENA DE MILLAR=0

                    CENTENA DE MILLAR++
                    SI (CENTENA DE MILLAR=10) ENTONCES
                       CENTENA DE MILLAR=0
```

```
                    UNIDAD DE MILLON++
                FINSI
            FINSI
        FINSI
    FINSI
FINSI

;Para relacionar los tres bytes se tuvo que recurrir a los saltos, lo que en programación
;estructurada está prácticamente prohibido, pero todos los saltos se dirigen al mismo punto.
;Aún con este inconveniente se identifican perfectamente las estructuras que estamos
;utilizando, además de que se encontró un patrón que nos permite agregar o quitar el
;número de bytes que conforman la trama.

TRAMA BCD
    INICIALIZAR CONTENEDORES BCD        ;Todos se inicializan a cero

Punto recurrente
    CODIFICA BCD
    SI (Byte2 > 0) ENTONCES
        Byte2 = Byte2 – 1
        RESTA_BCD
        SALTAR al Punto recurrente
    FIN SI

    SI (Byte3 > 0) ENTONCES
        Byte3 = Byte3 – 1
        Byte2 = Byte2 – 1
        RESTA_BCD
        SALTAR al Punto recurrente
    FINSI
REGRESAR

;Si quisiéramos relacionar un cuarto byte, se agrega otra estructura idéntica al final pero, con
;un decremento al cuarto byte.

;Por último, esta subrutina aplica el complemento a 2 a la trama completa, en esta subrutina
;se considera una trama de 22 bits

COMPEMENTO A 2 TRAMA
    COMPLEMENTAR a los tres bytes

    LIMPIAR bits de desbordamiento, OVL y OVH
    Byte1++
    SI (Carry=1) ENTONCES       ;Se activa la bandera carry del registro estatus
                                ;cuando ocurre un desbordamiento
        Byte2++
        SI (Carry=1) ENTONCES   ;Se activará si ocurre un desbordamiento
            Byte3++
        FINSI
    FINSI
REGRESAR
;_____________________________________________________________________________
```

Este código está formado por muchas estructuras, que aunque son simples, no deja de tener su nivel de complejidad, por lo que es muy importante tener en cuenta al estructurarlo que debemos tener cuidado en relacionar bien las etiquetas que utiliza cada subrutina.

Ahora el siguiente paso es codificar. El siguiente ejemplo muestra cómo debe quedar el código, se realiza una pequeña rutina extra para probar nuestro código, así podremos simular y verificar que codifique correctamente.

```
;_____________________________________________________________________________
;***********************************************
;UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
;FACULTAD DE ESYUDIOS SUPERIORES
;"ARAGÓN"
;
;INGENIERÍA MECÁNICA ELÉCTRICA
;
;NOMBRE
;3_BYTES a BDC; PIC18F4550
;
;DESCRIPCIÓN
;Una trama de 3bytes (24bits) los cuales están en registros
;independientes y en complemento a 2, de los cuales los
;bit23 OVL y bit22 OVH son bit de desbordamiento del Vref
;y el bit 21 es el bit de signo.
;La trama es de la siguiente forma
;
;<OVL OVH Sig 20 19 18 17 16> <15 14 13 12 11 10 9 8> <7 6 5 4 3 2 1 0>
;       Byte3                   Byte2                   Byte1
;       Byte3_BCD               Byte2_BCD               Byte1_BCD
;
;Antes de usar la rutina, hay que cargar
;los valores en los buffer bytes correspondiente
;en el orden que indica la trama
;
;***********************************************
;CABECERA
#INCLUDE<P18F4550.INC>

        #DEFINE OVL     Byte3_BCD,7
        #DEFINE OVH     Byte3_BCD,6
        #DEFINE Sig     Byte3_BCD,5

;CONTENEDORES
        CBLOCK          0x00
        Byte3_BCD
        Byte2_BCD
        Byte1_BCD
        ENDC
```

```
;RUTINA que sirve únicamente para probar el funcionamiento del código
        movlw   h'FF'
        movwf   Byte1_BCD,A

        movlw   h'00'
        movwf   Byte2_BCD,A

        movlw   h'00'
        movwf   Byte3_BCD,A

        call    CONVIERTE_BCD

        sleep
infinito
        nop
        goto    infinito

;PROGRAMA PRINCIPAL
;*************************************
;Verifica si se trata de un número que excede el rango (- +Vref)
;o si se trata de un Número positivo, o Negativo
;*************************************
CONVIERTE_BCD
        btfsc   OVL,A                   ;SI(OVL=1)
        goto    EXED_BCD_NEG            ; Excede
                                        ;SINO
        btfsc   OVH,A                   ; SI(OVH=1)
        goto    EXED_BCD_POS            ;  Excede
                                        ; SINO
SIGNO_BCD
        btfsc   Sig,A                   ;  SI(Sig=1)
        goto    SIGNO_NEG               ;    Signo negativo
                                        ;  SINO
SIGNO_POS                               ;    Signo Positivo
        call    POS_BCD
        goto    FIN_CONVIERTE

SIGNO_NEG
        call    NEG_BCD
        goto    FIN_CONVIERTE

EXED_BCD_NEG
        call    Limpiar_Unidades
        movlw   0x0E
        movwf   SIGNO
        movlw   0x0D
        movwf   UNIMILL
        goto    FIN_CONVIERTE
```

```
EXED_BCD_POS
        call            Limpiar_Unidades
        movlw   0x0E
        movwf   SIGNO
        movlw   0x0C
        movwf   UNIMILL
FIN_CONVIERTE
        return
;**************************************
;Se diferencian por el signo y a los números negativos
;se les aplica el Complemento a 2
;
;**************************************
POS_BCD                                 ;Proceso para números positivos
        movlw   .10                     ;La variable signo se carga con 10
        movwf   SIGNO
        call            COD_BCD         ;Codifica a BCD directamente
        return
NEG_BCD                                 ;Proceso para números negativos
        movlw   .11                     ;La variable signo se carga con negativo
        movwf   SIGNO
        call    COMP_2                  ;Aplicamos a la trama el complemento a 2
        call    COD_BCD                 ;Codifica a BCD
        return

;SUBRUTINAS
;**************************************
;Se encarga de transformar la trama de 24 bits a BCD
;Es la rutina completa que relaciona los 3 Bytes
;
;**************************************
COD_BCD
        call    Limpiar_Unidades
Bucle_BCD
        call    OBTENER_BCD             ;Transformar Byte a BCD
        movlw   .0                      ;SI(BYTE2_BCD>0)
        cpfsgt  Byte2_BCD
        goto    Bucle2_BCD
        decf    Byte2_BCD,F,A           ; --BYTE2_BCD
        call            Bloque_BCD      ; Bloque_BCD
        goto    Bucle_BCD               ;SINO al inicio

Bucle2_BCD
        movlw   .0                      ;SI(BYTE3_BCD>0)
        cpfsgt  Byte3_BCD
        goto    FIN_POS_BCD
        decf    Byte3_BCD,F,A           ; --BYTE3_BCD
        decf    Byte2_BCD,F,A           ; --BYTE2_BCD
        call    Bloque_BCD              ; Bloque_BCD
        goto    Bucle_BCD               ;SINO al inicio

FIN_POS_BCD
        return
```

```
;*************************************
;Se encarga de transformar un byte a BCD
;
;*************************************
OBTENER_BCD
        movlw   .9                      ;MIENTRAS(BYTE_BCD > 9)
        cpfsgt  Byte1_BCD,A             ;f con w;salta si >
        goto    FIN_OBTENER_BCD
        call    Bloque_BCD
        goto    OBTENER_BCD
FIN_OBTENER_BCD
        movff   Byte1_BCD,UNI
        return

Bloque_BCD
        movlw   .10                     ;BYTE_BCD=BYTE_BCD - 10
        subwf   Byte1_BCD,F,A
        call    Colocar_BCD             ;Coloca en sus respectivos buffer
        return

;*************************************
;Pequeña rutina que limpia los buffer de unidades
;
;*************************************
Limpiar_Unidades
        clrf            UNI
        clrf            DEC
        clrf            CEN
        clrf            UNIMIL
        clrf            DECMIL
        clrf            CENMIL
        clrf            UNIMILL
        return

;*************************************
;Coloca el número  bcd en sus respectivos buffer de unidades
;
;*************************************
;CONTENEDORES
        CBLOCK
        SIGNO
        UNI
        DEC
        CEN
        UNIMIL
        DECMIL
        CENMIL
        UNIMILL
        ENDC
```

```
Colocar_BCD
        incf    DEC,F,A                 ;++DEC
        movlw   .10                     ;SI(DEC=10)
        cpfseq  DEC,A
        goto    FIN_Colocar
        clrf    DEC,A                   ;DEC=0

        incf    CEN,F,A                 ;++CEN
        movlw   .10                     ;SI(CEN=10)
        cpfseq  CEN,A
        goto    FIN_Colocar
        clrf    CEN,A                   ;CEN=0

        incf    UNIMIL,F,A              ;++UNIMIL
        movlw   .10                     ;SI(UNIMIL=10)
        cpfseq  UNIMIL,A
        goto    FIN_Colocar
        clrf    UNIMIL,A                ;UNIMIL=0

        incf    DECMIL,F,A              ;++DECMIL
        movlw   .10                     ;SI(DECMIL=10)
        cpfseq  DECMIL,A
        goto    FIN_Colocar
        clrf    DECMIL,A                ;DECMIL=0

        incf    CENMIL,F,A              ;++CENMIL
        movlw   .10                     ;SI(DECMIL=10)
        cpfseq  CENMIL,A
        goto    FIN_Colocar
        clrf    CENMIL,A                ;DECMIL=0

        incf    UNIMILL

FIN_Colocar
        return

;**************************************
;Subrutina que aplica el Complemento a una trama
; de 22bits
;**************************************
COMP_2
        comf    Byte1_BCD,F,A           ;Complemento a uno a BYTE_BCD
        comf    Byte2_BCD,F,A           ;Complemento a uno a BYTE2_BCD
        comf    Byte3_BCD,F,A           ;Complemento a uno a BYTE3_BCD

        bcf     Byte3_BCD,7,A           ;Limpiamos los bits de desbordamiento
        bcf     Byte3_BCD,6,A           ;OVL y OVH

        incf    Byte1_BCD,F,A           ;BYTE_BCD=BYTE_BCD + 1
        btfss   STATUS,C,A              ;SI(Carry=1)
        goto    FIN_COMP_2
```

```
        incf    Byte2_BCD,F,A           ; BYTE2_BCD =BYTE2_BCD+ 1
        btfss   STATUS,C,A              ; SI(Carry=1)
        goto    FIN_COMP_2
        incf    Byte3_BCD,F,A           ;   BYTE3_BCD =BYTE3_BCD+ 1
                                        ;FINSI
FIN_COMP_2
        return


        END
;_____________________________________________________________________________________________
```

2.5.5. CONTROL DEL MÓDULO USB

Uno de los principales objetivos es el uso del módulo USB integrado en el PIC18F4550. El convertidor A/D MCP3550 es un dispositivo que, en comparación de su velocidad de transmisión y conversión con respecto a la del USB, no es muy rápido. Entonces no es necesario explotar la gran capacidad de este módulo. Asi que se optó por un tipo de comunicación bastante sencillo que no requiere de muchas capas de software. Se trata del USB cdc (comunication device class). En síntesis se trata de una comunicación bidireccional serial que se puede realizar entre el PIC y algún software que cuente con interface serial, ya que aunque se trata de una comunicación USB full speed, lo hace como si se tratara de un puerto COM serie.

VID&PID

El VID (Vendor Identification) es un número de identificación ó codigo que reconoce al fabricante del hardware a conectar. Es un número de 16bits. Microchip presta su número para podernos conectar, es el 04D8h.

EL PID (Product Identification) es la identificación del producto o código que reconoce al dispositivo o hardware a conectar. Es un número de 16bits. La familia de los PIC18 de microchip se identifica con el número 000Bh.

La conjunción de los números VID&PID sirve para conectar con el driver de Windows XP. Cuando el Sistema Operativo conecte con nuestro firmware recibirá el VID&PID y buscará entre sus drivers instalados para encontrar el que corresponde con esta identificación. U en caso de no encontrarlo, preguntará donde debe buscarlo y deberemos decir en donde se encuentra su ubicación. **El driver deberá estar configurado para conectar con un hardware cuyo VID&PID sea el mismo.**

El driver para puerto serie ya existe en Windows, solo debemos darle el enlace con el .inf para que conecte correctamente con el firmware que tiene el PIC.

Para un ejemplo de transmisión se propone el siguiente programa:

```
//***********************************************//
//UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
//FACULTAD DE ESTUDIOS SUPERIORES
//"ARAGÓN"
//
//INGENIERÍA MECÁNICA ELÉCTRICA
//
//PROGRAMACIÓN EN LENGUAJE "C"
//ORIENTADO A MICROCONTROLADORES
//
//NOMBRE
//MÓDULO USB; PIC18F4550
//USB_cdc (Universal Serial Bus / Comunication Device Class)
//
```

```
//DESCRIPCIÓN
//Se configura el módulo USB del PIC18F4550 para que se
//comunique con la PC, como si tratase de una comunicación
//de puerto serial.
//En la hyperterminal se podrá ver un envió de
//números enteros
//El conteo va desde 0 a 255
//
//***********************************************//

//CABECERA
#INCLUDE <18F4550.h>      //PIC18F4550 esta familia tiene módulo USB
#FUSES HSPLL,NOWDT,MCLR,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL3,CPUDIV4,VREGEN,NOPBADEN
/*
HSPLL: Habilitar cristal HS de alta velocidad, con el PLL para generar los 48Mhz.
MCLR: Habilitar reset manual.
NOWDT: Deshabilitar el perro guardián.
NOPROTECT: Deshabilitar la protección de código.
NOLVP: Deshabilitar la programación a bajo voltaje.
NODEBUG: No entrar al modo debug.
USBDIV: El clock del usb se tomará del PLL/2 = 96Mhz/2 = 48Mhz.
PLL3: El PLL prescaler dividirá en 3 la frecuencia del cristal. Para HS = 12Mhz/3 = 4Mhz.
CPUDIV4: El PLL postscaler decide la división entre 6 de la frecuencia de salida del PLL de 96MHZ, si queremos
16MHZ
VREGEN: Habilitar el regulador de 3.3 volts que usa el módulo USB.
NOPBADEN: No utilizar las entradas analógicas del puerto B.
*/
#USE STANDARD_IO(B)                  //Directiva que configura automáticamente los puertos
//Como copiamos las librerías en otra carpeta como medida de precaución, indicamos su ubicación.
#include ".\include\usb_cdc.h"        //Descripción de funciones cdc del USB.
#include ".\include\usb_desc_cdc.h"   //Descriptores del dispositivo

//CONTENEDORES
int8 iConteo;

//PROGRAMA PRINCIPAL
void main (void)
{
  usb_cdc_init();                    //Configura el protocolo serial
  usb_init();                        //Inicializa periférico USB

  WHILE(!usb_cdc_connected())        //Determina si la PC esta lista para desplegar datos
  {
    output_high(PIN_B0);             //Función USB no detectada Enciende LED rojo
    output_low(PIN_B1);              //Apaga LED verde
  }
  output_low(PIN_B0);                //Función USB detectada Apaga LED rojo
  output_high(PIN_B1);               //Enciende LED verde
```

```
  DO                                    //Estructura HAZ-MIENTRAS (primero ejecuta el bloque)
  {
    usb_task();                         //Mantiene vigilado el pin conection sense para
                                        //determinar si estamos conectados o no a un cable USB.
                                        //Si estamos conectados, inicializa el periférico si es necesario
                                        //Si no, deshabilita el periférico
    IF(usb_enumerated())                //Estructura SI-ENTONCES de una solo rama
    {
      output_high(PIN_B3);              //Enciende LED indicador de enumeración

      FOR(iConteo=0; iConteo<=255; ++iConteo)     //Estructura PARA
      {
        printf(usb_cdc_putc,"%X",iConteo);        //Lo pone en el buffer de transmisión y envía Hexadecimal
        delay_ms(300);
      }
    }

  }WHILE(TRUE);                         //Bucle infinito
}                                       //Fin
;_____________________________________________________________________________________
```

Es más fácil entender de qué trata el programa con la descripción de las funciones de las librerías usadas.

Estas funciones son generales:

usb_init()

Inicializa la pila USB, el periférico USB y conecta la unidad al bus USB. Habilita las interrupciones.

usb_init_cs()

Una pequeña usb_init(), no conecta la unidad al bus USB ni habilita las interrupciones.

usb_enumerated()

Regresa TRUE (verdadero=1) si el dispositivo se ha enumerado (configurado) por el host y regresa FALSE (falso =0) si no es así. No se debe tratar de usar el periférico USB hasta que no sea enumerado.

usb_wait_for_enumeration()

Espera en un bloque infinito hasta que el dispositivo sea enumerado.

usb_kbhit()

Regresa un TRUE si el Endpoint OUT contiene datos del Host

usb_puts()

Envia un mensaje de paquetes múltiples para el host.

usb_gets()

Obtiene múltiples paquetes desde el host.

Estas funciones sirven para la comunicación tipo USB_cdc:

usb_cdc_init()

Inicializa los parámetros de la comunicación tipo serial como bits por segundo, bits de datos, paridad, bits de parada, etc.

Para recibir tenemos:

usb_cdc_kbhit()

Regresa TRUE si hay uno o más caracteres recibidos y esperando en el buffer de recepción.

usb_cdc_getc()

Toma un caracter del buffer de recepción. Si no hay ningún dato en el buffer de recepción, esperara indefinidamente hasta que haya datos en el buffer de recepción.

Si no se quiere esperar infinitamente, lo mejor es checar con sub_cdc_kbhit() primero, antes de usar usb_cdc_getc().

Para enviar tenemos:

usb_cdc_putready()

Regresa TRUE si hay espacio en los buffer de transmisión para otro caracter.

usb_cdc_putc(char c)

Pone un caracter en el buffer de transmisión. Si el buffer de transmisión está lleno, esperará infinitamente hasta que el buffer no esté lleno, antes de poner el caracter en el buffer de transmisión. El buffer de transmisión es leído por la PC muy rápidamente, por lo que el buffer debe estar lleno en pocos milisegundos. Si no se quiere quedar en un bucle infinito, entonces use primero usb_cdc_putready() para ver si hay espacio en el buffer de transmisión, antes de poner un dato.

usb_cdc_putc_fast(char c)

similar a usb_putc(), exepto si el buffer está lleno se saltara el char.

Para la conexión:

usb_cdc_conected()

Regresa TRUE (1) si recibimos un Set_Line_Coding. En la mayoría de los programas por puerto serie (ejemplo el hyperterminal), se enviará un mensaje Set_Line_Coding cuando se inicia el programa y se habre el puerto virtual COM. Esta es una forma simple para determinar si la PC está lista para mostrar los datos en un programa de terminal serial, pero esto no es garantía para trabajar todo el tiempo o en otros programas de terminal serial. Para que esta función opere correctamente, debemos definir el pin sensor de conexión. Si no se define regresará siempre un 1. Se ocupa más adelante en el ejemplo final.

Estas descripciones fueron tomadas de las mismas librerías del Programa PIC C. Si queremos más detalles de su función hay que consultarlas. Las librerías se encuentran por default en C:\Archivos de programa\PICC\Drivers.

Ahora que sabemos dónde se encuentran estas librerías, podemos también modificarlas un poquito. Debemos estar concientes que al modificarlas, los cambios serán permanentes en cualquier otro proyecto que hagamos. Así que también podemos optar por copiarlas y ponerlas en una carpeta aparte e indicar con el uso de la directiva #include, donde el compilador debe buscar dicha librería.

Las modificaciones fueron las siguientes:

En la librería **usb_cdc.h** hay que buscar este apartado, y en este ejemplo se pone en negritas la línea que se modificó:

```
;_____________________________________________________________________________

////////////////////////////////////////////////////////////////////////////
//
// Include the CCS USB Libraries.  See the comments at the top of these
// files for more information
//
////////////////////////////////////////////////////////////////////////////
#ifndef __USB_PIC_PERIF__
#define __USB_PIC_PERIF__          1
#endif

#if __USB_PIC_PERIF__
 #if defined(__PCM__)
  #error CDC requires bulk mode!  PIC16C7x5 does not have bulk mode
 #else
  #include <pic18_usb.h>  //Microchip 18Fxx5x hardware layer for usb.c
 #endif
#else
 #include <usbn960x.c>  //National 960x hardware layer for usb.c
#endif
#include <usb_desc_cdc.h>          //USB Configuration and Device descriptors for this UBS device
#include <usb.c>       //handles usb setup tokens and get descriptor reports
;_____________________________________________________________________________
```

Se cambia por la dirección nueva ejemplo: #include ".\TESIS\usb_desc_cdc.h"

Ahora los cambios para la librería ***usb_desc_cdc.h.***

Se debe asegurar que el VID&PID de nuestro dispositivo coincida con el del driver. Esto podemos verificarlo en la librería usb_desc_cdc.h en el siguiente apartado, en las líneas marcadas en negritas.

