UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IDENTIFICACIÓN DEL MODELO DE UN MOTOR DC UTILIZANDO EL TOOL BOX DE IDENTIFICACIÓN DE MATLAB Y UN MICROCONTROLADOR PIC

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

MODESTO ALARCÓN MORE

PROMOCIÓN
1990- I

LIMA – PERU
2011

IDENTIFICACIÓN DEL MODELO DE UN MOTOR DC UTILIZANDO EL TOOL BOX DE IDENTIFICACION DE MATLAB Y UN MICROCONTROLADOR PIC

DEDICATORIA

A mis padres,

Luz y Hortencio,

por el apoyo incondicional

que me brindaron en vida.

SUMARIO

El informe consiste en una transferencia y procesamiento de datos a alta velocidad entre el PIC18F2550 y la PC usando MATLAB vía puerto USB2.0.

El microcontrolador, un motor DC con encoder acoplado en su eje están implementados en una tarjeta de adquisición de datos, todo el circuito esta unido fisicamente mediante el cable USB y se conecta a la computadora. El microcontrolador recibe una cantidad de pulsos (Encoder) en un periodo de muestreo, luego; esta información es enviada a la PC estableciéndose de esta manera la comunicación en un sentido. Un programa en MATLAB se encarga de recibir, procesar y enviar la información hacia el PIC de esta manera se logra una comunicación de ida y vuelta entre PC y microcontrolador.

El intercambio de información se pueden efectuar de hasta 64bytes por paquete cada milisegundo es decir se puede transferir datos a alta velocidad del PIC18F2550 a la PC de manera bidireccional con la finalidad de procesarlos y/o graficarlos.

El trabajo consiste en la obtención de curvas de arranque del motor DC sin carga a efectos de lograr un modelo más adecuado de la planta (Motor DC), y luego dar paso al diseño de un controlador más acertado. Se requiere importar datos de eventos muy rápidos, como por ejemplo el arranque de un motor DC (velocidad con respecto al tiempo) que tiene una duración aproximada de una decima de segundo.

También se controla la velocidad del motor DC por modulación de ancho de pulso (PWM), pues ante la señal PWM el voltaje promedio es proporcional a la velocidad del motor DC. MATLAB ha evolucionado al grado que se puede conectar casi con cualquier dispositivo, y su manejo es relativamente fácil porque incluye una ayuda en línea.

La comunicación entre el microcontrolador y la PC por software para el tratamiento de datos es la manera más económica y sin algún software ejecutable que sirva de intermediario. MICROCHIP proporciona en su página web los drivers y archivos necesarios para establecer la comunicación por puerto USB con la familia del PIC18. [1]

INDICE

INTRODUCCIÓN 1

1.1. Introducción 1

CAPITULO I 3

PLANTEAMIENTO DEL PROBLEMA 3

1.1. Descripción del problema 3

1.1.1 Programación del Microcontrolador PIC18F2550 3

1.1.2.Transmisión y recepción de datos a la PC desde el Microcontrolador 5

1.2. Objetivos del problema 6

1.2.1 Tiempo de muestreo 6

1.2.2 Generación de PWM1 y PWM2 7

CAPITULO II 11

DESCRIPCIÓN DE LOS ELEMENTOS DE HARDWARE Y SOFTWARRE 11

2.1. Antecedentes del problema 11

2.1.1.Arquitectura del Puerto USB 11

2.2. Bases Teóricas 14

2.2.1. La MPUSBAPI.DLL de Microchip 14

2.2.2. Tipos de transferencias soportados por estas instrucciones 18

2.2.3.Declaración de Constantes y Variables 19

2.3. Definición de términos 20

CAPITULO III 27

METODOLOGÍA PARA LA SOLUCIÓN DEL PROBLEMA 27

3.1. Alternativas de Solución 27

3.1.1 Enlace a la PC mediante USB 27

3.1.2.Configurando el Hardware 27

3.1.3.Configurando el Software 28

3.2. Enlace de MATLAB al PIC para transferencia de datos.....30
3.3. Pruebas finales y correciones 33
3.4. Software.....33
3.5. Procedimientos para la realización del Software.....34
3.6. Simulación en PROTEUS.....40
3.7. Comunicación fluida MATLAB-PROTEUS.....40
CAPITULO IV 44
ANÁLISIS Y PRESENTACIÓN DE RESULTADOS.....44
4.1. Análisis Descriptivo 44
4.1.1. Parámetros del Motor DC.....44
4.2 Adquisición de Datos 45
CONCLUSIONES Y RECOMENDACIONES 49
ANEXO A.....50
Análisis Teórico 50
ANEXO B.....79
Resultados del Tool Box de Identificación de MATLAB.....79
BIBLIOGRAFÍA.....89

INTRODUCCION

Hoy en día resulta muy interesante observar como los avances tecnológicos nos sorprenden por la evolución tan rápida que presentan, y que son fáciles de usar, es decir se están volviendo muy amigables al punto de comprender fácilmente su funcionamiento al usarlos e instalarlos, tal es el caso del Universal Serial Bus más conocido como USB.