```
;_____________________________________________________________________________

////////////////////////////////////////////////////////////////
///
///  start device descriptors
///
////////////////////////////////////////////////////////////////

  const char USB_DEVICE_DESC[USB_DESC_DEVICE_LEN] ={
    //starts of with device configuration. only one possible
      USB_DESC_DEVICE_LEN, //the length of this report   ==0
      0x01, //the constant DEVICE (DEVICE 0x01)  ==1
      0x10,0x01, //usb version in bcd  ==2,3
      0x02, //class code. 0x02=Communication Device Class ==4
      0x00, //subclass code ==5
      0x00, //protocol code ==6
      USB_MAX_EP0_PACKET_LENGTH, //max packet size for endpoint 0. (SLOW SPEED SPECIFIES 8) ==7

      0xD8,0x04,       //vendor id (0x04D8 is Microchip)
      0x0A,0x00,       //product id
```

```
// RR2 cambiado para      0x61,0x04, //vendor id (0x04D8 is Microchip, or is it 0x0461 ??)  ==8,9
// compatibilidad con .inf 0x33,0x00, //product id   ==10,11
// de Microchip

      0x00,0x01, //device release number  ==12,13
      0x01, //index of string description of manufacturer. therefore we point to string_1 array (see below)  ==14
      0x02, //index of string descriptor of the product  ==15
      0x00, //index of string descriptor of serial number  ==16
      USB_NUM_CONFIGURATIONS  //number of possible configurations  ==17
  };
;_____________________________________________________________________________
```

Con la siguiente cadena se puede poner nombre al dispositivo.

USB_STRING_DESC[ ]

La tabla USB_STRING_DESC contiene la descripción del dispositivo detectado por el Driver de Windows XP, y que nos va a mostrar en la correspondiente entrada en la lista del Hardware Instalado en el Sistema.

Consta de dos partes o tablas, la propiamente dicha **USB_STRING_DESC,** que contiene las descripciones requeridas, y una tabla accesoria llamada **USB_STRING_DESC_OFFSET,** que contiene los offset o desplazamientos con respecto al inicio de USB_STRING_DESC, en donde, se encuentran las correspondientes cadenas. Ambas constan de tres elementos cada una de ellas.

USB_STRING_DESC_OFFSET tiene tres números que indican cada uno de ellos donde comienza el correspondiente dato en la tabla USB_STRING_DESC. Así, un contenido de {0,4,12} nos dice que el primer string comienza en el byte 0, el segundo en el byte 4 y el tercero se encuentra a partir del byte número 12. Si cambiamos la longitud de cualquiera de los strings, deberemos reordenar esta tabla correspondientemente con solo contar los caracteres y apuntar en esta tabla el número de byte donde comienza cada uno de ellos.

USB_STRING_DESC contiene los tres strings en concreto que deseamos transmitir con el descriptor USB. Cada uno de ellos tiene la misma estructura, que consta de un primer byte que indica la longitud total de la correspondiente cadena, un segundo byte que indica el tipo de dato que viene a continuación, y por último tantos bytes como sean necesarios como contenido del string.

El primer dato de esta tabla es:

4, USB_DES_STRING_TYPE, 0x09, 0x04 que puede leerse como 4 : Longitud en bytes del dato, incluido él mismo. USB_DES_STRING_TYPE que es una constante cuyo valor es 3 y que dice que lo que sigue es un string. Y 0x09, 0x04 que le indica al Windows que los strings que siguen están escritos en correcto inglés americano (US-English).

Los dos siguientes datos son los dos strings que definen nuestro dispositivo y cuya estructura es idéntica al caso anterior:

8, USB_DES_STRING_TYPE, 'F', 0, 'E', 0, 'R', 0 que define el string el nombre "FER" de mis Hardware. Total 8 bytes, ya que "FER" se codifica añadiendo un 0x00 tras cada uno de los caracteres.

Y 24, USB_DESC_STRING_TYPE, 'U', 0, 'N', 0, 'I', 0, 'V', 0, 'E', 0, 'R', 0, 'S', 0, 'I', 0, 'D', 0, 'A', 0, 'D' 0 que define el nombre de mi dispositivo como "UNIVERSIDAD".

Esto se puede configurar en el apartado final de la librería usb_desc_cdc.

```
//////////////////////////////////////////////////////////////////
///
///  start string descriptors
///  String 0 is a special language string, and must be defined.  People in U.S.A. can
///  leave this alone.
///
///  You must define the length else get_next_string_character() will not see the string
///  Current code only supports 10 strings (0 thru 9)
///
//////////////////////////////////////////////////////////////////

//the offset of the starting location of each string.  offset[0] is the start of string 0, offset[1] is the start of string
1, etc.
char USB_STRING_DESC_OFFSET[]={0,4,12};

char const USB_STRING_DESC[]={
  //string 0
      4, //length of string index
      USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
      0x09,0x04,  //Microsoft Defined for US-English
  //string 1
      8, //length of string index
      USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
      'F',0,
      'E',0,
      'R',0,
  //string 2 --> nombre del dispositivo
      24, //length of string index
      USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
      'U',0,
      'N',0,
      'I',0,
      'V',0,
      'E',0,
      'R',0,
      'S',0,
      'I',0,
      'D',0,
      'A',0,
      'D',0
};
;_____________________________________________________________________________
```

Ponemos estas librerías en una carpeta dentro de la carpeta de nuestro proyecto, por ejemplo:

Nuestro proyecto en la siguiente dirección: F:\Doc_Fer\Tesis\USB_cdc_Envio

Las librerías usb en la siguiente dirección F:\Doc_Fer\Tesis\\USB_cdc_Envio\include

Entonces podemos compilar. Esto como precaución si no se quiere alterar de forma negativa las librerías, para hacer algunas pruebas.

Finalmente, los pasos a seguir para conectar el PIC a la PC:

1. Instalar el driver para XP que proporciona microchip, llamado mchpcdc.inf, esto se hace dándole al archivo click con el botón secundario del mouse y seleccionando y dando click en la opción instalar.
2. Una vez bien montado el pic en su circuito, lo conectamos a la PC y, cuando el S.O. de la PC lo detecte recibiendo su VID&PID, y buscará entre sus drivers instalados, si no lo encuentra nos preguntara donde encontrarlo, entonces debemos indicar su dirección.
3. Abriendo la hyperterminal, podemos ver el conteo que se programó.

Con esto finaliza el estudio de cada módulo por separado, esto resulta indispensable para la elaboración del código final.

2.6. CONSTRUCCIÓN DE LA FUNCIÓN BÁSICA DEL SISTEMA DE ADQUISICIÓN

Una vez probado el funcionamiento de cada una de las etapas, podemos entrar a la vista funcional, subiendo de nivel, de manera que no es necesario un estudio tan detallado. Pero antes, contando con las etapas construidas con diferentes plataformas, lenguaje ensamblador en MPLAB IDE y lenguaje de alto nivel en el PIC C de CCSC. La pregunta obvia es ¿Como integrarlas, y en que plataforma de software?, y la solución nos la da el programa PIC C, que incluye directivas con las que podemos integrar instrucciones de lenguaje ensamblador.

Para poner código ensamblador en un programa en C, se hace escribiéndolo dentro de las directivas #ASM y #ENDASM. Pero no es tan fácil como copiar nuestro código ensamblador y pegarlo asi nada mas entre estas dos directivas. Nuestro código ensamblador no es 100% puro, hacemos uso de muchos recursos que nos proporciona el otro compilador, como las directivas para declarar variables, constantes etc. Los siguientes ejemplos explican esto de mejor forma.

```
ENSAMBLADOR MPLAB                    MIGRADO A C
Para las variables
;CONTENEDORES                        //CONTENEDORES
      CBLOCK    0x05F ;Asigna desde la    //Se asigna la dirección de cada
                      ;dirección 0x05F    // una si queremos reservar en
                                          //específico
      Byte3_MCP                           #LOCATE Byte3_MCP = 0x05F
      Byte2_MCP                           #LOCATE Byte2_MCP = 0x05E
      Byte1_MCP                           #LOCATE Byte1_MCP = 0x05D
      ENDC
Para declarar los puertos
;BUS DE CONTROL Y DATOS                  //BUS DE CONTROL Y DATOS
#DEFINE EnaLCD   PORTD,7          #BIT    EnaLCD = PORTD.7
#DEFINE ReaWriLCD      PORTD,6            #BIT ReaWriLCD = PORTD.6
#DEFINE RegSelLCD      PORTD,6            #BIT RegSelLCD = PORTD.5
#DEFINE BusyBitLCD     PORTB.7            #BIT BusyBitLCD = PORTB.7
#DEFINE BusDatLCD      PORTB              #BYTE BusDatLCD = PORTB

Unas macro para definir constantes
ClearDisplay    EQU   01h                 #DEFINE ClearDisplay    0x01
ReturnHome      EQU   02h                 #DEFINE ReturnHome      0x02
```

La diferencia entre las etiquetas es que en ensamblador puede o no llevar dos puntos, y en C si es obligatorio, ejemplo:

Etiqueta1 Etiqueta1:

Estos son los cambios básicos, pero faltan algunas cosas como el que PIC C no reconoce que parámetro es PORTC, ESTATUS, etc. Significa que necesitamos una librería para eso, librería que podemos tomar prestada del MPLAB IDE como P18F4550.INC, y que puede declararse igual #INCLUDE <P18F4550.INC>, y para que funcione se tiene que modificar. Según el tutorial incluido en PIC C, es de esta forma, por ejemplo: #byte status = 3 ó #byte b_port = 6. Otro ejemplo ya enfocado en el PIC modificando la librería sería:

```
#BYTE   PORTC   =   0x0F82
#BYTE   PORTD   =   0x0F83
#BYTE   PORTE   =   0x0F84
#BYTE   LATA    =   0x0F89
#BYTE   LATB    =   0x0F8A
#BYTE   LATC    =   0x0F8B
#BYTE   LATD    =   0x0F8C
#BYTE   LATE    =   0x0F8D
etc.
```

De esta forma solo borramos lo nesesario y solo pegamos #BYTE un = y 0x junto a las direcciones en hexadecimal, unas diagonales (//) en los comentarios y listo. Una vez modificada a nuestras necesidades, se debe pegar dentro de la carpeta del proyecto y, al declararla, esta lista para funcionar.

No es un procedimiento muy ortodoxo, que si no se es cuidadoso, se puede cometer muchos errores. Pero sigue cumpliendo con uno de los objetivos que es tener en la medida de lo posible un código eficiente, refinado y que además se puede simular en el MPLAB SIM, con la garantía de que funciona correctamente.

Este trabajo no es una labor muy fácil, ya que cuenta con muchas etapas, cada una con su respectivo nivel de complejidad. Una vez listas las etapas, se ensamblan formando toda la función. Es recomendable hacer una lista de las subrutinas en ensamblador de las que podemos hacer uso y a qué etapa pertenecen. Este concepto está en diagramas a bloques con sus entradas y salidas, como el ejemplo de la figura (2.6 – 1).

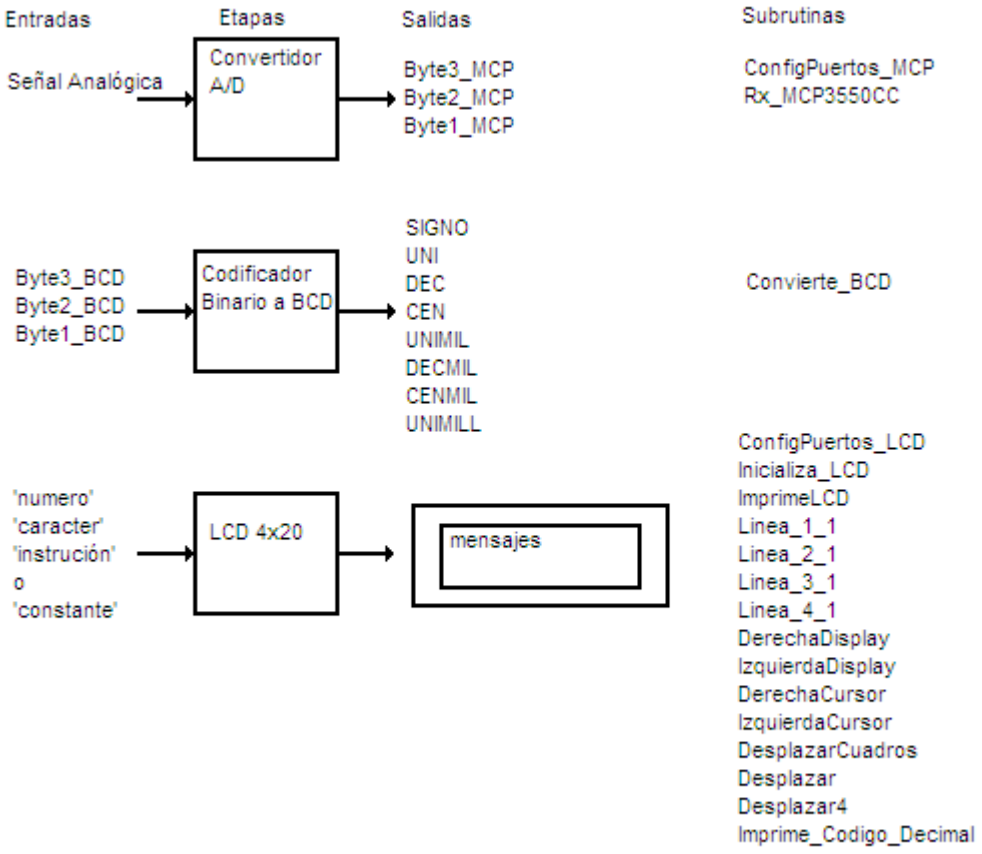


Figura (2.6 - 1) Abstracción de los códigos representándolos como diagrama a bloques.

El ensamblado se hace en PIC C, aprobechando la base del programa que envía un conteo de 0 a 255, haciendo pruebas enviando la trama que se obtiene con el codificador. Esto significa que se junta primero la etapa del convertidor A/D (Analógico/Digital) con la del USB, si funciona, se agrega la etapa del codificador enviando los datos ahora en BCD, por último agregamos la etapa del LCD, para saber si funcionan juntos.

En la práctica fue un arduo trabajo en pruebas, debido a que al principio no se tomó la precaución de ir juntando etapa por etapa, para asegurar que hiba funcionando parte por parte al juntarlas. Y aunque ya se habían hecho pruebas independientes de cada una de las etapas, no garantiza un buen resultado a la hora de integrarlas, ya que el proceso de sincronización al unir todas las etapas puede fluctuar un poco, ya que debemos encontrar el mejor orden para eso. Como comentario extra, este proceso de ensamble se fue realizando metódicamente con la elaboración de borradores, pseudocódigo, simulación en el MPLAB IDE, en otros simuladores también, que no fueron incluidos por sus características, que los hacían poco fiables al momento de la simulación, y terminaron en muchos de los casos en pérdida de tiempo.

Otra de las situaciones adversas es que no se puede hacer un llamado (call) dentro del programa principal, y que este llame a su subrutina (otro archivo aparte, pegado como las librerías), por lo que se tienen que pegar en un solo documento, si es que queremos disfrutar de los beneficios del ensamblador. La razón de que no se puede es que c funciona en base a funciones prototipo, y una instrucción como “call” no contiene los parámetros de una función en C. Por esta razón el compilador no puede enlazar una subrutina con una instrucción “call” a una etiqueta de una subrutina, dentro de un archivo independiente.

Un tip para quienes hagan un proyecto similar en donde inserten lenguaje ensamblador y recurran a un manejo de tiempos muy exacto, como en este caso. Se recomienda no hacer uso de los retardos del PIC C, con la directiva #USE DELAY(clock=16000000), y hacer algo como lo siguiente. Recordando la inicialización del LCD. No se debe hacer algo como esto:

```
Inicializa_LCD:                                 ;EL ENCENDER (POWER ON)
      delay_ms(20);                             //Power ON
#ASM
      call        ModoComandoLCD                ;RS=0 y R/W=0
      movlwFunctionSet2           ;(0011 xxxx)
      call        EnviaDatLCD
#ENDASM
      delay_ms(5);                              // Esperar por mas de 4.1ms
#ASM
      movlwFunctionSet4           ;(0011 xxxx)
      call        EnviaDatLCD
#ENDASM
      delay_us(200);
.
.
.
etc
```

Se intercala indebidamente ensamblador y C.

Parece ser que se conserva la linealidad del programa. Incluso se realizaron pruebas con programas más sencillos que funcionan correctamente, pero la prueba definitiva, con un dispositivo como el LCD que es más complejo, no la pasó y simplemente no funciona bien este método. También es posible que se pierda esta supuesta linealidad con retardos más grandes. Queda la advertencia hecha, así que es recomendable las subrutinas de retardos en ensamblador que funcionan perfectamente.

Este archivo es tomado de las librerías del MP ASM y solo modificamos para su uso en C, de esta manera no es necesario recurrir al manual del PIC para ver las direcciones de los RFS (Registros de Funciones Específicas), etc. En resumen, se utilizan los cambios de directivas para hacer la migración a C.

El siguiente programa fue compilado por completo en PIC C y funciona perfectamente.

```
;_____________________________________________________________________________

//***********************************************//
//UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
//FACULTAD DE ESTUDIOS SUPERIORES
//"ARAGÓN"
//
//INGENIERÍA MECÁNICA ELÉCTRICA
//PRÁCTICAS DE LABORATORIO
//PROGRAMACIÓN EN LENGUAJE "C"
//ORIENTADO A MICROCONTROLADORES
//
//Ortega Nava Carlos Fernando
//
//NOMBRE
//SISTEMA DE ADQUISICION DE DATOS USB
//
//DESCRIPCIÓN
//Este programa está construido por varios códigos
//que son enlazados para construir toda la función
//Esta es la función base más importante del proyecto.
//***********************************************//

//CABECERA
#INCLUDE <18F4550.H>      //PIC18F4550 esta familia tiene módulo USB
#FUSES HSPLL,NOWDT,MCLR,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL3,CPUDIV4,VREGEN,NOPBADEN

/*
HSPLL: Habilitar cristal HS de alta velocidad, con el PLL para generar los 48Mhz.
MCLR: Habilitar reset manual.
NOWDT: Deshabilitar el perro guardián.
NOPROTECT: Deshabilitar la protección de código.
NOLVP: Deshabilitar la programación a bajo voltaje.
NODEBUG: No entrar al modo debug.
USBDIV: El clock del usb se tomará del PLL/2 = 96Mhz/2 = 48Mhz.
PLL3: El PLL prescaler dividirá en 3 la frecuencia del cristal. Para HS = 12Mhz/3 = 4Mhz.
CPUDIV4: El PLL postscaler decide la división entre 6 de la frecuencia de salida del PLL de 96MHZ, si queremos
16MHZ
VREGEN: Habilitar el regulador de 3.3 volts que usa el módulo USB.
NOPBADEN: No utilizar las entradas analógicas del puerto B.
*/
```

```
//Indican donde se encuentran
#USE DELAY(clock=16000000)          //Retardos para CPU trabajando a 16MHz
#INCLUDE ".\include\usb_cdc.h"      //Descripción de funciones cdc del USB.
#INCLUDE ".\include\usb_desc_cdc.h"   //Descriptores del dispositivo
#INCLUDE <P18F4550.INC>            //Librería del PIC18F4550 modificada

//CONTENEDORES DEL PROGRAMA PRINCIPAL
int iConteo;

//#############################################################################
//CONTENEDORES PARA LOS RETARDOS
//#############################################################################
#LOCATE CONTA_K = 0x04B
#LOCATE CONTA_L = 0x04A
//#############################################################################
//CONTENEDORES Y CONSTANTES PARA EL CÓDIGO DEL MCP3550
//#############################################################################
//CONTENEDORES
#LOCATE Byte_MCP = 0x060
#LOCATE Byte3_MCP = 0x05F              //Estos buffer son requerimientos
#LOCATE Byte2_MCP = 0x05E              //de la Librería Mues_MCP3550_SC.INC
#LOCATE Byte1_MCP = 0x05D
#LOCATE Contador_8 = 0x05C            //Este lleva la cuenta de bits
//FINCONTENEDORES

//DECLARACION DE LOS PUERTOS
//BUS DE CONTROL
#BIT   CS = PORTD.2               //Chip Select
#BIT   SCK = PORTD.0               //Serial Clock
#BIT   RDY = PORTD.1               //Data Ready
//Para configurar los puertos
#BIT   TrisCS = TRISD.2
#BIT   TrisSCK = TRISD.0
#BIT   TrisRDY = TRISD.1

//#############################################################################
//CONTENEDORES Y CONSTANTES PARA CÓDIGO DEL LCD 4x20
//#############################################################################
#LOCATE CaracterLCD = 0x05B
#LOCATE InstruccionLCD = 0x05A
#LOCATE iCuatro = 0x059
#LOCATE iCuadros = 0x058
//FINCONTENEDORES

//DECLARACION DE LOS PUERTOS
//BUS DE CONTROL Y
//BUS DE DATOS
#BIT   EnaLCD = PORTD.7            //Enable
#BIT   ReaWriLCD = PORTD.6         //Read Write
#BIT   RegSelLCD = PORTD.5         //Register Select
#BIT   BusyBitLCD = PORTB.7        //Busy Bit
#BYTE  BusDatLCD = PORTB           //Bus Datos LCD
```

```
//Para configurar los puertos
#BIT    TrisEnaLCD = TRISD.7
#BIT    TrisReaWriLCD = TRISD.6
#BIT    TrisRegSelLCD = TRISD.5
#BIT    TrisBusyBitLCD = TRISB.7
#BYTE   TrisBusDatLCD = TRISB

//DECLARACIÓN DE CONSTANTES
//Comandos de Control
#DEFINE ClearDisplay    0x01       //Borra pantalla y cursor al origen (0DDRAM)
#DEFINE ReturnHome      0x02       //Cursor Origen (00h de la DDRAM)

//Sirven para la estructura de datos (modo entrada)
//Entry mode set (0000 0 1 I/D S)    Combinaciones    Descripción
#DEFINE EntryMode1      0x07       //D=1,   S=1;      S=1 -> Información no se desplaza
#DEFINE EntryMode2      0x06       //I/D=1, S=0;      S=0-> Información si se desplaza
#DEFINE EntryMode3      0x05       //I/D=0, S=1;      I/D=1-> Posición DDRAM aumenta
#DEFINE EntryMode4      0x04       //I/D=0, S=0;      I/D=0-> Posición DDRAM disminuye

//Para el control del display
//Display Control (0000 1 D C B)     Combinaciones    Descripción
#DEFINE DisplayControl1 0x0F       //D=1, B=1, C=1;    D=1->Display ON
#DEFINE DisplayControl2 0x0E       //D=1, B=1, C=0;    D=0->Display OFF
#DEFINE DisplayControl3 0x0D       //D=1, B=0, C=1;    C=1->Cursor ON
#DEFINE DisplayControl4 0x0C       //D=1, B=0, C=0;    C=0->Cursor OFF
#DEFINE DisplayControl5 0x0B       //D=0, B=1, C=1;    B=1->Blink ON
#DEFINE DisplayControl6 0x0A       //D=0, B=1, C=0;    B=0->Blink OFF
#DEFINE DisplayControl7 0x09       //D=0, B=0, C=1
#DEFINE DisplayControl8 0x08       //D=0, B=0, C=0

//Para el control de desplazamientos
//Cursor and Display Shift (0001 S/C R/L X X)
                                   //Combinaciones  Descripción
#DEFINE CursorDisplay1  0x1C       //S/C=1, R/L=1;  S/D->Efecto desplazamiento sobre cursor
#DEFINE CursorDisplay2  0x1B       //S/C=1, R/L=0;  S/D->Efecto desplazamiento sobre display
#DEFINE CursorDisplay3  0x1A       //R/L=0, R/L=1;  R/L->Left
#DEFINE CursorDisplay4  0x19       //R/L=0, R/L=0;  R/L->Right

//Control de Hardware, solo se usan en la inicialización.
//Function Set (0 0 1 DL N F X X)    Combinaciones   Descripción
#DEFINE FunctionSet1    0x3C       //DL=1, N=1, F=1; DL=1->Data Length Comunicación 8bit
#DEFINE FunctionSet2    0x38       //DL=1, N=1, F=0; DL=0->Data Length Comunicación 4bit
#DEFINE FunctionSet3    0x34       //DL=1, N=0, F=1; N=1->Number Line de 2, Pantalla de 2 línea
#DEFINE FunctionSet4    0x30       //DL=1, N=0, F=0; N=0->Number Line de 1, Pantalla de 1 línea
#DEFINE FunctionSet5    0x2C       //DL=0, N=1, F=1;  F=1->Font Caracteres de 5x10 puntos
#DEFINE FunctionSet6    0x28       //DL=0, N=1, F=0; F=0->Font Caracteres de 5x7 puntos
#DEFINE FunctionSet7    0x24       //DL=0, N=0, F=1
#DEFINE FunctionSet8    0x20       //DL=0, N=0, F=0
```

```
//#############################################################################
//CONTENEDORES Y CONSTANTES PARA EL CODIFICADOR BCD
//#############################################################################
#LOCATE Byte3_BCD = 0x057
#LOCATE Byte2_BCD = 0x056
#LOCATE Byte1_BCD = 0x055

#LOCATE SIGNO = 0x054
#LOCATE UNI = 0x053
#LOCATE DEC = 0x052
#LOCATE CEN = 0x051
#LOCATE UNIMIL = 0x050
#LOCATE DECMIL = 0x04F
#LOCATE CENMIL = 0x04E
#LOCATE UNIMILL = 0x04D
//FINCONTENEDORES

//DEFINICIONES
#BIT   OVL = Byte3_BCD.7
#BIT   OVH = Byte3_BCD.6
#BIT   Sig = Byte3_BCD.5

//#############################################################################
//CONTENEDORES PARA LA TABLA_LCD
//#############################################################################
#LOCATE NumBCD = 0x04C

//#############################################################################
//PROGRAMA PRINCIPAL
//#############################################################################

void main (void)
{
  #ASM                          ;Directiva que indica que se inserta código ensamblador
  call   ConfigPuertos_MCP
  call   ConfigPuertos_LCD
  call   Inicializa_LCD
  movlw  'E'
  call   ImprimeLCD
  movlw  'S'
  call   ImprimeLCD
  movlw  'P'
  call   ImprimeLCD
  movlw  'E'
  call   ImprimeLCD
  movlw  'R'
  call   ImprimeLCD
  movlw  'A'
  call   ImprimeLCD
  call   MCP3550_CC_ON
  #ENDASM                       ;Fin del código insertado.
```

```
usb_cdc_init();
usb_init();

WHILE(!usb_cdc_connected()) //Mientras el USB no esté conectado
                  //Determina si la PC esta lista para desplegar datos
{
}


DO
{
  usb_task();     //Mantiene vigilado el pin conection sense para
                  //deteminar si estamos conectados o no a un cable USB.
                  //Si estamos conectados, inicializa el periferico si es necesario
                  //Si no, deshabilita el periferico
  IF(usb_enumerated())
  {
    FOR(iConteo=0; iConteo<=255; ++iConteo)
    {
      #ASM
      call   Rx_MCP3550_CC      //Recibe muestra
      call   MCP3550_CC_ON      //Prepara otra muestra

      movff  Byte3_MCP,Byte3_BCD //Copiamos la muestra en los
      movff  Byte2_MCP,Byte2_BCD //buffer de entrada del codificador
      movff  Byte1_MCP,Byte1_BCD
      call   Convierte_BCD      //MANDAR convertir
      call   Linea_1_1
      call   Imprime_Codigo_Decimal //Imprime los dígitos en LCD
      #ENDASM
      //Este código envía la trama por USB
      //Enviamos la muestra en Hexadecimal
      printf(usb_cdc_putc,"\rTrama");
      printf(usb_cdc_putc,"%X",Byte3_MCP);
      printf(usb_cdc_putc,"%X",Byte2_MCP);
      printf(usb_cdc_putc,"%X",Byte1_MCP);
      printf(usb_cdc_putc," ");
      //Enviamos la muestra en decimal
      printf(usb_cdc_putc,"%X",SIGNO);
      printf(usb_cdc_putc,"%d",UNIMILL);
      printf(usb_cdc_putc,"%d",CENMIL);
      printf(usb_cdc_putc,"%d",DECMIL);
      printf(usb_cdc_putc,"%d",UNIMIL);
      printf(usb_cdc_putc,"%d",CEN);
      printf(usb_cdc_putc,"%d",DEC);
      printf(usb_cdc_putc,"%d",UNI);
    }
  }
}WHILE(TRUE);
```

```
//*****************************************************************************
//ENSAMBLADOR
//*****************************************************************************
//*****************************************************************************
//CÓDIGO DEL MCP3550
//####################################
//Este programa permite extraer la trama de 22 bit que manda el MCP
//El MCP3550 será manejado en SC (Single Conversion) en modo SPI 1,1
//El orden de extracción de pulsos es primero el Byte_3, segundo el Byte_2
//y por último el Byte_1, que son los buffer necesarios para tener la trama
//completa.
//Cada byte es extraído bit por bit, sincronizando con el SCK
//La trama completa debe ser como la siguiente
//Byte_3      Byte_2      Byte_1
//<23:16>     <15:8>      <7:0>
//MSB----------------------LSB
//####################################
//########################################
//SUBRUTINA DE CONFIGURACION DE PUERTOS PARA EL MCP3550
//########################################
#ASM
ConfigPuertos_MCP:
   movlw   0x0F                  //Configurra todos los pin ANX
   movwf   ADCON1                //como I/O digital
                                 //Bus de control
   bcf     TrisCS                //CS->Salida
   bcf     TrisSCK               //SCK->Salida
   bsf     TrisRDY               //RDY->Entrada
                                 //Inicializar
   bsf     CS                    //Preparar CS Modo ShutDown
   bsf     SCK                   //Preparar SCK Modo SPI 1,1 (Idle High)
   return
#ENDASM

//########################################
//SUBRUTINA DE RECEPCIÓN Rx
//Esta subrutina Obtiene la trama de 22 bits del MCP3550 en modo Conversión
//Única y la guarda en tres buffer.
//########################################
#ASM
MCP3550_CC_ON:
   bcf     CS                    //INICIAR la conversión Pulso low en Chip Select
   call    retardo_200us
   return

Rx_MCP3550_CC:
//  bsf     CS                    //ACTIVAR Single Conversion Mode
   call    Busy_MCP3550          //MIENTRAS no hay muestra
//  bcf     CS                    //INDICAR que tomaremos la muestra
```

```
   call    CLKByte                  //TOMAR Byte
   movff   Byte_MCP,Byte3_MCP       //CARGAR variable (MCPByte_1=MCPByte)
   call    CLKByte                  //  TOMAR Byte_2
   movff   Byte_MCP,Byte2_MCP
   call    CLKByte                  //  TOMAR Byte_3
   movff   Byte_MCP,Byte1_MCP
   bsf     CS                       //ENTRAR Modo ShutDown
  return          ;FINSI

CLKByte:
   movlw   8                        //INICIALIZAR
   movwf   Contador_8               //Contador_8=8
BucleByte:                          //MANDAR pulso de reloj
                                    //HAZ
   bcf     SCK                      // CS->Low
   bsf     SCK                      // CS->High
   bcf     STATUS,C                 // Carry=0
   btfsc   RDY                      // SI (RDY=1) ENTONCES
   bsf     STATUS,C                 //  Carry=1
   rlcf    Byte_MCP,f               // FINSI
                                    // ROTAR a la derecha MCPByte
   decfsz  Contador_8,f             // Contador=Contador-1
   goto    BucleByte                //MIENTRAS (Contador_8=0)
  return

//#####################################
//Checa si el MCP3550 está desocupado
//#####################################
Busy_MCP3550:                       //MIENTRAS(RDY=1)
   btfsc   RDY                      //FIN MIENTRAS
   goto    Busy_MCP3550
   return

 //  btfss   RDY
 //  goto    FINBUSYMCP
  // bsf     CS
  // bcf     CS
  // goto    Busy_MCP3550
FINBUSYMCP:
   return
#ENDASM

//*****************************************************************************
//CÓDIGO DEL LCD 4x16 COMUNICACIÓN 8BITS
//#####################################
//Este documento es una libreria para la
//configuración y control del Display LM016L
//#####################################
//#####################################
//SUBRUTINA DE CONFIGURACION DE PUERTOS PARA EL LCD
//#####################################
```

```
#ASM
ConfigPuertos_LCD:
   movlw   0x0F         //Como digitales
   movwf   ADCON1
   movlw   0x07         //Configura los comparadores
   movwf   CMCON        //como entrada digital
   bcf     TrisEnaLCD   //Bus de control como salida<0:3> <PORTA>
   bcf     TrisReaWriLCD
   bcf     TrisRegSelLCD
   clrf    TrisBusDatLCD  //Bus de datos como salida
   bcf     EnaLCD       //RS, R/W y E = 0 (Impide el funcionamiento)
   bcf     ReaWriLCD
   bcf     RegSelLCD
   return

//#####################################
//SUBRUTINA DE INICIALIZACIÓN PARA EL LCD
//NOTA: Si las condiciones de suministro de poder para operar correctamente
//el circuito interno reset no se encuentran, la INICIALIZACION POR INSTRUCCIONES
//es necesaria.
//#####################################
Inicializa_LCD:                 //EL ENCENDER (POWER ON)
                                //Esperar por más de 15ms después de Power ON
   call    retardo_20ms
   call    ModoComandoLCD       //RS=0 y R/W=0
   movlw   FunctionSet2         //(0011 xxxx)
   call    EnviaDatLCD
                                //Esperar por más de 4.1ms
   call    retardo_5ms
   movlw   FunctionSet4         //(0011 xxxx)
   call    EnviaDatLCD
                                //Esperar por más de 100us
   call    retardo_200us
   movlw   FunctionSet4         //(0011 xxxx)
   call    EnviaDatLCD
                                //Esperar por más de 100us
   call    retardo_200us
//ULTIMOS ENVIOS
   call    ModoComandoLCD       //Tipo de comunicación (8bits)
   movlw   FunctionSet1         //El número de lineas y tamaño de fuente no puede
   call    EnviaDatLCD          //ser cambiada despues de este punto
   call    retardo_200us
   call    ModoComandoLCD
   movlw   DisplayControl1      //Display ON
   call    EnviaDatLCD
   call    ModoComandoLCD
   movlw   ClearDisplay         //ClearDisplay
   call    EnviaDatLCD
   call    ModoComandoLCD
   movlw   EntryMode1           //EntryModeSet
   call    EnviaDatLCD
   return
```

```
//#####################################
//Modos de funcionamientoLCD
//#####################################
ModoComandoLCD:          //La operación en este modo tarda un máx de 1.6ms
   bcf    RegSelLCD      //RS=0; Register Select
   bcf    ReaWriLCD      //R/W=0; Read Write
   return

ModoDatoLCD:             //La operación en este modo tarda un máx 40us
   bsf    RegSelLCD      //RS=1; Register Select
   bcf    ReaWriLCD      //R/W=0; Read Write
   return

ModoBusyLCD:             //Esperar a que BusyBitLCD (P14) sea igual a 0
   bcf    RegSelLCD      //RS=0; Register Select
   bsf    ReaWriLCD      //R/W=1; Read Write
   bsf    TrisBusyBitLCD //RB7 como entrada
   bsf    EnaLCD         //FIJAR Enable
BucleBusyLCD:
   btfsc  BusyBitLCD     //MIENTRAS (Busy=1)
   goto   BucleBusyLCD   //FINMIENTRAS
   bcf    EnaLCD         //Desactiva la señal Enable
   bcf    TrisBusyBitLCD //RB7 como salida
   return

//#####################################
//Subrutina Enable
//#####################################
AplicaEnaLCD:            //Pulso Enable
   bsf    EnaLCD
   bcf    EnaLCD
   return

//#####################################
//Subrutina para Imprimir un caracter al LCD
//#####################################
ImprimeLCD:
   movwf  CaracterLCD    //GUARDAR el caracter a enviar
   call   ModoBusyLCD    //MIENTRAS (Busy=1)
                         //FINMIENTRAS
   call   ModoDatoLCD    //Activar Modo Dato
   movf   CaracterLCD,w  //Recuperar el dato a enviar
   call   EnviaDatLCD    //Enviar dato por el bus de datos
   return

//#####################################
//Subrutina para enviar una constante al bus de datos
//#####################################
EnviaDatLCD:
   movwf  BusDatLCD
   call   AplicaEnaLCD
   return
```

```
//#####################################
//SUBRUTINA PARA POSICIONAR EL CURSOR DE LA DDRAM
//#####################################
Linea_1_1:
   movlw  0x80                  //Primer posición de la linea1
   goto   PosicionDDRAM
Linea_2_1:
   movlw  0xC0                  //Primer posición de la linea2
   goto   PosicionDDRAM
Linea_3_1:
   movlw  0x94                  //Primer posición de la linea3
   goto   PosicionDDRAM
Linea_4_1:
   movlw  0xD4                  //Primer posición de la linea4
   goto   PosicionDDRAM
PosicionDDRAM:
   movwf  InstruccionLCD        //Guardar dirección DDRAM
   call   ModoBusyLCD           //Checar si está listo el LCD
   call   ModoComandoLCD        //Activar modo comando
   movf   InstruccionLCD,w      //Recuperar dirección
   call   EnviaDatLCD       //Enviar dato por el bus de datos
   return

//#####################################
//CONTROL DE LOS DESPLAZAMIENTOS DEL CURSOR Y DE LA PANTALLA
//Carga la instrucción necesaria para preparar el tipo de ejecución
//en el desplazamiento
//#####################################
DerechaDisplay:
   movlw  CursorDisplay1
   movwf  InstruccionLCD
   return
IzquierdaDisplay:
   movlw  CursorDisplay2
   movwf  InstruccionLCD
   return
DerechaCursor:
   movlw  CursorDisplay3
   movwf  InstruccionLCD
   return
IzquierdaCursor:
   movlw  CursorDisplay4
   movwf  InstruccionLCD
   return

//#####################################
//REALIZA UN DESPLAZAMIENTO
//#####################################
Desplazar:
   call   ModoBusyLCD           //Checar si está listo el LCD
   call   ModoComandoLCD        //Activa modo comando
   movf   InstruccionLCD,w      //Recuperar Instrucción
   call   EnviaDatLCD           //Envia Instrucción por el bus de datos
   return
```

```
//#####################################
//REALIZA 4 DESPLAZAMIENTOS
//#####################################
Desplazar4:                         //Desplaza un cuadro (4 desplazamientos)
   movlw   0x04
   movwf   iCuatro

BucleDes4:
   call    Desplazar
   decfsz  iCuatro,f
   goto    BucleDes4
   return

//#####################################
//DESPLAZA EL NUMERO DE ESPACIOS x 4 INDICADOS PREVIAMENTE (movlw n)
//#####################################
DesplazarCuadros:                   //Desplaza n cuadros
   movwf   iCuadros
BucleDesCuadros:
   call    Desplazar4
   decfsz  iCuadros,f
   goto    BucleDesCuadros
   return
#ENDASM



//*************************************************************************
//CODIGO DEL CODIFICADOR
//#####################################
//Una trama de 3bytes (24bits) los cuales están en registros
//independientes y en complemento a 2, de los cuales los
//bit23 OVL y bit22 OVH son bit de desbordamiento del Vref
//y el bit 21 es el bit de signo.
//La trama es de la siguiente forma
//
//<OVL OVH Sig 20 19 18 17 16> <15 14 13 12 11 10 9 8>  <7 6 5 4 3 2 1 0>
//          Byte3              Byte2              Byte1
//          BYTE_BCD           BYTE2_BCD          BYTE3_BCD
//
//Antes de usar la rutina, hay que cargar
//los valores en los buffer bytes correspondiente
//en el orden que indica la trama
//#####################################
//#####################################
//SUBRUTINA PRINCIPAL
//Verifica si se trata de un numero que excede el rango (- +Vref)
//o si se trata de un número positivo, o Negativo
//#####################################
```

```
#ASM
Convierte_BCD:
   btfsc  OVL                    //SI(OVL=1)
   goto   EXED_BCD_NEG           //Exed
                                 //SINO
   btfsc  OVH                    //SI(OVH=1)
   goto   EXED_BCD_POS           //Exed
                                  //SINO
SIGNO_BCD:
   btfsc  Sig                    //SI(Sig=1)
   goto   SIGNO_NEG              //Signo negativo
                                 //SINO
SIGNO_POS:                       //Signo Positivo
   call   POS_BCD
   goto   FIN_CONVIERTE

SIGNO_NEG:
   call   NEG_BCD
   goto   FIN_CONVIERTE

EXED_BCD_NEG:
   call   Limpiar_Unidades
   movlw  0x0E
   movwf  SIGNO
   movlw  0x0D
   movwf  UNIMILL
   goto   FIN_CONVIERTE

EXED_BCD_POS:
   call   Limpiar_Unidades
   movlw  0x0E
   movwf  SIGNO
   movlw  0x0C
   movwf  UNIMILL

FIN_CONVIERTE:
   return
//#####################################
//Se aplica un proceso diferente, dependiendo del signo.
//Se diferencian por el signo y a los números negativos
//se les aplica el Complemento a 2
//#####################################
POS_BCD:              //Proceso para números positivos
   movlw  10          //La variable signo se carga con cero
   movwf  SIGNO
   call   COD_BCD      //Codifica a BCD directamente
   return

NEG_BCD:              //Proceso para números negativos
   movlw  11          //La variable signo se carga con negativo
   movwf  SIGNO
   call   COMP_2       //Aplicar a la trama el complemento a 2
   call   COD_BCD      //Codifica a BCD
   return
```

```
//#####################################
//Se encarga de transformar la trama de 24 bits a BCD
//Es la rutina completa que relaciona los 3 Bytes
//#####################################
COD_BCD:
   call    Limpiar_Unidades
Bucle_BCD:
   call    OBTENER_BCD              //Transformar Byte a BCD
   movlw   0                        //SI(BYTE2_BCD>0)
   cpfsgt  Byte2_BCD
   goto    Bucle2_BCD
   decf    Byte2_BCD,F              //--BYTE2_BCD
   call    Bloque_BCD               //Bloque_BCD
   goto    Bucle_BCD                //SINO al inicio
Bucle2_BCD:
   movlw   0                        //SI(BYTE3_BCD>0)
   cpfsgt  Byte3_BCD
   goto    FIN_POS_BCD
   decf    Byte3_BCD,F              //--BYTE3_BCD
   decf    Byte2_BCD,F              //--BYTE2_BCD
   call    Bloque_BCD               //Bloque_BCD
   goto    Bucle_BCD                //SINO al inicio
FIN_POS_BCD:
   return

//#####################################
//Se encarga de transformar un byte a BCD
//#####################################
OBTENER_BCD:
   movlw   9                        //MIENTRAS(BYTE_BCD > 9)
   cpfsgt  Byte1_BCD                //f con w;salta si >
   goto    FIN_OBTENER_BCD
   call    Bloque_BCD
   goto    OBTENER_BCD
FIN_OBTENER_BCD:
   movff   Byte1_BCD,UNI
   return
Bloque_BCD:
   movlw   10                       //BYTE_BCD=BYTE_BCD - 10
   subwf   Byte1_BCD,F
   call    Colocar_BCD              //Coloca en sus respectivos buffer
   return
//#####################################
//Pequeña rutina que limpia los buffer de unidades
//#####################################
Limpiar_Unidades:
   clrf    UNI
   clrf    DEC
   clrf    CEN
   clrf    UNIMIL
   clrf    DECMIL
   clrf    CENMIL
   clrf    UNIMILL
   return
```

```
//#####################################
//Coloca el num bcd en sus respectivos buffer de unidades
//#####################################
Colocar_BCD:
   incf   DEC,F                  //++DEC
   movlw  10                     //SI(DEC=10)
   cpfseq DEC
   goto   FIN_Colocar
   clrf   DEC                    //DEC=0

   incf   CEN,F                  //++CEN
   movlw  10                     //SI(CEN=10)
   cpfseq CEN
   goto   FIN_Colocar
   clrf   CEN                    //CEN=0

   incf   UNIMIL,F               //++UNIMIL
   movlw  10                     //SI(UNIMIL=10)
   cpfseq UNIMIL
   goto   FIN_Colocar
   clrf   UNIMIL                 //UNIMIL=0

   incf   DECMIL,F               //++DECMIL
   movlw  10                     //SI(DECMIL=10)
   cpfseq DECMIL
   goto   FIN_Colocar
   clrf   DECMIL                 //DECMIL=0

   incf   CENMIL,F               //++CENMIL
   movlw  10                     //SI(DECMIL=10)
   cpfseq CENMIL
   goto   FIN_Colocar
   clrf   CENMIL                 //DECMIL=0

   incf   UNIMILL
FIN_Colocar:
   return

//#####################################
//Subrutina que aplica el Complemento a 2 a una trama
//de 22bits
//#####################################
COMP_2:
   comf   Byte1_BCD,F            //Complemento a uno a BYTE_BCD
   comf   Byte2_BCD,F            //Complemento a uno a BYTE2_BCD
   comf   Byte3_BCD,F            //Complemento a uno a BYTE3_BCD
   bcf    Byte3_BCD,7            //Limpiamos los bits de desbordamiento
   bcf    Byte3_BCD,6            //OVL y OVH
   incf   Byte1_BCD,F            //BYTE_BCD + 1  //Aplicar complemento a DOS
   btfss  STATUS,C               //SI(Carry=1)
   goto   FIN_COMP_2
```

```
   incf   Byte2_BCD,F              //BYTE2_BCD + 1
   btfss  STATUS,C                 // SI(Carry=1)
   goto   FIN_COMP_2
   incf   Byte3_BCD,F              //BYTE3_BCD + 1
                                   //FINSI
FIN_COMP_2:
   return
#ENDASM

//*****************************************************************************
//#####################################
//TABLA_BCD
//Hace uso de la librería "LCD" para poder enviar los respectivos
//símbolos a imprimir en la LCD
//También contiene una rutina que manda imprimir toda la trama
//de 22bits
//#####################################