La miniaturización de los componentes electrónicos es cada vez más sorprendente, la picoelectrónica trae consigo aun más la reducción del tamaño de los semiconductores y disminución de su consumo de corriente. El microcontrolador no es la excepción pues se ha desarrollado notablemente, al punto de tener más grande sus capacidades de comunicación, memoria, circuitos de apoyo adicionales (ADC, oscilador interno, más puertos).

Estos microcontroladores requieren de muy pocos componentes externos para su implementación. Microchip, se preocupa en desarrollar las herramientas adecuadas para su programación, tales como lenguajes de alto nivel para compilar el programa y programadores económicos para descargar el programa en el chip.

Actualmente ya no basta tener un chip inteligente que funcione de manera autónoma, ahora se requiere que trabajen en conjunto con la PC especialmente donde se requieren importar datos de eventos muy rápidos.

El propósito del trabajo consiste en lograr una comunicación entre el microcontrolador y la PC utilizando el cable USB de 4 hilos para el enlace físico, para ello necesitamos desarrollar tres puntos:

- Enlace PC –Microcontrolador mediante conector USB hasta observar en el administrador de dispositivos de la PC que un nuevo hardware se ha instalado.
- Enlace de MATLAB a PIC hasta obtener un fluido intercambio de información entre PC y el microcontrolador.
- Pruebas finales y correcciones.

Para lograr el enlace mediante USB se utilizan las funciones USB incorporadas en el

lenguaje C del programa CCS (Custom Computer Service) para PICs , dichas funciones están preparadas para que el microcontrolador sea reconocido como un dispositivo personalizado y usando los descriptores que incluye el mismo lenguaje, se establece la cantidad de datos a 64bytes. Para el enlace PC – Microcontrolador se debe realizar el diseño e implementación de una tarjeta de adquisición de datos. La tarjeta de adquisición está conformada por el Microcontrolador (PIC18F2550), éste PIC pertenece a la gama alta y la principal razón para su elección es su configuración para comunicación vía USB con la PC, que nos permite una transferencia de información es decir recibir y enviar datos desde la PC al PIC y del PIC a la PC.

El motor DC trabaja a tensión máxima de 30 Voltios con un Encoder acoplado en su eje de 400 ranuras, para esta aplicación es importante tener un Encoder con un número de ranuras aproximadamente de 360 ranuras por vuelta, pues se desea obtener un modelo preciso de la planta (Motor DC). De esta manera, se puede diseñar un controlador más adecuado que nos permitirá controlar la Velocidad y/o Posición del Motor DC válido para tareas de posicionamiento.

El objetivo del informe es obtener el conocimiento y las actitudes suficientes para lograr la utilización del bus USB, como interfaz con la PC para aplicaciones de uso general, tecnología que actualmente es muy utilizada en el mundo .Las limitaciones del trabajo es que la transferencia de datos, se realiza a corta distancia hasta un máximo de 5m entre la PC y la tarjeta de adquisición de datos.

En el Capítulo I, se realiza una presentación general del problema y en un esquema se sintetiza como se debe trabajar para lograr los objetivos. En el Capítulo II, se analiza la arquitectura USB para lograr un entendimiento de cómo fluyen los datos en una forma bidireccional entre Host y periféricos. Se necesita definir algunos términos que sirven para el mejor entendimiento sobre todo para el software.

En el Capítulo III, se propone un diagrama circuital de adquisición de datos para lograr el enlace PC mediante USB, también se analiza algunos comandos en MATLAB para lograr un enlace MATLAB con el PIC. Se presenta un programa en C que posteriormente es grabado en el PIC, el programa en MATLAB y el diagrama para la implementación. En el Capítulo IV se muestran algunos resultados obtenidos mediante la simulación.

CAPITULO I
PLANTEAMIENTO DE INGENIERIA DEL PROBLEMA

1.1 Descripción del Problema.

La finalidad es lograr una transferencia fluida de datos de PC a PIC y viceversa para ello se debe trabajar a nivel de software y hardware en una primera etapa se debe cumplir los siguientes objetivos:

- Diseño e implementación de la tarjeta de adquisición de datos.
- Lectura de los pulsos del Encoder acoplado al eje del motor DC cada 0.1seg.
- Generación de dos señales PWM (Modulación de ancho de pulso).

La tarjeta de adquisición de datos está conformada por:

- PIC18F2550 (Gama Alta).
- Condensadores, resistencias.
- Cristal de 20MHZ.
- Conector –Cable USB.
- Motor DC con Encoder acoplado en el eje de 400 ranuras.

El Hardware es bastante reducido y para su implementación es relativamente sencillo, en cambio el software es bastante complicado porque el enlace PC – PIC requiere de protocolos de software avanzados. El puerto USB nos ofrece más ventajas que sus predecesores pero, su complejidad para implementarlo es enorme ya que su funcionamiento está basado en protocolos de software.