//#####################################
//Subrutina que manda imprimir toda la trama
//#####################################
#ASM
Imprime_Codigo_Decimal:
   movff  SIGNO,NumBCD
   call   Tabla_LCD
   movff  UNIMILL,NumBCD
   call   Tabla_LCD
   movff  CENMIL,NumBCD
   call   Tabla_LCD
   movff  DECMIL,NumBCD
   call   Tabla_LCD
   movff  UNIMIL,NumBCD
   call   Tabla_LCD
   movff  CEN,NumBCD
   call   Tabla_LCD
   movff  DEC,NumBCD
   call   Tabla_LCD
   movff  UNI,NumBCD
   call   Tabla_LCD
   return
#ENDASM
//#####################################
//Subrutina que cambia los valores de BCD a constantes
//que se pueden visualizar en la LCD
//#####################################
#ASM
Tabla_LCD:
T_LCD0:
   movlw   0x00
   cpfseq   NumBCD
   bra      T_LCD1
   movlw   '0'
   call      ImprimeLCD
   return
```

```
T_LCD1:
   movlw   0x01
   cpfseq   NumBCD
   bra      T_LCD2
   movlw   '1'
   call     ImprimeLCD
   return
T_LCD2:
   movlw   0x02
   cpfseq   NumBCD
   bra      T_LCD3
   movlw   '2'
   call     ImprimeLCD
   return
T_LCD3:
   movlw   0x03
   cpfseq   NumBCD
   bra      T_LCD4
   movlw   '3'
   call     ImprimeLCD
   return
T_LCD4:
   movlw   0x04
   cpfseq   NumBCD
   bra      T_LCD5
   movlw   '4'
   call     ImprimeLCD
   return
T_LCD5:
   movlw   0x05
   cpfseq   NumBCD
   bra      T_LCD6
   movlw   '5'
   call     ImprimeLCD
   return
T_LCD6:
   movlw   0x06
   cpfseq   NumBCD
   bra      T_LCD7
   movlw   '6'
   call     ImprimeLCD
   return
T_LCD7:
   movlw   0x07
   cpfseq   NumBCD
   bra      T_LCD8
   movlw   '7'
   call     ImprimeLCD
   return
T_LCD8:
   movlw   0x08
   cpfseq   NumBCD
   bra      T_LCD9
   movlw   '8'
```

```
    call    ImprimeLCD
    return
T_LCD9:
    movlw   0x09
    cpfseq  NumBCD
    bra     T_LCDA
    movlw   '9'
    call    ImprimeLCD
    return
T_LCDA:
    movlw   0x0A
    cpfseq  NumBCD
    bra     T_LCDB
    movlw   ' '
    call    ImprimeLCD
    return
T_LCDB:
    movlw   0x0B
    cpfseq  NumBCD
    bra     T_LCDC
    movlw   '-'
    call    ImprimeLCD
    return
T_LCDC:
    movlw   0x0C
    cpfseq  NumBCD
    bra     T_LCDD
    movlw   '>'
    call    ImprimeLCD
    return
T_LCDD:
    movlw   0x0D
    cpfseq  NumBCD
    bra     T_LCDE
    movlw   '<'
    call    ImprimeLCD
    return
T_LCDE:
    movlw   0x0E
    cpfseq  NumBCD
    bra     FIN_Tabla_LCD
    movlw   'E'
    call    ImprimeLCD
    return
FIN_Tabla_LCD:
    return
#ENDASM
```

```
//*********************************************
//####################################
//SUBRUTINAS DE RETARDOS
//####################################
#ASM
//Para 200us requiere 800cm
//[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
//igualando la ecuación con 800cm
//10+774L=800 -> L=(800-10)/774=1.02 -> L=1
//L=1 -> 784cm -> 196us ; se compensa 4us con llamado al retardo 4us
retardo_200us:          //2cm     LLAMADO
  movlw  1              //1cm     CARGAR L
  movwf  CONTA_L        //1cm
  call   BucleAnidado1  //BA1     LLAMAR BA1
  call   retardo_4us    //compensación
  return                //2cm     REGRESAR

//Para 4us requiere 16cm entonces:
//[2+1+1+BB+2] = 6+BB = 6+(4+3K) = 10+3K
//igualando la ecuación con 16cm
//10+3K=16; -> K=(16-10)/3 =2; -> K=2
//con k=2 ->16cm ->4us;
retardo_4us:            //2cm     LLAMADO
  movlw  2              //1cm     CARGAR K
  movwf  CONTA_K        //1cm
  call   BucleBasico    //(BB)    LLAMAR BB
  return                //2cm     REGRESAR

//Para 5000us requiere 20000cm
//[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
//igualando la ecuación con 20000cm
//10+774L=20000 -> L=(20000-10)/774=25.8 -> L=25
//L=25 -> 19360cm -> 4840us ; compensa 160us con llamado al retardo 100,50,10us
retardo_5ms:            //2cm     LLAMADO
  movlw  25             //1cm     CARGAR L
  movwf  CONTA_L        //1cm
  call   BucleAnidado1  //BA1     LLAMAR BA1
  call   retardo_100us  //compensación
  call   retardo_50us
  call   retardo_10us
  return                //2cm     REGRESAR

//Para 100us requiere 400cm
//10+3K=400; -> K=(400-10)/3 =130; -> K=130
//con K=130 -> 400cm -> 100us
retardo_100us:          //2cm     LLAMADO
  movlw  130            //1cm     CARGAR K
  movwf  CONTA_K        //1cm
  call   BucleBasico    //BBcm    LLAMAR BB
  return                //2cm     REGRESAR
```

```
//Para 50us requiere 200cm
//10+3K=200; -> K=(200-10)/3 =63.3; -> K=63
//con K=63 -> 199cm -> 49.75us; se compensa .25us con 1cm
retardo_50us:          //2cm    LLAMADO
  movlw  63            //1cm    CARGAR K
  movwf  CONTA_K       //1cm
  call   BucleBasico   //BBcm   LLAMAR BB
  nop                  //compensación
  return               //2cm    REGRESAR

//Para 10us requiere 40cm
//10+3K=40; -> K=(40-10)/3 =10; -> K=10
//con K=10 ->40cm ->10us;
retardo_10us:          //2cm    LLAMADO
  movlw  10            //1cm    CARGAR K
  movwf  CONTA_K       //1cm
  call   BucleBasico   //BBcm   LLAMAR BB
  return               //2cm    REGRESAR

//Para 20 000us requiere 80 000cm
//[2+1+1+BA1+2] = [6+BA1] = 6+(4+774L) = 10+774L
//igualando la ecuación con 80 000cm
//10+774L=80 000 -> L=(80 000-10)/774=103.3 -> L=103
//L=103 -> 79732cm -> 19 933us ; se compensa 67us con llamado al retardo 50,10,5,2us
retardo_20ms:          //2cm    LLAMADO
  movlw  103           //1cm    CARGAR L
  movwf  CONTA_L       //1cm
  call   BucleAnidado1 //BA1    LLAMAR BA1
  call   retardo_50us  //compensación
  call   retardo_10us
  call   retardo_5us
  call   retardo_2us
  return               //2cm    REGRESAR

//Para 5us requiere 20cm
//10+3K=20; -> K=(20-10)/3 =3.3; -> K=3
//con K=3 ->19cm ->4.75us; se compensa .25us con 1cm
retardo_5us:           //2cm    LLAMADO
  movlw  3             //1cm    CARGAR K
  movwf  CONTA_K       //1cm
  call   BucleBasico   //BBcm   LLAMAR BB
  nop                  //Compensación
  return               //2cm    REGRESAR

//Un retardo de 2us requiere de 8cm
retardo_2us:           //2cm    LLAMADO
  goto   ret2us1       //2cm
ret2us1: goto  ret2us2 //2cm
ret2us2: return        //2cm    REGRESAR
```

```
//BUCLE BASICO
//su representación algebraica es la siguiente:
//[2+(k-1)+3+(k-1)2+2] = 7+(k-1)3 = 4+3K
//BucleBasico = BB = 4+3K
BucleBasico:            //2cm     LLAMADO
                         //HAZ
  decfsz  CONTA_K       //(k-1)cm+3cm    DECREMENTAR conta_k
  goto  BucleBasico     //(k-1)2cm     MIENTRAS (conta_k=0?)
  return                //2cm     REGRESAR

//BUCLE ANIDADO1
//su representación algebraica es la siguiente:
//[2+1L+1L+BB(L)+(L-1)+3+(L-1)2+2] = 7+2L+BB(L)+(L-1)3
//7+2L+3L-3+BB(L) = 4+5L+BB(L); sustituyendo BB con K=255
//+5L+(4+3(255))(L)=4+5L+769L = 4+774L
//BucleAnidado1 = BA1 = 4+774L
BucleAnidado1:          //2cm     LLAMADO
                        //HAZ
  movlw  255            //1cm(L)      CARGAR conta_k
  movwf  CONTA_K        //1cm(L)
  call  BucleBasico     //(BB)(L)      LLAMAR BucleBasico
  decfsz  CONTA_L       //(L-1)cm + 3cm    DECREMENTAR conta_L
  goto  BucleAnidado1   //(L-1)2cm     MIENTRAS (conta_L=0?)
  return                //2cm     REGRESAR
#ENDASM

}                       //FIN DEL PROGRAMA PRINCIPAL (CIERRA MAIN)
```

Finalmente, a grabar el programa en el Microcontrolador, montarlo en el circuito del protoboard y observar, su funcionamiento.

El primer paso, conectar el circuito, y ya que detectó nuestro dispositivo y que lo registro como un puerto serial (COM4 normalmente), entonces podemos abrir el hyperterminal.

Para abrir hyperterminal del Windows es siguiendo la ruta desde el escritorio, en la barra de tareas: Inicio > Todos los Programas > Accesorios > Comunicaciones > Hyperterminal.

Aparecerá la ventana "Información de comunicación", no importa la configuración que se de ahí. Despues aparece la ventana "Opciones de teléfono y modem", tampoco importa la configuración, aparecerá la ventana descripción de la comunicación, damos un nombre y escojemos cualquier icono. Despues aparece "conectar a" y no necesitamos configurar, damos aceptar y si aparece una ventana que dice conectar, le damos cancelar. Ahora frente a la ventana del hyperterminal, en la barra del menú nos vamos a Archivo > Propiedades. Aparece la ventana respectiva de propiedades, entonces en la ficha conectar a, en la opción path que dice "Conectar usando" escojemos el puerto serial de nuestro dispositivo (COM4 normalmente). Después esta un botón que dice configurar, le damos click y configuramos según las características de nuestro dispositivo, que son:

- Bits por segundo 9600
- Bits de datos 8
- Paridad Ninguno
- Bits de parada 1
- Control de flujo Hardware

Dar click en el icono llamar (teléfono), y se ve en la ventana la trama que estamos enviando.

3. Ajustes y Pruebas del Sistema

3.1. MULTICANALIZACIÓN Y EL LABVIEW

Para el uso final de este dispositivo, se tienen que hacer unas modificaciones en el programa y hardware para que haga juego con un programa para la PC, diseñado para trabajar con comunicación serial y que posee características especiales para poder trabajar con ocho canales de adquisición.

Este sistema se complementará con una interface en la PC con el programa Labview, que fue elaborado en el programa de Apoyo a Proyectos para la Innovación y Mejoramiento de la Enseñanza (PAPIME). Pero para que pueda ser utilizado, se agrega básicamente un conteo para direccionar ocho canales (switch) con dos multiplexores y la trama se debe enviar en hexadecimal. La elaboración de este programa gráfico no está incluido, ya que los objetivos se delimitan desde un inicio a la implementación del sistema de adquisición, el ajuste es solo para poder usarlo en un programa determinado para PC, como es en este caso el Labview, aunque también es posible hacer interface en otros programas que cuenten con protocolo de comunicación serial.

Otro detalle que se le agregará es que el PIC pueda detectar la conexión del puerto USB a la PC, para poder trabajar con la PC, o sin la PC y un botón de pausa para poder leer las lecturas obtenidas, en especial del LCD. El diagrama eléctrico de los últimos ajustes en el hardware, se muestran en la figura (3.1 – 1).

Para mayores detalles sobre los multiplexores analógicos MC14051BCP, consultar su respectivo manual. En el Apéndice B se propone un resumen con las características más importantes sobre este integrado.

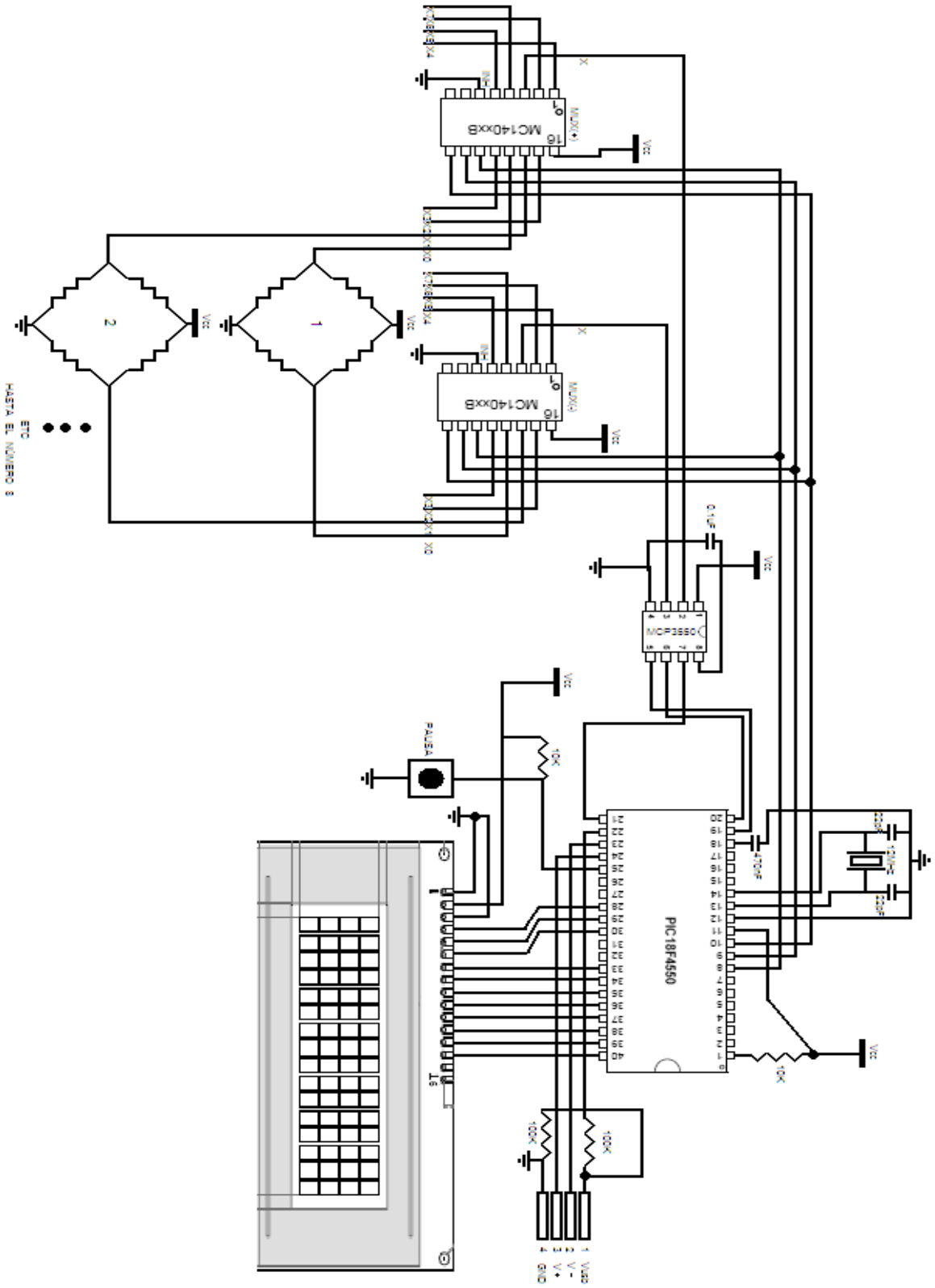


Figura (3.1 - 1) Sistema multicanal con dos multiplexores analógicos de 8 entradas y una salida conectadas al convertidor A/D y al Microcontrolador.

La rutina principal en C cambiará bastante, y se hace un ajuste al programa del Convertidor Analógico Digital (CAD). Esto es para aprovechar tiempo mientras el convertidor está tomando la muestra. El cambio en el programa fue el siguiente, nótese como solo se dividió en dos partes, y se usa esta vez el modo conversión continua.

```
#ASM
Inicializa_CC_MCP:
   bcf    CS                          //INICIAR la conversion Pulso low en Chip Select
   call   retardo_200us
   return
Rx_MCP3550_CC:
  bsf    CS                          //ACTIVAR Single Conversion Mode
   call   Busy_MCP3550                //MIENTRAS no hay muestra
  bcf    CS                          //INDICAR que tomaremos la muestra

   call   CLKByte                     //TOMAR Byte
   movff  Byte_MCP,Byte3_MCP          //CARGAR variable (MCPByte_1=MCPByte)
   call   CLKByte                     //  TOMAR Byte_2
   movff  Byte_MCP,Byte2_MCP
   call   CLKByte                     //  TOMAR Byte_3
   movff  Byte_MCP,Byte1_MCP
   bsf    CS                          //ENTRAR Modo ShutDown
  return                             //FINSI

//Recordemos que el busy para CC es diferente
Busy_MCP3550:                        //MIENTRAS(RDY=1)
   btfsc  RDY                         //FIN MIENTRAS
   goto   Busy_MCP3550
   return
#ENDASM
```

El resto de código es parecido, los cambios son pocos pero muy significativos. Ahora es posible pedir la muestra y después tomarla cuando sea necesario, con esto es más eficiente nuestro sistema.

Para el programa principal se diseña el siguiente pseudocódigo muy sencillo. Dividimos la operación en dos, con PC o sin PC.

```
SI (Esta conectado?) ENTONCES
        FUNCION CON PC
SINO
        FUNCION SIN PC
FINSI
```

Esta estructura irá dentro de la estructura WHILE que se repite infinitamente.

Sigue definir que características tendrá una función respecto de la otra. Como uno de los objetivos de la ingeniería es hacer las cosas lo más eficiente posible, entonces se toma la decisión de que mientras se esté trabajando con la PC, no necesitamos utilizar el LCD y en consecuencia el codificador, esto hará muy rápida la operación con la PC. Solamente cuando no se conecte a la PC, utilizaremos el codificador y el LCD. En lo que son parecidas

las dos funciones es en el uso de la multicanalización, ya que ambas requieren del manejo de los multiplexores.

Para hacer el control de la multicanalización se debe saber cómo opera el multiplexor, y resulta bien sencillo. Si por el bus de control le damos la dirección 000, entonces deja pasar la señal proveniente del switch 0 la cual se mostrará a la salida, si indicamos el valor 001, entonces a la salida mostrará la señal proveniente del switch 1 y así sucesivamente. Entonces será que para 000, switch0, muestra1, para 001, switch1, muestra2 ….hasta 111, switch7, muestra8.

Entonces se realiza el diseño del pseudocódigo, en el caso de la función con PC, se pueden usar los recursos que nos presta C haciendo uso de una cadena de caracteres. Y se debe ser cuidadoso, inicializando una muestra para no quedarnos en un bucle vicioso en el chequeo del busy del Convertidor Analógico Digital.

```
CONTENEDORES
        iConteo=0
FINCONTENEDORES

//Cadena de caracteres
IDENTIFICADOR [8]= [s, t, u, v, w, x, y z]

SWITCH 0
INICIAR Muestra

PARA(iconteo=0; iConteo<=7; ++ iConteo)
        RECIBIR muestra
        SWITCH iConteo+1
        INICIAR Muestra
        ENVIAR trama de 22bits a la PC
        ENVIAR IDENTIFICADOR (iConteo) a la PC
FINPARA
```

Ahora la función sin PC. En este caso se opta por incluir un proceso en ensamblador, ya que esto agiliza el proceso con el LCD. De la misma forma se anticipa la primera muestra del canal 0. El diseño del pseudocódigo queda:

```
SWITCH 0
INICIAR Muestra 0

RECIBIR muestra 0
SWITCH 1
INICIAR Muestra 1
CODIFICAR Muestra 0 a BCD
POSICION para muestra 0
IMPRIMIR código en el LCD
```

```
RECIBIR muestra 1
SWITCH 2
INICIAR Muestra 2
CODIFICAR Muestra 1 a BCD
POSICION para muestra 1
IMPRIMIR código en el LCD
.
.
.

RECIBIR muestra 8
SWITCH 0
INICIAR Muestra 0
CODIFICAR Muestra 8 a BCD
POSICION para muestra 8
IMPRIMIR código en el LCD
```

Este proceso se repetirá una y otra vez, es similar al anterior, solo que se usan más líneas de código, y si se utiliza el codificador a BCD, pero el mayor beneficio es en el tiempo de ejecución.

Ahora el pseudocódigo del programa principal visto de una forma general.

```
INICIALIZAR Periféricos, USB, LCD,MCP3550
MIENTRAS (1)                    //Se repite en un bucle infinito
SI (conectado?)ENTONCES
        CONECTAR dispositivo a la PC
        MIENTRAS (PC este lista)
                MIENTRAS (Dispositico este enumerado)
                        SI (PAUSA=1) ENTONCES
                                ENVIAR Multicanalización para PC
                        FINSI
                FINMIENTRAS
        FINMIENTRAS
SINO
        NO CONECTAR dispositivo a la PC
        SI (PAUSA=1) ENTONCES
                ENVIAR Multicanalización para LCD
        FINSI
FINSI
FINMIENTRAS                     //Llave que cierra el bucle infinito
```

Antes de mostrar cómo queda el código, es necesario discernir lo siguiente: Para que la función **usb_attached(),** y las que dependen de ella, funcionen correctamente, esto es que envíen el 0 o 1 que debe devolver, es necesario definir el pin sensor de conexión de lo contrario esta función siempre regresara un 1, lo cual compromete el resultado esperado. Pues al regresar siempre 1 se ejecutara siempre el bloque o proceso que sirve cuando está conectado nuestro dispositivo a la PC.

Entonces, se debe definir este pin en la cabecera de nuestro programa, de la forma siguiente:

```
#DEFINE USB_CON_SENSE_PIN PIN_A0      //Debe declararse antes la siguiente
                                      //librería
                                      //Pin A0 como sensor de conexión USB
#INCLUDE ".\include\usb_cdc.h"        //Descripción de funciones cdc del USB.
```

Hay que observar que se puede cambiar el atributo PIN_A0 por cualquier otro pin que sirva como entrada digital. Otro ejemplo podría ser USB_CON_SENSE_PIN **PIN_C5** o según convenga en nuestro diseño del circuito.

En los ejercicios de ejemplo que proporciona el PIC C, recomienda que para el pin sensor tomemos en cuenta la siguiente conexión en caso de definirlo.

```
////////////////////////////////////////////////////////////////////////
//
//Sin el pin sensor de conexión, no sabrá si el dispositivo
//se desconecta.
//     (El sensor de conexión debería tener este aspecto)
//                          100k
//         VBUS-----+----/\/\/\/\----- (I/O PIN del PIC)
//                  |
//                  +----/\/\/\/\-----Tierra
//                          100k
//      (Donde VBUS es pin1 del conector USB)
//
////////////////////////////////////////////////////////////////////////
```

El siguiente código es solamente el que se agregó para ajustar el programa principal.

```
;_____________________________________________________________________________

//DEFINICION PARA EL PROGRAMA PRINCIPAL
#BIT  Pausa = PORTD.3                 //Botón de pausa en el PIN

//PROGRAMA PRINCIPAL
void main (void)
{
  //Cadena de caracteres
  byte const IDENTIFICADOR[8]={'s','t','u','v','w','x','y','z'};
  //CONFIGURACIONES DE LOS PERIFERICOS DEL PIC y MÓDULOS EXTERNOS
  #ASM
  call  ConfigPuertos_MCP
  call  ConfigPuertos_LCD
  call  Inicializa_LCD
  call  Universidad                   //Presentación
  #ENDASM
  delay_ms(1000);                     //Duración de la presentación 1s
  #ASM
  call  Limpiar
  #ENDASM
```

```
  usb_init_cs();                                  //Reinicia e inicializa el periférico USB
  usb_cdc_init();                                 //Configura comunicación serial

  iConteo=0;
  output_E(iConteo);
  #ASM
  call  Inicializa_CC_MCP
  #ENDASM
WHILE (true)
{
  IF (usb_attached())      //Si está conectado
  {
    #ASM
    call  Conectando
    #ENDASM
    usb_task();                                   //Si está conectado,  habilita el módulo USB
    WHILE (usb_cdc_connected())                   //El programa está listo para trabajar
    {
      #ASM
      call  Limpiar
      call  Universidad
      #ENDASM
      WHILE (usb_enumerated())                    //Regresa 1 si ha sido enumerado
      {
        IF (Pausa==1)
        {
          //MUESTRAS PARA PC
          FOR (iConteo=0; iconteo<=7; ++iConteo)  //Mientras sea <=7
          {
            #ASM
            call  Rx_MCP3550_CC                   //Recibe la muestra
            #ENDASM
            output_E(iConteo+1);                  //Cambia switch
            #ASM
            call  Inicializa_CC_MCP               //Inicializa nuevo cambio switch
            #ENDASM
            printf(usb_cdc_putc,"%X",Byte3_MCP);
            printf(usb_cdc_putc,"%X",Byte2_MCP);
            printf(usb_cdc_putc,"%X",Byte1_MCP);
            usb_cdc_putc(IDENTIFICADOR[iConteo]);
          }
        }
      }
    }
  }
  ELSE    //TRABAJA SIN CONEXIÓN CON LA PC
  {
    usb_task();                                   //Si no está conectado deshabilita el módulo USB
    IF (Pausa==1)
    {
      //MUESTRAS PARA LCD
      #ASM
      call  Rx_MCP3550_CC                 //Recibe muestra 1
```

```
movlw 1                          //Switch 1
movwf PORTE
call  PreparaMuestra             //Inicia nueva y Convierte Muestra Anterior
call  Linea_1_1                  //Posición de la muestra 1
call  Imprime_Codigo_Decimal     //Imprime los dígitos en LCD

call  Rx_MCP3550_CC              //Recibe muestra 2
movlw 2                          //Switch 2
movwf PORTE
call  PreparaMuestra             //Inicia nueva y Convierte Muestra Anterior
call  Linea_2_1                  //Posición de la muestra 2
call  Imprime_Codigo_Decimal     //Imprime los dígitos en LCD

call  Rx_MCP3550_CC              //Recibe muestra 3
movlw 3                          //Switch 3
movwf PORTE
call  PreparaMuestra             //Toma y Convierte Muestra
call  Linea_3_1                  //Posición de la muestra 3
call  Imprime_Codigo_Decimal     //Imprime los dígitos en LCD

call  Rx_MCP3550_CC              //Recibe muestra 4
movlw 4                          //Switch 4
movwf PORTE
call  PreparaMuestra             //Toma y Convierte Muestra
call  Linea_4_1                   //Posición de la muestra 4
call  Imprime_Codigo_Decimal     //Imprime los dígitos en LCD

call  Rx_MCP3550_CC              //Recibe muestra 5
movlw 5                          //Switch 5
movwf PORTE
call  PreparaMuestra             //Toma y Convierte Muestra
movlw 0x8A                       //Posición de la muestra 5
call  PosicionDDRAM
call  Imprime_Codigo_Decimal     //Imprime los dígitos en LCD

call  Rx_MCP3550_CC              //Recibe muestra 6
movlw 6                          //Switch 6
movwf PORTE
call  PreparaMuestra             //Toma y Convierte Muestra
movlw 0xCA                       //Posición de la muestra 6
call  PosicionDDRAM
call  Imprime_Codigo_Decimal     //Imprime los dígitos en LCD

call  Rx_MCP3550_CC              //Recibe muestra 7
movlw 7                          //Switch 7
movwf PORTE
call  PreparaMuestra             //Toma y Convierte Muestra
movlw 0x9E                       //Posición de la muestra 7
call  PosicionDDRAM
call  Imprime_Codigo_Decimal     //Imprime los dígitos en LCD

call  Rx_MCP3550_CC              //Recibe muestra 8
movlw 0                          //Switch 7
movwf PORTE
```

```
      call  PreparaMuestra                    //Toma y Convierte Muestra
      movlw 0xDE                              //Posición de la muestra 8
      call  PosicionDDRAM
      call  Imprime_Codigo_Decimal            //Imprime los dígitos en LCD
      #ENDASM
    }
  }
}
//*****************************************************************************
//ENSAMBLADOR
//SUBRUTINA QUE Prepara la muestra
#ASM
PreparaMuestra:
call   Inicializa_CC_MCP                      //Prepara otra muestra
movff  Byte3_MCP,Byte3_BCD                    //Copiamos la muestra en los
movff  Byte2_MCP,Byte2_BCD                    //buffer de entrada del codificador
movff  Byte1_MCP,Byte1_BCD
call   Convierte_BCD                          //MANDAR convertir
return
#ENDASM

//SUBRUTINAS DE MENSAJES
#ASM
Limpiar:
call    ModoBusyLCD
call    ModoComandoLCD
 movlw   ClearDisplay
call    EnviaDatLCD
return

Conectando:
call  Linea_1_1
movlw 'C'
call  ImprimeLCD
movlw 'o'
call  ImprimeLCD
movlw 'n'
call  ImprimeLCD
movlw 'e'
call  ImprimeLCD
movlw 'c'
call  ImprimeLCD
movlw 't'
call  ImprimeLCD
movlw 'a'
call  ImprimeLCD
movlw 'n'
call  ImprimeLCD
movlw 'd'
call  ImprimeLCD
movlw 'o'
call  ImprimeLCD
return
```

```
Universidad:
call  Linea_1_1
movlw   'U'
call    ImprimeLCD
movlw   'N'
call    ImprimeLCD
movlw   'I'
call    ImprimeLCD
movlw   'V'
call    ImprimeLCD
movlw   'E'
call    ImprimeLCD
movlw   'R'
call    ImprimeLCD
movlw   'S'
call    ImprimeLCD
movlw   'I'
call    ImprimeLCD
movlw   'D'
call    ImprimeLCD
movlw   'A'
call    ImprimeLCD
movlw   'D'
call    ImprimeLCD

call    Linea_2_1
movlw   'N'
call    ImprimeLCD
movlw   'A'
call    ImprimeLCD
movlw   'C'
call    ImprimeLCD
movlw   'I'
all     ImprimeLCD
movlw   'O'
call    ImprimeLCD
movlw   'N'
call    ImprimeLCD
movlw   'A'
call    ImprimeLCD
movlw   'L'
call    ImprimeLCD
call    Linea_3_1
movlw   'A'
call    ImprimeLCD
movlw   'U'
call    ImprimeLCD
movlw   'T'
call    ImprimeLCD
movlw   'O'
call    ImprimeLCD
movlw   'N'
call    ImprimeLCD
movlw   'O'
```

```
call    ImprimeLCD
movlw   'M'
call    ImprimeLCD
movlw   'A'
call    ImprimeLCD
call    Linea_4_1
movlw   'D'
call    ImprimeLCD
movlw   'E'
call    ImprimeLCD
movlw   ' '
call    ImprimeLCD
movlw   'M'
call    ImprimeLCD
movlw   'E'
call    ImprimeLCD
movlw   'X'
call    ImprimeLCD
movlw   'I'
call    ImprimeLCD
movlw   'C'
call    ImprimeLCD
movlw   'O'
call    ImprimeLCD
return
#ENDASM
;_____________________________________________________________________________________________
```

3.2. DESCRIPCIÓN DEL PROGRAMA DISEÑADO EN EL LABVIEW

Este programa contiene varios módulos que cumplen con tareas específicas, que corresponden al diseño de diagramas a bloques que realizan las operaciones o configuraciones necesarias, para poderlas mostrar en sus respectivos paneles frontales.

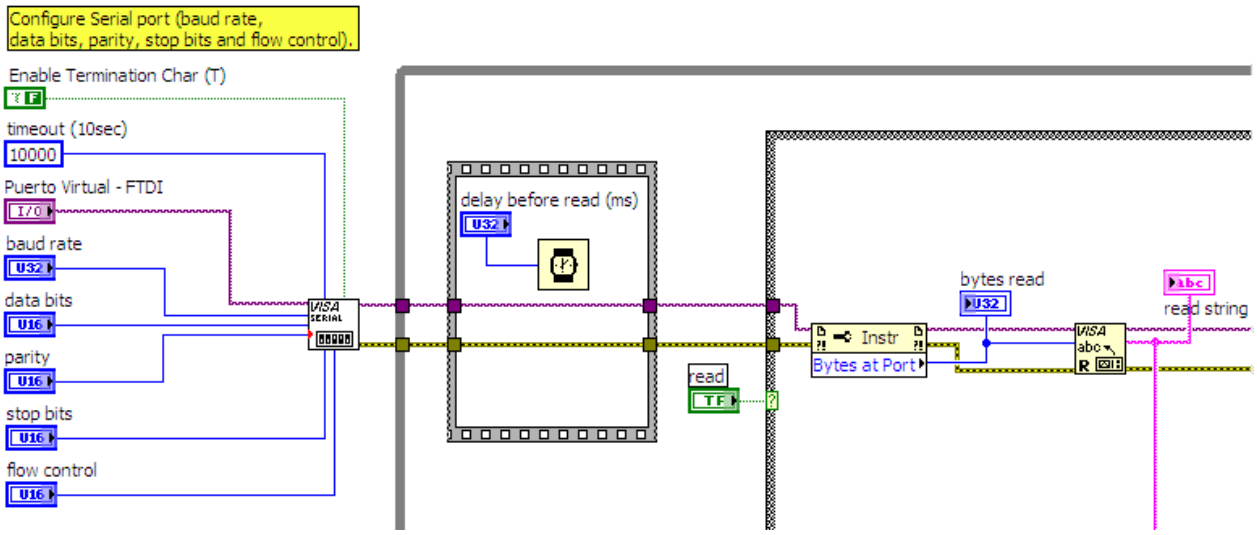
Módulo de comunicación.

El módulo de comunicación contiene los siguientes controles y ajustes:

- Asignación del puerto virtual FTDI, COM4.
- Velocidad de transmisión de 9600 baudios.
- Bits de datos 8.
- Paridad, no.
- Bit de paro 1.
- Control de flujo, no.
- Tiempo de retardo de 1000ms.

Se asigna un control para leer los datos provenientes del hardware, que son caracteres que representan números en hexadecimal. Ver la figura (3.2 - 1).

Figura (3.2 - 1). Módulo de comunicación - Panel frontal.



Figura(3.2 -2). Módulo de comunicación - Diagrama a bloques.

Ciclo principal para la multicanalización

Conocemos lo que hace nuestro programa principal en C, esto es, una tramas de tres bytes, comenzando por el byte3 hasta el byte1en hexadecimal, agregando al final un carácter ó identificador. Ahora este ciclo recibe la trama gracias también al módulo de comunicación. Es entonces como este ciclo recibe la trama y pasa por la función “Scan String For Tokens” que determina el identificador o señal (token), que previamente se asigno en el arreglo. Y una vez identificado el caracter que indica a qué canal pertenece el dato enviado, se conecta a una estructura CASE que selecciona si es un número en hexadecimal, a través de la función “Scan From String” o cualquier otro tipo de caracter. En el primer caso el análisis gramatical (parse) comienza decidiendo si es un número positivo o negativo, en el rango de 00000 a 1FFFFF. De no ser así, entonces se resta el número 3FFFFF y el resultado se multiplica por (-1). En cualquier otro caso que no se trate de un número hexadecimal, solo se deja pasar el dato sin ningún procesamiento. Después de la selección, se almacenan los datos en un arreglo que posteriormente se utiliza para realizar la calibración de los datos. Ver la figura (3.2 – 3).

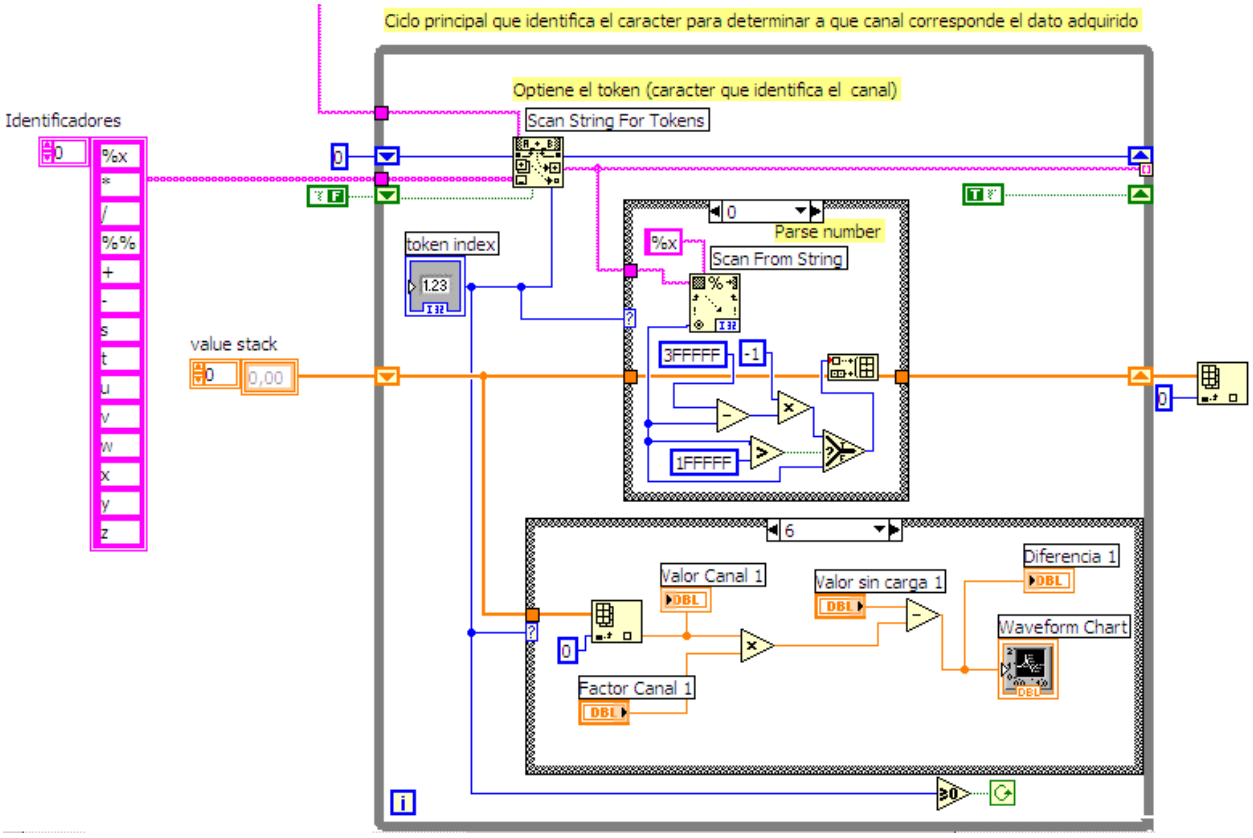


Figura (3.2 - 3). Módulo de multicanalización - Diagrama a bloques.

Módulo de calibración de datos

La calibración de datos se lleva a través de un CASE que selecciona, por medio del índice del indicador ó señal (token), el carácter encontrado, ya que solo existe la posibilidad de identificar un número hexadecimal, o los caracteres ASCII ‘s’, ‘t’, ‘u’, ‘v’, ‘w’, ‘x’, ‘y’, ‘z’, que representan el canal 0 hasta el 7 respectivamente. Una vez identificado el canal que corresponde, se utiliza el valor del proceso anterior contenido en el índice cero del arreglo,

para ser multiplicado por el factor asignado, y posteriormente restado por el valor sin carga. Y por último se gráfica el valor del canal correspondiente.

El módulo de calibración contiene un control numérico que indica el factor de calibración, que es la multiplicación de la medición de un escalar, de tal forma que se puede atenuar o amplificar la medición (figura (3.2 -4)). En este modulo también se hace la calibración a cero, ejecutando una simple diferencia entre la medición sin carga con ella misma. Ver la figura (3.2 – 5).

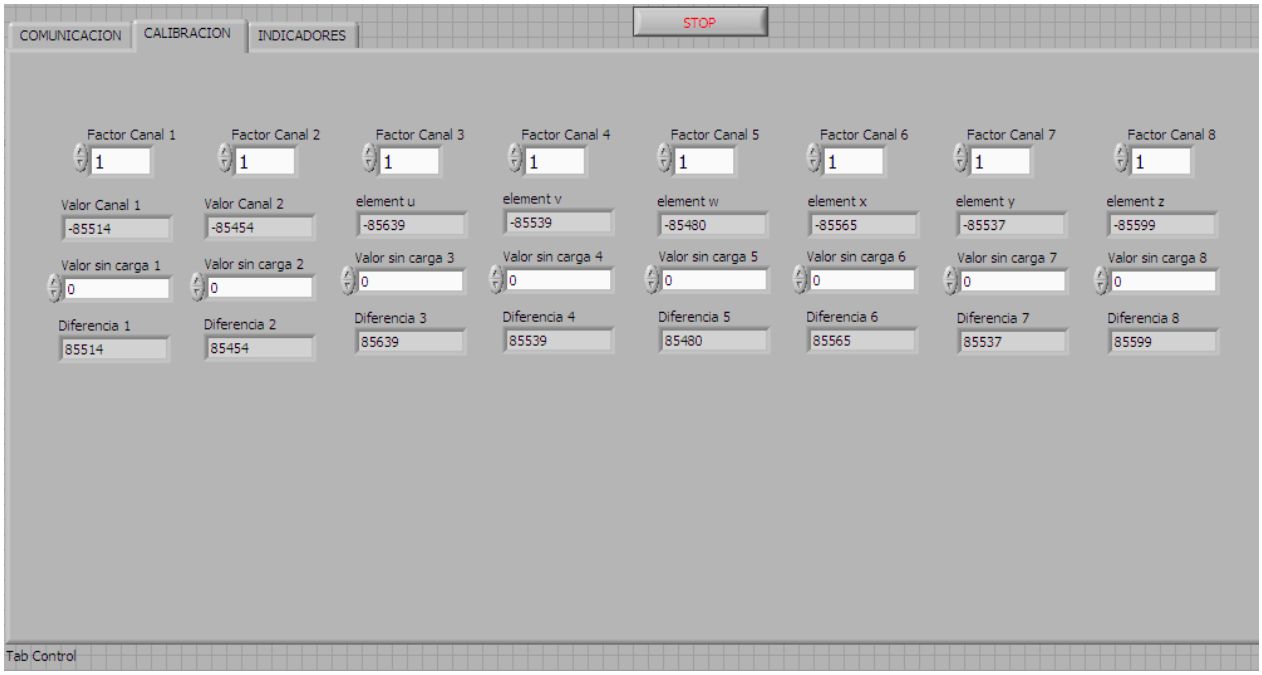


Figura (3.2 - 4). Módulo de calibración - Panel frontal.

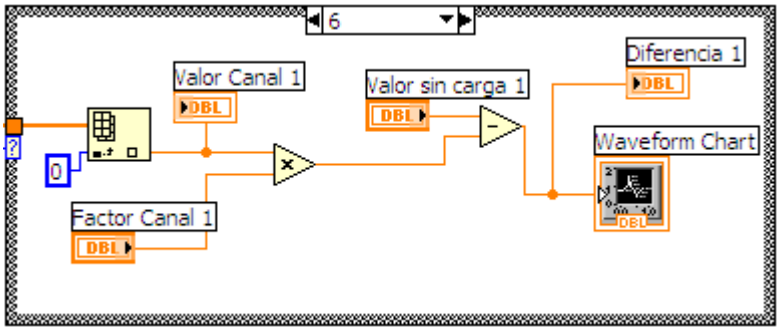


Figura (3.2 - 5). Módulo de calibración - Diagrama a boques.

En la ficha indicadores del panel frontal (Figura 3.2 – 6)), muestra gráficamente la adquisición de datos en su respectivo canal.

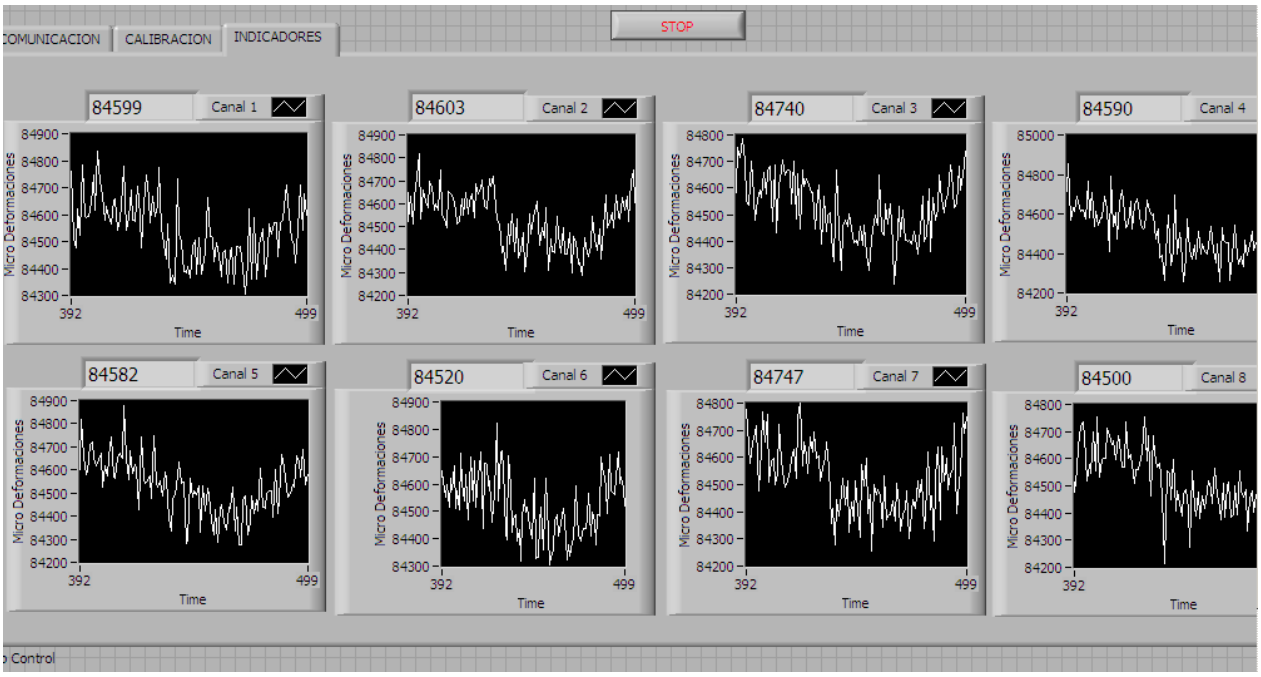


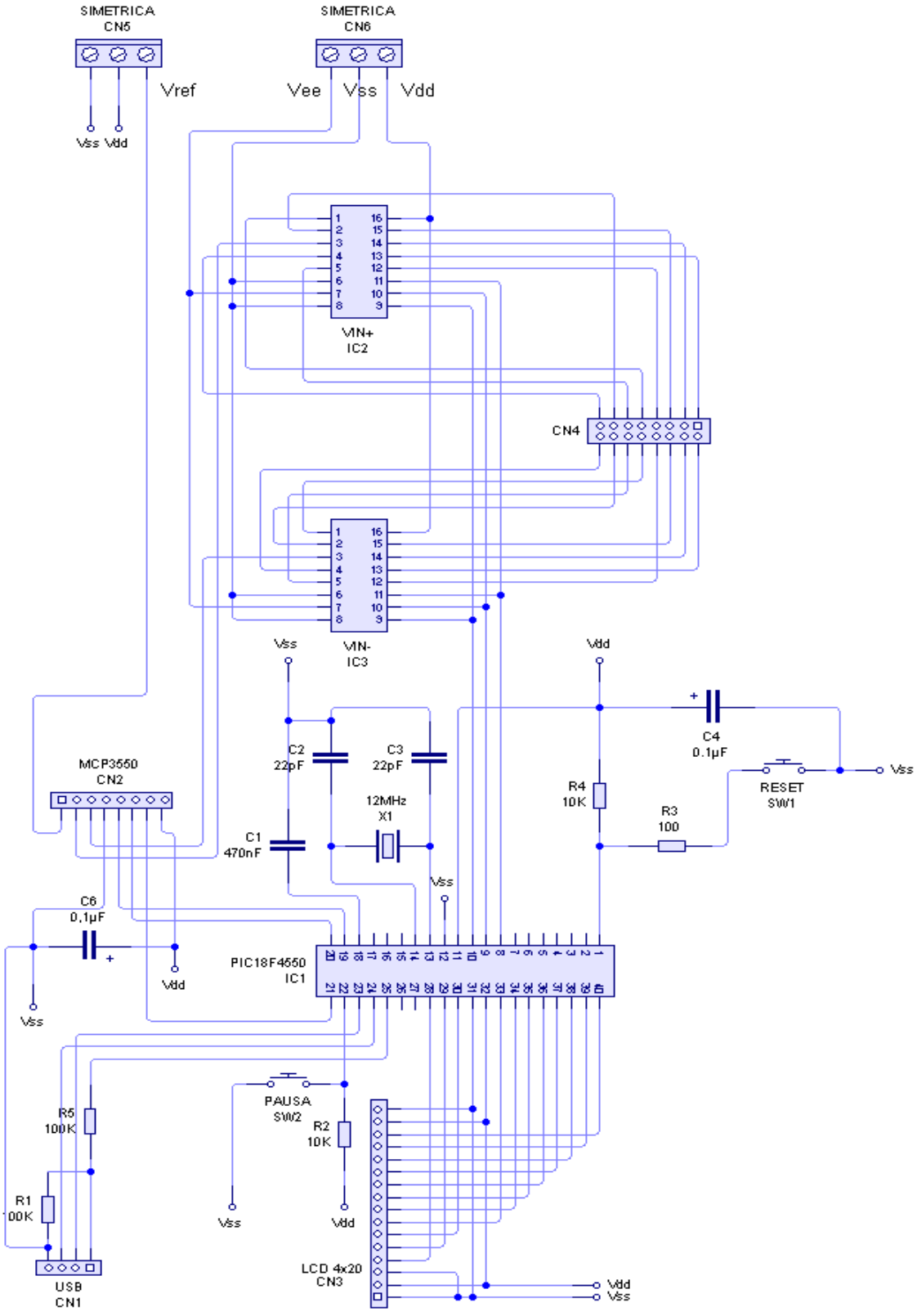
Figura (3.2 - 6). Panel Frontal – Indicadores.

Este programa en Labview se realizó en la versión 8.5, para que pueda ser usado es necesario haber instalado el programa labview 8.5 y un complemento para poder hacer la interface serial. En mi caso instale el complemento Ni LabVIEW 8.5 PDA Module for Windows Mobile. Ya que este complemento instala los drivers NI-VISA que sirven para poder utilizar el protocolo RS 232 interface serial en el Labview.

3.3. PROYECTO FINAL

Falta la presentación, se cuenta con el diagrama eléctrico que sirve para poder crear un diseño en circuito impreso o PCB (printed circuit board). El diseño propuesto para este proyecto en PCB fue el siguiente; que fue creado en PCB Wizard – Professional Edition 3.50, que es un software de manejo muy intuitivo para la creación de PCB. Para la presentación en PCB se propone el diseño de la figura (3.3 – 1), integrando todos los elementos que describe el circuito original.

La figura (3.3 – 1) muestra el diagrama eléctrico del proyecto (primer paso) para la elaboración del PCB final.



Figura(3.3 - 1) Diagrama eléctrico creado en el PCB WIZARD.

LISTA DE COMPONENTES

Es importante contar con la lista de materiales para estimar costos. La tabla (3.3 -1), es un ejemplo propuesto, ya que se pueden variar algunos elementos como los conectores, ya que es posible poner bases de puros Header, o remplazarlos por algún elemento similar.

Nombre	Cantidad
0.1µF Capacitor Electrolítico	2
22pF Capacitor	2
470nF Capacitor	1
100Ω Resistor (1/2W)	1
10KΩ Resistor (1/2W)	2
100KΩ Resistor (1/2W)	2
Cristal Oscilador (12.0 MHz)	1
Push Button	2
8 Pin Header	3
16 Pin Header	1
3 Pin Terminal Block	2
PCB de dos caras 10x10cm	1
Conector USB Hembra tipoB	1

Tabla (3.3.1) Lista de componentes y estimaciones.

Hubo la necesidad de trabajar con ambos lados, ya que los Multiplexores tienen desordenadas las entradas de los canales y provoca que se crucen mucho las líneas, por lo que quedarían muchos puentes y también las líneas deben ser muy delgadas, si se espera obtener un tamaño reducido.

La primer cara es de 10x10 cm, es donde van las componentes, con nombre propuesto caraA figura (3.3 -2), y la segunda también debe ser de 10x10cm es la caraB que pertenece a la figura (3.3 - 3).

Por último, se muestra la Tarjeta de Adquisición de Datos en una vista preliminar ó cara de componentes en la figura (3.3 – 4), realizada en PCB Wizard.

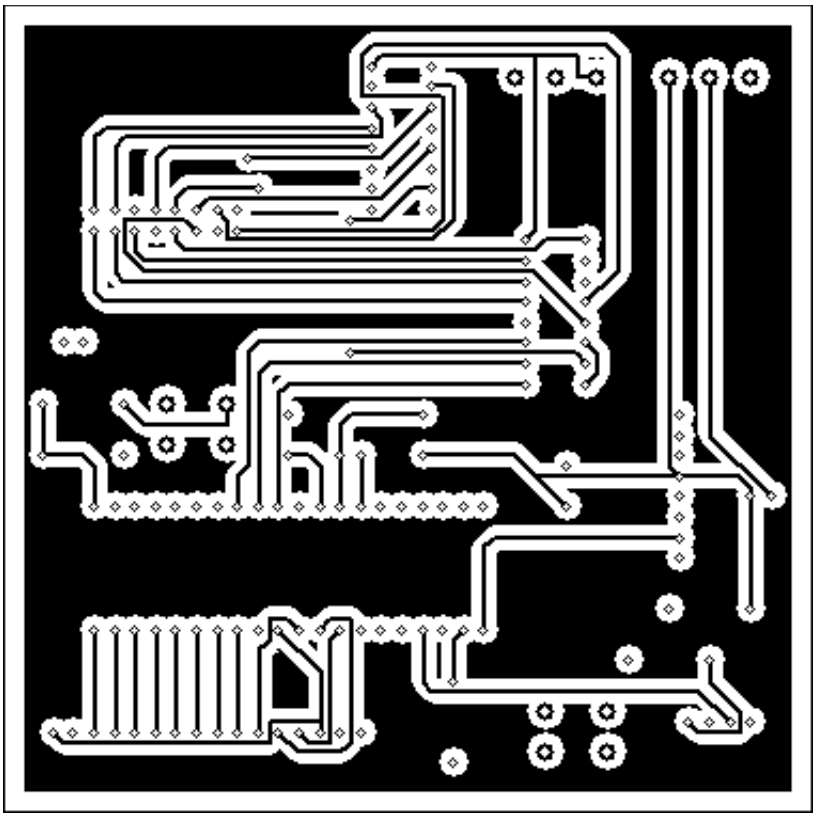


Figura (3.3 – 2) CaraA Placa de circuito Impreso.

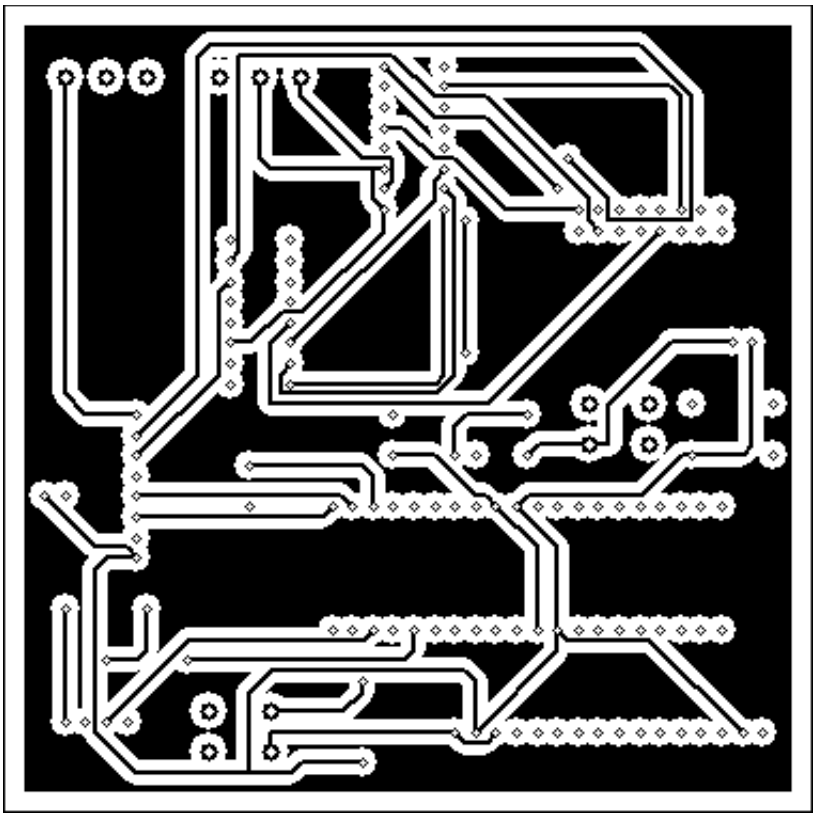
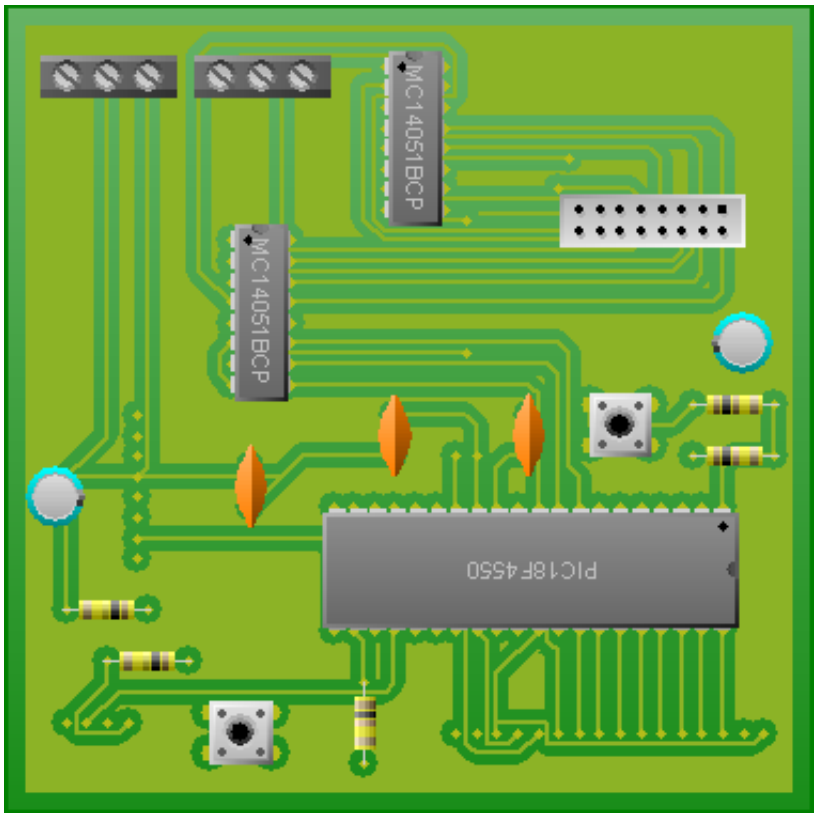
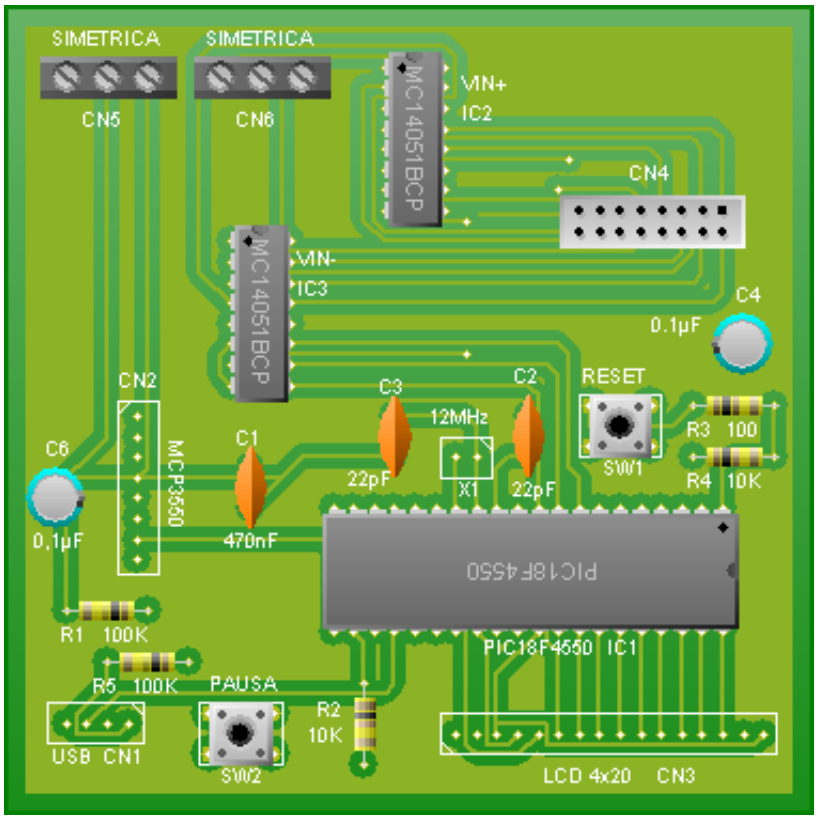


Figura (3.3 – 3) CaraB Placa de circuito Impreso.



a) Vista real.



b) Nombre y denominación de los componentes.

Figura (3.3 – 4) Cara de componentes.

Conclusiones

Para el desarrollo del proyecto sin duda fue mejor utilizar el microcontrolador que tuviera integrado el módulo USB, ya que con modelos anteriores el circuito sería más grande, lo que hizo del circuito uno más reducido, a pesar que se trabajo con el modelo de 40 pin. Entonces en vez de tener un circuito con una etapa extra para el módulo USB, se aprovecha el modelo 18F4550 que ya lo tiene incluido y con dos líneas (V+ y V-) es posible hacer la conexión a la PC.

Lo que hace del convertidor analógico digital el más adecuado no fue la velocidad, ya que modelos anteriores a este pueden obtener varias muestras en menor tiempo. Lo que lo hace atractivo es su muy alta resolución de 22 bits que ni siquiera el convertidor, integrado en el PIC se acerca, y resulta mejor para señales muy pequeñas como es el caso de la galga extensiométrica. Su interface serial es simple y muy rápida, esto también es muy importante ya que otros modelos de convertidor analógico digital, normalmente son paralelos, esto provocaría un diseño más grande. Esto no sucede con el MCP3550/60 ya que con solo tres líneas se controla el dispositivo y obtener una trama de 24bits. También posee algunos filtros que hacen que el convertidor sea muy estable y conserve su precisión.

El protocolo de comunicación USB cdc (comunicación device class) fue la mejor elección, ya que no se necesitaran muchas capas de software para la comunicación con la PC. Tampoco se necesitaron demasiados conocimientos para poder implementar el sistema, fue viable el diseño del programa para el proyecto gracias a las librerías que contiene el CCSC (PIC C). Tambien es más fácil contar con software de protocolo serial RS232 para hacer una comunicación con el proyecto, como es el caso del hyperterminal del Windows. Esto permitió poder hacer pruebas antes de siquiera probarlo con el programa de instrumentación virtual LabView. De esta forma se pudo probar la etapa de la comunicación USB cdc independientemente de las demás etapas.

Algo común durante el desarrollo del proyecto fue que en cada etapa se puede extender bastante cada tema, como es el caso de los protocolos de comunicación USB, de la arquitectura del puerto USB integrado al PIC, el diseño del software USB, incluyendo el driver, algún circuito amplificador como parte del acondicionamiento de la señal para el convertidor A/D, los tipos de arreglos en los pentes wheatstone de las galgas extensiométrícas asi como los tipos de galgas, eso implica una muy profunda depuración del proyecto. Dentro de todo esto, se pudo cumplir con los objetivos logrando implementar una tarjeta de adquisición de 8 canales con un convertidor de alta resolución de 22 bits, con un indicador LCD ya sea para mostrar la trama de los ocho canales, o para interactuar con la PC en un programa de instrumentación virtual, que puede procesar mejor la información (calibrar) o darle otros usos, como elaborar gráficas en programas de hojas de cálculo, etc.

Apéndices

APÉNDICE A

REPERTORIO DE INSTRUCCIONES

LENGUAJE ENSAMBLADOR

TIPOS DE SINTAXIS EN SISTEMAS DE NUMERACIÓN Y CÓDIGO ASCII

SISTEMA	SINTAXIS	EJEMPLO
Binario	b'Número$_2$' B'Número$_2$'	b'00001111' B'00001111'
Octal	o'Número$_8$' O'Número$_8$'	o'352' O'352'
Decimal	d'Número$_{10}$' D'Número$_{10}$' .Número$_{10}$	d'213' D'213' .213
Hexadecimal	h'Número$_{16}$' H'Número$_{16}$' 0xNúmero$_{16}$ Número$_{16}$h Número$_{16}$H	h'7F' H'7F' 0x7F 7Fh 7FH
Código ASCII	a'Caracter' A'Caracter' 'Caracter'	a'N' A'N' 'A'

NOTA:

1. Las constantes alfanuméricas que empiecen por una letra (A,B,C,D,E,F) deben ir precedidas por un cero, para que no sean confundidas con una etiqueta; ejemplo 0BFh.
2. Las constantes pueden ser precedidas por un signo "+" (positivos) o "-" (negativos). Si no se pone signo, se considera positivo por default.

Tabla (1.5 - 1) Constantes alfanuméricas.

DESCRIPCIÓN DEL CAMPO DE CÓDIGO

Campo	Descripción
a	Bit del Acces RAM a=0; BSR se ignora y se usa el área de RAM llamada “Acces RAM” a=1; El área de RAM usada se especifica con BSR
bbb	Para dirección de un bit en un archivo de registro de 8 bit
BSR	Para seleccionar un Banco de RAM (Bank Select Register)
C,DC,Z,OV,N	Bits de Estado del ALU: Carry, Digital Carry, Overflow, Negative
d	Bit de Selección de Destino d=0; El resultado se almacena en WREG d=1; El resultado se almacena en el archivo de registro f
dest	Ya sea el registro WREG o la ubicación del archivo de registro
f	Dirección de 8 bit de archivo de registro (00h a FFh) o designador FSR de 2bit (0h a 3h).
f_s	Dirección de 12 bit de archivo de registro (000h a FFFh). Esta es la dirección de origen
f_d	Dirección de 12 bit de archivo de registro (000h a FFFh). Esta es la dirección de destino
GIE	Bit global, habilitador de Interrupción (Global Interrupt Enable)
k	Campo literal, dato constante o etiqueta (puede ser de 8 bits, 12 bits o un valor de 20-bits).
PC	Contador de programa (Program Counter)
PCL	Byte bajo del contador de programa (Program Counter Low)
PCH	Byte alto del contador de programa (Program Counter High)
PCLATH	Byte alto latch del contador de programa
PCLATU	Byte superior latch del contador de programa
$\overline{PD}$	Bit de Poder bajo (Power Down)
PRODH	Producto de la multiplicación del bit alto
PRODL	Producto de la multiplicación del bit bajo
s	Bit de selección del modo Rápido de Call/Return s=0; No actualizará dentro de los registros sombra s=1;Algunos registros cargan dentro de los registros sombra(modo rápido)
TBLPTR	Puntero Tabla (Table Pointer) de 21 bit, (señala una ubicación de la memoria de programa)
TABLAT	Tabla Latch de 8 bit
$\overline{TO}$	bit Tiempo fuera (Time-out)
TOS	Cima de la pila (Time-of-Stack)
u	No utilizados ó sin cambios
WTD	Timer Watchdog
WREG	Registro de trabajo (Work Register)
x	Condición no importa
z_s	Valor de desplazamiento de 7 bit para el direccionamiento indirecto de los archivos de registro (fuente)
z_d	Valor de desplazamiento de 7 bit para el direccinamiento indirecto de los archivos de registro (destino)

Tabla (1.5 - 2) Descripción de campos de código.

SIMBOLOGÍA UTILIZADA EN LA DESCRIPCIÓN

Símbolo	Descripción
{ }	Argumento opcional
[Texto]	Indica una dirección indexada
(Texto)	El contenido del texto
[Expresión]<n>	Bit n especifica el registro indicado por la expresión del puntero
→	Asignado
< >	Campo del bit de registro
∈	En el conjunto de o pertenece a
Itálicas	Termino definido por el usuario (La fuente es Courier)

Tabla (1.5 - 3) Complemento de instrucciones.

JUEGO DE INSTRUCCIONES

INSTRUCCIONES DE CARGA

Mnemónico		Descripción	Comportamiento	CM	Status	Notas
Código	**Operando**					
Instrucciones de CARGA						
Orientadas a BYTE						
CLRF	f,a	Borra registro	$000h \rightarrow f$	1	Z	2
MOVF	f,d,a	Copia el contenido del registro	$f \rightarrow dest$			
MOVFF	f_s,f_d	Copia el contenido del registro a otro registro	$(f_s) \rightarrow f_d$	2		
MOVWF	f,a	Copia el contenido del WREG a otro registro	$(W) \rightarrow f$	1		
Orientadas a BIT						
BCF	f,b,a	Borra un bit de un registro	$0 \rightarrow f < b >$	1		1,2
BSF	f,b,a	Pone un bit de un registro	$1 \rightarrow f < b >$	1		1,2
BTG	f,b,a	Alterna un bit de un registro	$(\overline{f < b >}) \rightarrow f < b >$	1		1,2
Con LITERALES						
MOVLB	k	Carga literal al Nibble bajo en BSR	$k \rightarrow BSR$	1		
MOVLW	k	Carga la	$k \rightarrow W$	1		

		literal al WREG				
LFSR	f,k	Carga la literal (12bit) al registro	$k \rightarrow FSRf$	2		

Tabla (1.5 - 4) Instrucciones de carga.

INSTRUCCIONES ARITMÉTICAS

Mnemónico		Descripción	Comportamiento	CM	Status	Notas
Código	**Operando**					
Instrucciones ARITMÉTICAS						
Orientadas a BYTE						
ADDWF	f,d,a	Suma el contenido de WRG al registro	$(W) + f \rightarrow dest$	1	C,DC,ZOV,N	1,2
ADDWFC	f,d,a	Suma el contenido de WREG, registro y carry	$(W) + (f) + (C) \rightarrow dest$	1	C,DC,ZOV,N	1,2
DECF	f,d,a	Resta 1 al contenido del registro	$(f) - 1 \rightarrow dest$	1	C,DC,ZOV,N	1,2 3,4
INCF	f,d,a	Suma 1 al contenido del registro	$(f) + 1 \rightarrow dest$	1	C,DC,ZOV,N	1,2 3,4
MULWF	f,a	Multiplica los contenidos de (WREG y registro)	$(W)(f) \rightarrow PRODH{:}PRODL$	1		1,2
NEGF	f,a	Contenido del registro negado (C'2)	$(\bar{f}) + 1 \rightarrow f$	1	C,DC,ZOV,N	
SUBFWB	f,d,a	Resta WREG al contenido del registro con acarreo	$(f) - (W) - (\bar{C}) \rightarrow dest$	1	C,CD,ZOV,N	
SUBWF	f,d,a	Resta WREG al contenido del registro	$(f) - (W) \rightarrow dest$	1	C,CD,ZOV,N	1,2
SUBWFB	f,d,a	Resta WREG al contenido del registro con acarreo	$(f) - (W) - (\bar{C}) \rightarrow dest$	1	C,CD,ZOV,N	
Con LITERALES						

ADDLW	k	Suma la literal a WREG	$(W)+k \rightarrow W$	1	C,CD,ZOV,N	
MULLW	k	Multiplica la literal con WREG	$(W)k \rightarrow$PRODH:PRODL	1		
SUBLW	k	Resta WREG a la literal	$k-(W) \rightarrow W$	1	C,CD,ZOV,N	

Tabla (1.5 - 5) Instrucciones aritméticas.

INSTRUCCIONES LÓGICAS

Mnemónico		Descripción	Comportamiento	CM	Status	Notas
Código	**Operando**					
Instrucciones LÓGICAS						
Orientadas a BYTE						
ANDWF	f,d,a	And WREG con el contenido del registro	$(W)AND(f) \rightarrow dest$	1	D,DC,ZOV,N	1,2
COMF	f,d,a	(C'1) al contenido del registro	$(\bar{f}) \rightarrow dest$	1	Z	2
IORWF	f,d,a	OR a la literal con WREG	$(W)\ OR\ k \rightarrow W$	1	Z,N	1,2
RLCF	f,d,a	Rota a la izquierda el contenido del registro con bit Carry	$(f<n>) \rightarrow dest<n+1>$, $(f<7>) \rightarrow C$, $(C) \rightarrow dest<0>$	1	C,ZN	1,2
RLNCF	f,d,a	Rota a la izquierda el contenido del registro sin usar bit Carry	$(f<n>) \rightarrow dest<n+1>$, $(f<7>) \rightarrow dest<0>$	1	Z,N	
RRCF	f,d,a	Rota a la derecha el contenido del registro con bit Carry	$(f<n>) \rightarrow dest<n-1>$, $(f<7>) \rightarrow C$, $(C) \rightarrow dest<0>$	1	C,Z,N	
RRNCF	f,d,a	Rota a la derecha el contenido del registro sin usar bit Carry	$(f<n>) \rightarrow dest<n-1>$, $(f<7>) \rightarrow dest<0>$	1	Z,N	
SWAPF	f,d,a	Intercambia Nibbles del	$(f<3:0>) \rightarrow dest<7:4>$ $(<7:4>) \rightarrow dest<3:0>$	1		4

		contenido del registro				
XORWF	f,d,a	OR Exclusiva WREG con el contenido del registro	$(W)XOR(f) \rightarrow dest$	1	Z,N	
Con LITERALES						
ANDLW	k	AND Literal con WREG	$(W)\ AND\ k \rightarrow W$	1	Z,N	
IORLW	k	OR Literal con WREG	$(W)\ OR\ k \rightarrow W$	1	Z,N	
XORLW	k	OR Literal con WRG	$(W)\ XOR\ k \rightarrow W$	1	Z,N	

Tabla (1.5 - 6) Instrucciones Lógicas.

INSTRUCCIONES DE DECISIÓN CON SALTOS

Mnemónico		Descripción	Comportamiento	CM	Status	Notas
Código	**Operando**					
Instrucciones de SALTOS						
Orientadas a BYTE						
DECFSZ	f,d,a	Resta "1" del contenido del registro y salta si es "0"	$(f)-1 \rightarrow dest$ $salta\ si\ f=0$	1(2o 3)		1,2 3,4
DCFSNZ	f,d,a	Resta "1" al contenido del registro y salta si no es "0"	$(f)-1 \rightarrow dest$ $salta\ si\ f \neq 0$	1(2o 3)		1,2 3,4
INCFSZ	f,d,a	Suma "1" al contenido del registro y salta si es "0"	$(f)+1 \rightarrow dest$ $salta\ si\ f=0$	1(2o 3)		4
INFSNZ	f,d,a	Suma "1" al contenido del registro y salta si no es "0"	$(f)+1 \rightarrow dest$ $salta\ si\ f \neq 0$	1(2o 3)		1,2
Orientado a BITS						
BTFSC	f,b,a	Checa el bit del registro y salta si es "0"	$salta\ si(f<b>)=0$	1(2o 3)		3,4
BTFSS	f,b,a	Checa el bit del registro y salta si es "1"	$salta\ si(f<b>)=1$	1(2o 3)		3,4
Con operaciones LÓGICAS						
CPFSEQ	f,a	Compara el contenido del	$(f)-(W)$, $salta\ si\ (f)=(W)$	1(2o 3)		4

		registro con W y salta si son iguales				
CPFSGT	f,a	Compara el contenido del registro con WREG y salta si f>W	$(f)-(W)$, $salta\ si\ (f)>(W)$	1(2o 3)		4
CPFSLT	f,a	Compara el contenido del registro con WREG y salta si f<W	$(f)-(W)$, $salta\ si\ (f)<(W)$	1(2o 3)		1,2
TSTFSZ	f,a	Checa el contenido del registro y salta si es “0”	$salta\ si\ (f)=0$	1(2o 3)		1,2
Para SUBRUTINAS						
CALL	n,s	Llama a la subrutina	$(PC+4)\rightarrow TOS$, $k\rightarrow PC<20:1>$ $si\ s=1$ $(W)\rightarrow WS$, $(STATUS)\rightarrow STATUSS$ $(BSR)\rightarrow BSRS$	2		
RCALL	n	Llamada relativa	$(PC+2)\rightarrow TOS$, $(PC+2)+2+2n\rightarrow PC$	2		
RETLW	K	Regresa con la literal a WREG	$K\rightarrow W$, $(TOS)\rightarrow PC$, $PCLATU, PCLATH\ estan\ \text{sin}\ cambios$	2		
RETURN	s	Regresa de la subrutina	$(TOS)\rightarrow PC$, $si\ s=1$ $(WS)\rightarrow W$, $(STATUSS)\rightarrow STATUS$, $(BSRS)\rightarrow BSR$, $PCLATU, PCLATH\ estan\ sin\ cambios$	2		

Tabla (1.5 - 7) Instrucciones de decisión con saltos.

INSTRUCCIONES ESPECIALES Y DE SALTO

Mnemónico		Descripción	Comportamiento	CM	Sta
Código	**Operando**				
BC	n	bifurcación si carry	$Si\ bit\ Carry\ es\ 1$ $(PC) + 2 + 2n \rightarrow PC$	1(2)	
BN	n	Bifurcación si es negativo	$Si\ bit\ Negativo\ es\ 1$ $(PC) + 2 + 2n \rightarrow PC$	1(2)	
BNC	n	Bifurcación si no es Negativo	$Si\ bit\ Carry\ es\ 0$ $(PC) + 2 + 2n \rightarrow PC$	1(2)	
BNN	n	Bifurcación si no es negativo	$Si\ bit\ Negativo\ es\ 0$ $(PC) + 2 + 2n \rightarrow PC$	1(2)	
BNOV	n	Bifurcación si no desborda	$Si\ bit\ Overflow\ es\ 0$ $(PC) + 2 + 2n \rightarrow PC$	1(2)	
BNZ	n	Bifurcación si no es Zero	$Si\ bit\ Zero\ es\ 0$ $(PC) + 2 + 2n \rightarrow PC$	1(2)	
BOV	n	Bifurcación si desborda	$Si\ bit\ Overflow\ es\ 1$ $(PC) + 2 + 2n \rightarrow PC$	1(2)	
BRA	n	Bifurcación incondicional	$(PC) + 2 + 2n \rightarrow PC$	2	
BZ	n	Bifurcación si es Zero	$Si\ bit\ Zero\ es\ 1$ $(PC) + 2 + 2n \rightarrow PC$	1(2)	
CLRWDT	----	Borrar Watchdog-Timer	$000h \rightarrow WDT,$ $000h \rightarrow WDT\ postescaler$ $1 \rightarrow \overline{TO},$ $1 \rightarrow \overline{PD}$	1	$\overline{TO},$ $\overline{PD}$
DAW	----	Ajuste Decimal del registro W	$si[W < 3:0 > > 9]o[DC = 1]$ $(W < 3:0 >) + 6 \rightarrow W < 3:0 >;$ $sino$ $(w < 3:0 >) \rightarrow w < 3:0 >$ $si[W < 7:4 > + DC > 9]o[C = 1]$ $(W < 7:4 >) + 6 + DC \rightarrow W < 7:4 >$ $sino$ $(W < 7:4 >) + DC \rightarrow W < 7:4 >$	1	C
NOP	----	No opera	Sin operación	1	
POP	----	Pop inicio retorno de pila(STACK)	$(TOS) \rightarrow cubo\ de\ bits$	1	
PUSH	----	Pon inicio al retorno de pila	$(PC + 2) \rightarrow TOS$	1	
RESET	----	Reinicio por software del dispositivo	$Reinicia\ todos\ los\ registros\ y$ $banderas\ que\ son\ afectados\ por$ $un\ reinicio\ de\ \overline{MCLR}$	1	
RETFIE	s	regreso de la interrupción	$(TOS) \rightarrow PC,$ $1 \rightarrow GIE/GIEH\ o\ PEIE/GIEL,$	2	GIE $GIEH$

	enable	$si\ s = 1$ $(WS) \rightarrow W,$ $(STATUSS) \rightarrow STATUS,$ $(BSRS) \rightarrow BSR$ $PCLATU, PCLATH\ no\ son\ cambiados$		$PEIE$ $GIEL$
SLEEP ----	Duerme	$000h \rightarrow WDT,$ $0 \rightarrow WDT\ postescaler,$ $1 \rightarrow \overline{TO},$ $1 \rightarrow \overline{PD}$	1	$\overline{TO}$ $\overline{PD}$

Tabla (1.5 - 8) Instrucciones especiales y de salto.

LENGUAJE ALTO NIVEL

TIPOS DE DATOS EN C COMPILER

TIPO	TAMAÑO EN BITS	SIN SIGNO	CON SIGNO
int1	número de 1 bit	0 a 1	no
int8	número de 8 bit	0 a 255	-128 a 127
int16	número de 16 bit	0 a 65 635	-32768 a 32767
int32	número de 32 bit	0 a 4294967295	-2147483648 a 2147483647
float32	bit 32 flotante	$-1.5x10^{46} a\ 3.4x10^{36}$	

Tabla (1.5 - 9) Tipos de datos, para los PIC.

SIMILITUD EN LOS TIPOS DE DATOS C ESTÁNDAR CON PIC C

TIPO C ESTANDARD	TIPO C DEFAULT
short	int1
char	unsigned int8
int	int8
long	int16
long long	int32
float	float32

Tabla (1.5 - 10) Tipos de datos.

OPERADORES

OPERADOR	NOMBRE
Aritméticos	
+	suma
-	resta
*	multiplicación
/	división
%	división en módulo (residuo)
--	decremento
++	incremento
lógicos	
>	mayor que
<	menor que
>=	mayor o igual que
<=	menor o igual que
==	igual
!=	distinto de
!	no
||	O
&&	Y
&	dirección

Tabla (1.5 - 11) Tipos de operadores más utilizados.

PRECEDENCIA EN LOS OPERADORES ARITMÉTICOS Y LÓGICOS

OPERADOR	PRECEDENCIA
Aritméticos	
( )	Mayor
++ --	
*** / %**	
+ -	
=	Menor
Lógicos	
!	Mayor
> < >= <=	
== !=	
&$	
||	Menor

Tabla (1.5 - 12) Orden de importancia entre los operadores.

LAS CONSTANTES

EXPRESIÓN	TIPO
123	decimal
0123	octal
0x123	hexadecimal
0b010010	binario
‘x’	carácter
‘/010’	octal
‘/xA5’	hexadecimal
‘/c’	carácter
“abcdef”	cadena de carácter

Tabla (1.5 - 13) Las constantes y expresiones permitidas en PIC C.

CÓDIGOS DE FORMATO

CÓDIGO	FORMATO
%c	Carácter
%d	Enteros decimales con signo
%i	Enteros decimales con signo
%e	Notación científica (e minúscula)
%E	Notación científica (E mayúscula)
%e	Coma flotante
%g	Usar %e ó %f el que resulte más corto
%G	Usar %E ó %f el que resulte más corto
%o	Octal sin signo
%s	Cadena de caracteres (arreglo)
%u	Enteros decimales sin signo
%x	Hexadecimales sin signo (letras minúsculas)
%X	Hexadecimales sin signo (letras mayúsculas)
%p	Mostrar puntero
%n	El argumento asociado es un puntero, a a entero al que no se asigna el número de caracteres escritos
%%	Imprimir el signo

Tabla (1.5 - 14) Códigos de Formato.

CÓDIGO DE BARRAS INVERTIDAS

CÓDIGO	SIGNIFIVADO
\b	Espacio atrás
\f	Salto de página
\n	Salto de línea
\r	Retorno de carro ¶
\t	Tabulador
\”	Comillas
\’	Apostrofo
\0	Nulo
\\	Barra invertida
\r	Tabulador vertical
\a	Alerta (sonido en C estandar)

Tabla (1.5 - 15) Código de barras invertidas.

CARACTERÍSTICAS TÉCNICAS DE LOS DISPOSITIVOS

CARACTERÍSTICAS TÉCNICAS DEL PIC18F4550

CARACTERÍSTICAS ESPECIALES DEL MICROCONTROLADOR

- Optimizada la Arquitectura del Compilador C, con juego de instrucciones Extendida Opcional
- Memoria Flash con retención de hasta 40años
- Misma programación bajo el mando de Software
- Niveles de prioridad para Interrupciones
- Hardware Multiplicador de 8 x 8 de un ciclo
- Temporizador Perro guardián Extendido (WDT)
- Periodo de Programación de 4ms a 131s
- Código de Protección programable
- Único Suministro en programación de circuito Serial vía dos Pines (ICSP)
- In Circuit Debug (ICD) vía dos Pines
- Puerto Opción Dedicada ICD/ICSP (44pin solo dispositivos TQFP)
- Ancho de Rango de Operación de Voltaje (2.0V a 5.5V)

MICROCHIP PIC18F2450/4450 PIN 28/40/44-Alto Rendimiento,

Microcontroladores USB con Tecnología de nano Watt

Características Bus Serie Universal (USB)

- USB Versión de Compilación 2.0
- Velocidad Baja (1.5Mb/s) y Velocidad Total (12Mb/s)
- Control de Soporte, Interrupciones, Isochronous y traslados de volumen
- Soporta más de 32 Endpoints (16 bidireccional)
- 256 Byte RAM de Acceso Dual para USB
- Transceptor USB Integrado con Regulador de Voltaje Integrado
- Interface para Transceptor Externo USB Integrado

MODOS DE MANEJO DE PODER

- Activo: CPU encendido, Periféricos encendidos
- Ocioso: CPU apagado, Periféricos encendidos
- Dormido: CPU apagado, Periféricos apagados
- Modo Ocioso Corriente Baja a 5.8μA Típica
- Modo Dormido Corriente Baja a 0.1μA Típica
- Oscilador Temporizador1: 1.8μA Tipica, 32KHz, 2V
- Temporizador Perro Guardián: 2.1μA Tipica
- Oscilador de dos velocidades Start-up

ARQUITECTURA DE OSCILADOR FLEXIBLE

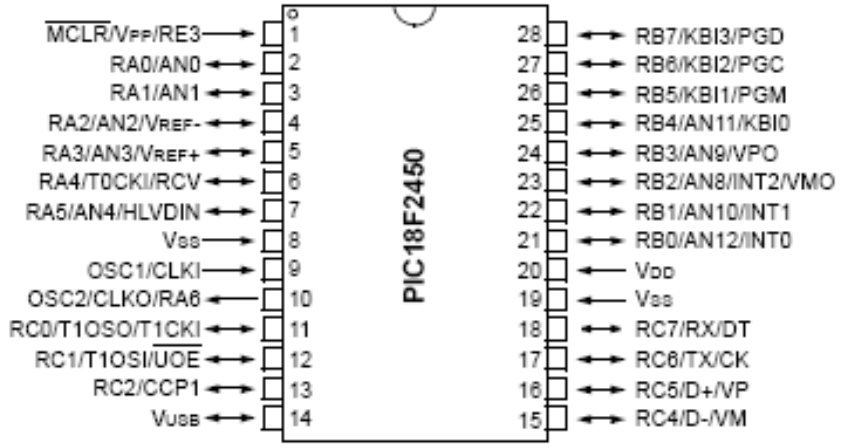
- Cuatro modos de cristal, Incluido Alta precisión PLL para USB
- Dos modos de reloj Externo, arriba de 48MHz
- Bloque de Oscilador Interno
  - Ocho frecuencias seleccionables para 31 KHz a 8MHz
- Oscilador Secundario con uso de Temporizador1 a 32KHz
- El microcontrolador permite Opciones para Oscilador Dual y Modulo USB que opera a diferentes velocidades de reloj
- Seguridad de Fallo Monitoreo de Reloj
  - Permite seguro, disparo bajo si ocurre parada del reloj

PERIFÉRICOS IMPORTANTES

- Corriente Alta Sink/Fuente: 25mA/25mA
- Tres interrupciones Externas
- Cuatro módulos de Temporizador (Modulo0 al Modulo3)
- Captura/Compara/PWM (CCP) Modulo:
  - La captura es de 16bit, máximo. resolución de 5ns
  - La comparación es de 16bit, máximo. resolución de 83.3ns
  - Salida PWM: resolución de PWM de 1 a 10bit
- Enhanced Addressable USART: Este modulo de comunicación serial es capaz de operar bajo la norma RS-232 y soporta el protocolo de bus LIN. Otros apoyos incluyen la detección automático de Proporción del baudaje y una generación de proporción de baudaje a 16bit para mejorar la resolución.
- 10 bit, Arriba de 13 Canales en modulo convertidor Analógico a Digital (A/D):
  - Arriba de 100 ksps de muestreo
  - Tiempo de adquisición programable
- Comparadores Analógicos Duales con entrada multiplexada.

DIAGRAMA DE PINES

28-Pin SPDIP, SOIC



40-Pin PDIP

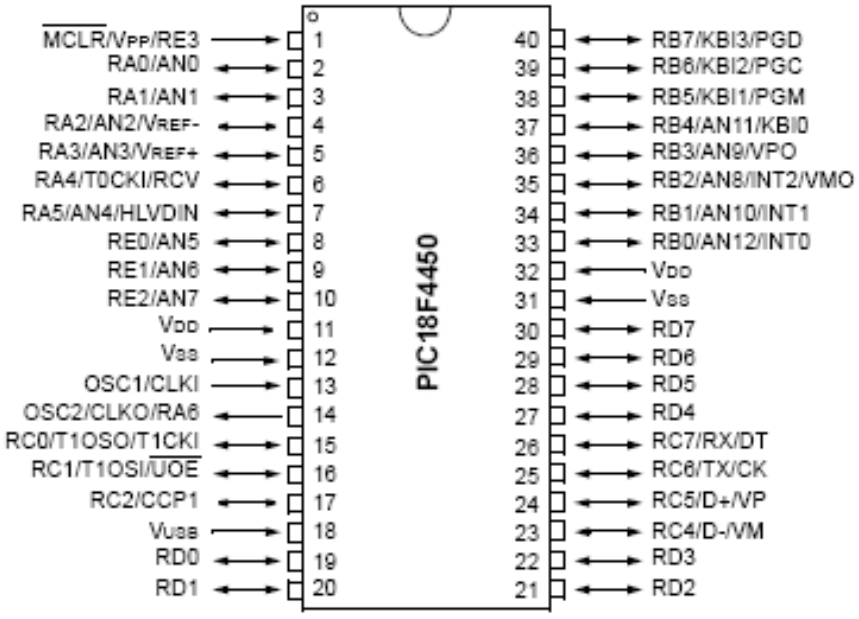


Figura (2.2 -1) Encapsulados PDIP 18F2450 y 18F240.

TABLA COMPARATIVA ENTRE LOS DISPOSITIVOS DE 24PIN Y 40PIN

Características	PIC18F2450	PIC18F4450
Frecuencia de Operación	DC – 48MHz	DC-48MHz
Memoria de Programa (bytes)	16384	16384
Memoria de Programa (Instrucciones)	8192	8192
Memoria de Datos(Bytes)	768	768
Tipos de Interrupción	13	13
Puertos I/O	Puertos A B C (E)	Puertos A B C D E
Temporizadores	3	3
Módulos de Captura/Comparación/PWM	1	1
USART	1	1
Modulo USB	1	1
Modulo 10 bit A/D	10 canales de entrada	13 canales de entrada
RESET (Reinicio) (y Retardos)	POR, BOR, Instrucción RESET, Stack Full, Stack underflow (PWRT, OST), $\overline{MCLR}$ (Opcional), WDT	POR, BOR, Instrucción RESET, Stack Full, Stack underflow (PWRT, OST), $\overline{MCLR}$ (Opcional), WDT
Detección Programable de alto voltaje	si	si
Reinicio Programable Brown-out	si	si
Juego de Instrucciones	75 Instrucciones, 83 Instrucciones si se activa el juego extendido	75 Instrucciones, 83 Instrucciones si se activa el juego extendido
Paquetes	28-pin SPDIP 28-pin SOIC 28-pinQFN	40-pin SPDIP 44-pin SOIC 44-pinQFN

Tabla (2.2 - 1) Comparación de las características entre los PIC18F 4550 y 2450.

ARQUITECTURA INTERNA DEL DISPOSITIVO DE 40PIN

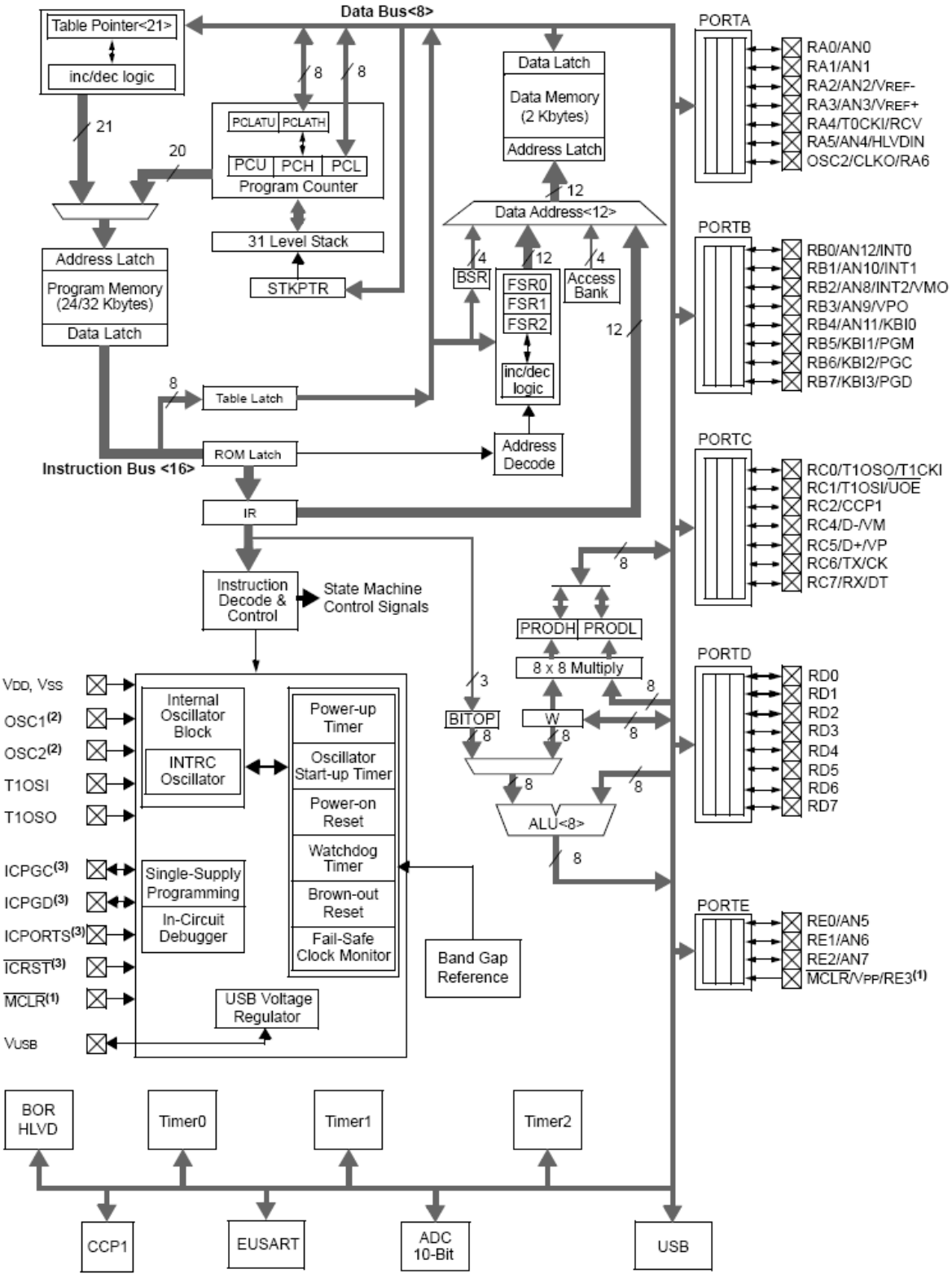


Figura (2.2 – 2) DIAGRAMA A BOQUES PIC18F4450 (40/44-pin).

CARACTERÍSTICAS TÉCNICAS DEL MCP3550/60

Bajo-Consumo, Único-Canal, 22-Bit Delta-Sigma ADC

- ADC 22-Bit en un pequeño paquete MSOP Compensado Automático Interno y Calibración de Ganancia.
- Baja-Producción de Ruido de 2.5V RMS con efectiva resolución de 21.9Bits (MCP3550/1)
- Error típico de 3V
- Error del total de la escala de 2ppm
- Error máximo INL 6ppm
- Error Total sin ajustar de menos de 10ppm
- Marco de Tiempo y Filtro no Digital, Único-Control Conversión a través de 3-Cables Interfaz SPI
- Conversión de Corriente Ultra-Baja (MCP3550/1):
  -100A Típico (VDD=2.7V)
  -120A Típico (VDD=5.0V)
- Entrada Diferencial con Modo de Rango Común VSS a VDD
- Operación con Único-Suministro 2.7V a 5.5V
- Rango Extendido de Temperatura: -40°C a +125°C

DIAGRAMA A BLOQUES

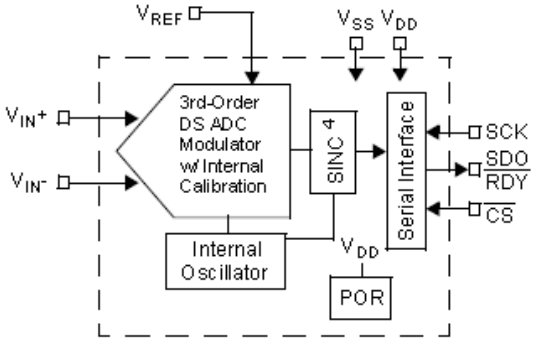
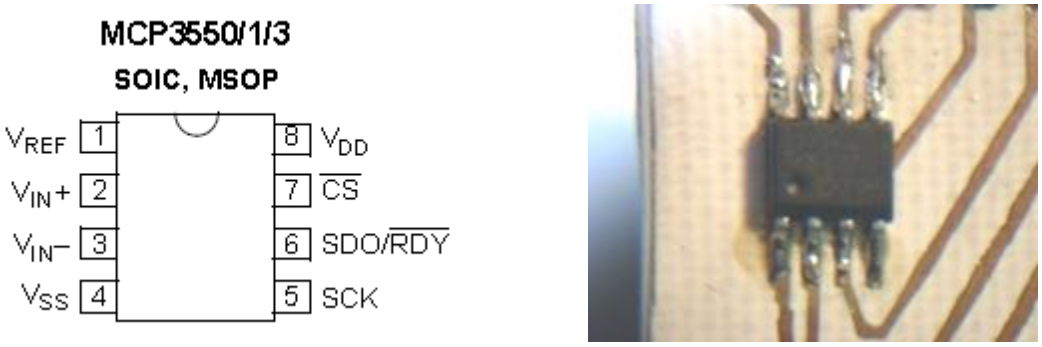


Figura (2.4 – 1) Diagrama a bloques del MCP3550.

TIPOS DE PAQUETES



Figura(2.4 – 2) Ubicación de los pines y Foto real de presentación tipo SOIC.

TABLA COMPARTIVA ENTRE CONVERTIDORES SPI

Número	Sample Rate	Resolución Efectiva	50/60Hz Ejecución
MCP3550-50	12.5 sps	21.9 bits	50 Hz
MCP3550-60	15 sps	21.9 bits	60 Hz
MCP3551	13.75 sps	21.9 bits	50/60 Hz (simultaneo)
MCP3553	60 sps	20.6 bits	N/A

Tabla (2.4.1) de selección del Dispositivo.

DESCRIPCIÓN DE PINES

Pin No.	Símbolo	I/O/P	Función
1	V_{REF}	I	Pin de Entrada Analógica de Referencia de Voltaje
2	V_{IN}+	I	Pin de Entrada Analógica No-Invertida
3	V_{IN}-	I	Pin de Entrada Analógica Invertida
4	V_{SS}	P	Pin de Tierra
5	SCK	I	Pin de Entrada Digital de Reloj Serial
6	SDO/RDY	O	Pin de Salida Digital Ready/Data (Preparar/Dato)
7	CS	I	Pin de Entrada Digital de Selección del Chip
8	V_{DD}	P	Pin de Alimentación de Voltaje Positivo

Identificación de Símbolo: I=Entrada, O=Salida, P=Alimentación

Tabla (2.4 - 2) Función de los Pines del MCP3550.

CARACTERÍSTICAS TÉCNICAS DEL MC14051BCP MULTIPLEXOR ANÁLOGO

ASIGNACIÓN DE PINES

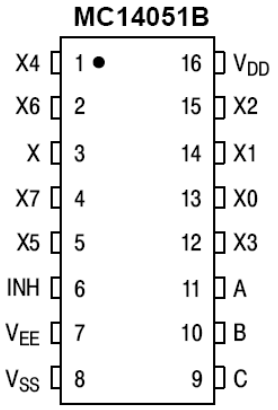


Figura (3.1 - 2) Asignación de pines del MC14051.

TABLA DE VERDAD

Control Inputs				ON Switches					
	Select								
Inhibit	C*	B	A	MC14051B	MC14052B		MC14053B		
0	0	0	0	X0	Y0	X0	Z0	Y0	X0
0	0	0	1	X1	Y1	X1	Z0	Y0	X1
0	0	1	0	X2	Y2	X2	Z0	Y1	X0
0	0	1	1	X3	Y3	X3	Z0	Y1	X1
0	1	0	0	X4			Z1	Y0	X0
0	1	0	1	X5			Z1	Y0	X1
0	1	1	0	X6			Z1	Y1	X0
0	1	1	1	X7			Z1	Y1	X1
1	x	x	x	None	None		None		

*Not applicable for MC14052
x = Don't Care

Figura (3.1 - 3) Tabla de verdad.

EJEMPLO DE APLICACIÓN

Los niveles lógicos son determinados por VDD y VSS. El voltaje VDD es el nivel lógico alto y el VSS es el nivel lógico bajo. Para el ejemplo VDD = 5V = “1” (nivel lógico alto), mientras que VSS = 0V = “0” (nivel lógico bajo) de las entradas de control.

El nivel máximo permitido para la señal analógica es determinado por los voltajes de VDD y VEE. El voltaje VDD determina el pico máximo recomendado por encima de VSS. El voltaje VEE determina la máxima oscilación debajo de VSS. Para el ejemplo, VDD – VSS = 5V oscilación máxima por encima de VSS; VSS – VEE = 5V, oscilación máxima por debajo de VSS. El ejemplo muestra una señal de $\pm$ 4.5V que permite un margen de ½ V en cada pico. Si un voltaje transita por encima de VDD y/o por debajo de VEE se recomienda, en los canales analógicos, diodos externos (Dx) como se muestra en la figura (3.1 – 5). Estos diodos deben ser de tipos de señal pequeña capaz de absorber los aumentos repentinos de corriente máxima durante la saturación.

La diferencia de potencial máxima absoluta entre VDD y VEE son 18V. La mayoría de parámetros se especifican a 15V que es la diferencia máxima recomendada entre VDD y VEE. No son necesarias fuentes simétricas. Sin embargo VSS debe ser mayor o igual a VEE. Por ejemplo VDD = 10V, VSS = 5V, y VEE = -3V es aceptable. Véase la tabla (3.1 – 6), de posibles conexiones (3.1 – 6).

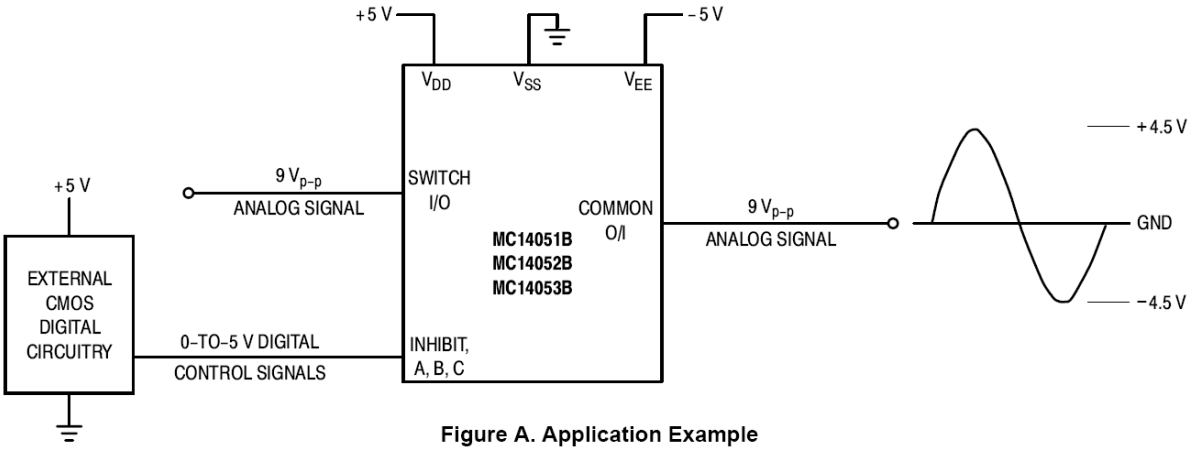


Figura (3.1 - 4) Diagrama a bloques, Ejemplo de aplicación conectado con fuentes simétricas de 5V.

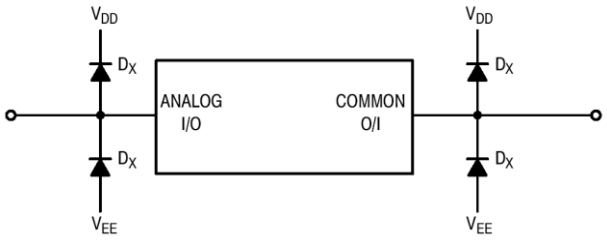


Figura (3.1 – 5) Diodos externos de germanio o Schottky de corte

La siguiente tabla (3.1 -5) muestra ejemplos de posibles conexiones de fuentes que se pueden tener con el MC14051B.

V_{DD} In Volts	V_{SS} In Volts	V_{EE} In Volts	Control Inputs Logic High/Logic Low In Volts	Maximum Analog Signal Range In Volts
+ 8	0	– 8	+ 8/0	+ 8 to – 8 = 16 V_{p-p}
+ 5	0	– 12	+ 5/0	+ 5 to – 12 = 17 V_{p-p}
+ 5	0	0	+ 5/0	+ 5 to 0 = 5 V_{p-p}
+ 5	0	– 5	+ 5/0	+ 5 to – 5 = 10 V_{p-p}
+ 10	+ 5	– 5	+ 10/ + 5	+ 10 to – 5 = 15 V_{p-p}

Tabla (3.1 - 6) Tabla de posibles conexiones de fuentes para el MC14051B.

DIRECCIONES DE INTERNET

FABRICANTES Y DISTRIBUIDORES

- www.microchip.com *Microchip Tecnology Incorporated.* Fabricante de microcontroladores PIC, convertidores A/D MCP
- www.agelectronica.com Distribuidora de componentes electrónicas en México
- www.displaytech-us.com Fabricante de Display LCD

PAGINA SOBRE MICROCONTROLADORES

- www.garcia-cuervo.net/picmania Incluye ejemplos con códigos e imágenes.
- www.muchotrasto.com Buen referente para el estudio del USB 2.0 de microchip

FOROS EN ESPAÑOL SOBRE MICROCONTROLADORES

- www.forosdeelectronica.com
- www.todopic.com.ar

CURSO C ESTANDARD

- www.elrincondelc.com/cursoc Curso de C ideal para principiantes.

Acondicionador.- Es la etapa de un sistema que se encarga adaptar los niveles a los del resto del circuito. También se encarga de linealizar el sensor y compensar sus variaciones. La compensación puede ser hardware o software, el caso del software ya no es parte del acondicionador.

Adquisición de Datos.- También se le llama adquisición de señales, ya que consiste en la toma de muestras analógicas que son convertidas en señales digitales para que puedan ser procesadas por un sistema digital como la PC. El elemento que hace dicha transformación es el módulo de digitalización o tarjeta de adquisición de datos (DAQ).

Conductividad.- Es la propiedad que tiene un material o sustancia para conducir la corriente eléctrica y es lo contrario de la resistencia. La unidad de medición utilizada es el Siemens/cm [µS/cm]

Digitalización.- Consiste en traducir una señal analógica en una digital.

Instrumentación Electrónica.- Es la parte de la electrónica, principalmente analógica, que se encarga del diseño y manejo de los aparatos electrónicos y eléctricos, sobre todo para su uso en mediciones. La instrumentación electrónica se aplica en el sensado y procesamiento de la información proveniente de variables físicas y químicas, a partir de las cuales realiza el monitoreo y control de procesos, empleando dispositivos y tecnologías electrónicas.

Instrumentación Virtual.- La idea es sustituir y ampliar elementos hardware por otros software, para ello se emplea un procesador (normalmente un PC) que ejecute un programa específico, este programa se comunica con los dispositivos para configurarlos y leer sus mediciones.

Algunos programas especializados en este campo son LabView y Agilent – VEE (antes HP – VEE). Algunos buses de comunicación populares son GPIB, RS – 232, USB, etc.

ppm.- Partes por millón. Es la unidad de conductividad en grados americanos por definición equivale a 2µS/cm. También 1.4µS/cm = 1ppm.

Bibliografía

FUNDAMENTOS DE PROGRAMACION C/C++
Autor: Ernesto Peñaloza Romero
4ta Edición
Editorial: Alfaomega

DISEÑO DE CIRCUITOS Y SISTEMAS INTEGRADOS
Autores: Antonio Rubio; Josep Altet;Xavier Aragonés; José Luis González Diego Mateo; Francesc Moll
1ra Edición
Editorial: Alfaomega
Pag 89, 92

MICROCONTROLADOR PIC16F84 Desarrollo de Proyectos
Autores: Enrique Palacios; Fernando Remiro; Lucas J. López
Primera Edición
Editorial: Alfaomega

DIGITAL CIRCUITS AND MICROCOMPUTERS
Authors: David E. Johnson; John L. Hilburn; Paul M. Julich
Editorial: Prentice Hall

DIGITAL CIRCUITS – A preparation for Microprocessor
Authors: Choules W. McKay
Editorial: Prentice Hall

DIGITAL FUNDAMENTALS
Authors: Floyd
Fifth Edition
Editorial: Prentice Hall.

INTRODUCCIÓN A LOS SISTEMAS DE COMUNICACIÓN
Autor: F.G. STREMLER
Tercera Edición
Editorial: Pearson