Se presenta un esquema general formado por un diagrama de bloques donde se visualiza la conexión PC – Tarjeta de Adquisición mediante el cable USB del presente trabajo Figura 1.1.

1.1.1Programación del Microcontrolador PIC18F2550

El software que se utiliza es el MPLABv8.5, el lenguaje de programación es el C éste código nos permitirá lo siguiente.

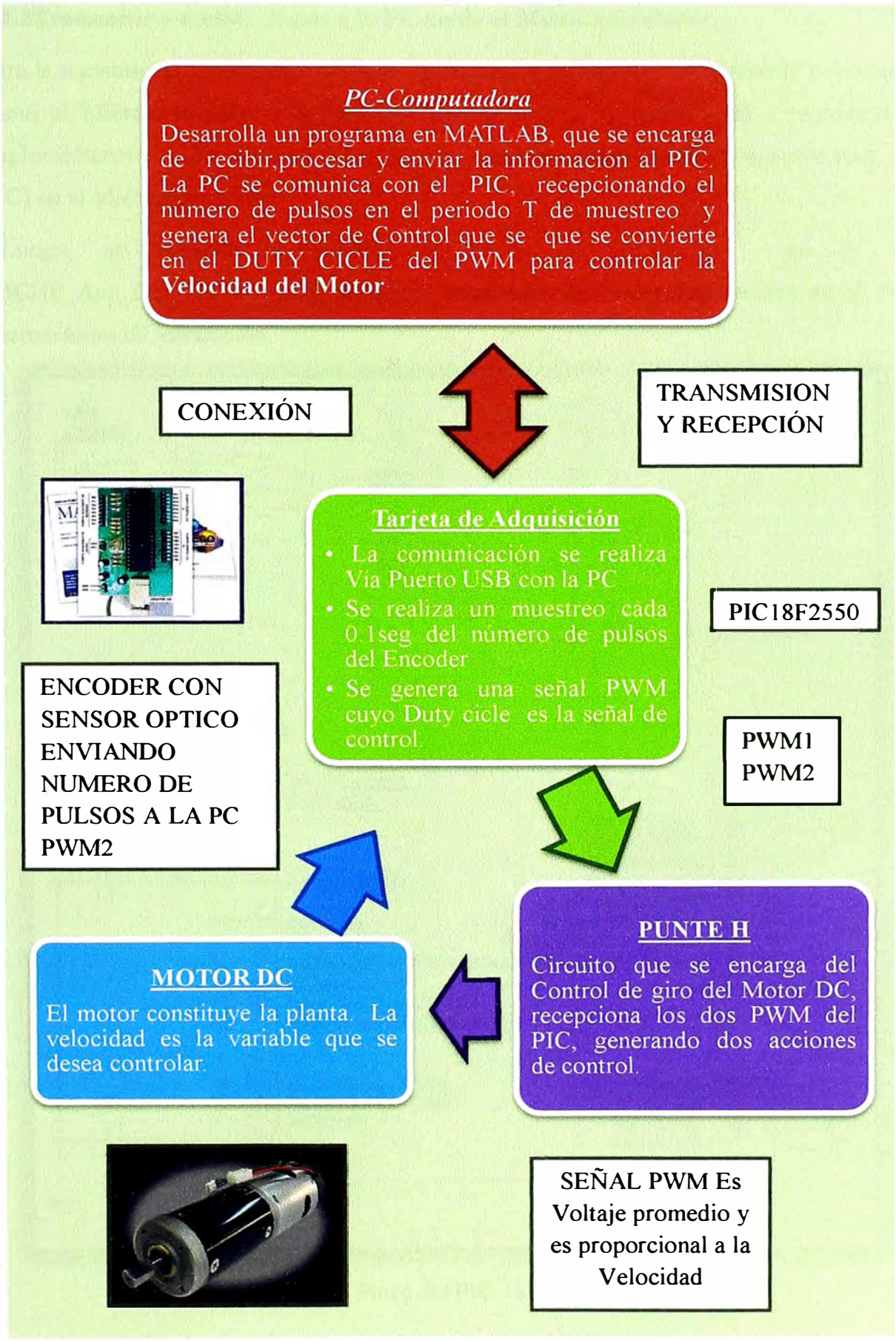


Fig. 1.1 Esquema General

1.1.2Transmitir y recibir Datos a la PC desde el Microcontrolador

Para la transmisión y recepción de datos es necesario utilizar los componentes principales como el Microcontrolador PIC18F2550 (figura1.2) y el conector USB y terminada la implementación del Hardware se compila y se carga el código DAQ.c (Programa en C del PIC) en el Microcontrolador.

Luego, se procede a instalar el driver de Microchip en la PC (MCHP_App_Lib_v2010_10_19_Installer), siguiendo los pasos establecidos en el PDF instrucciones de instalación.

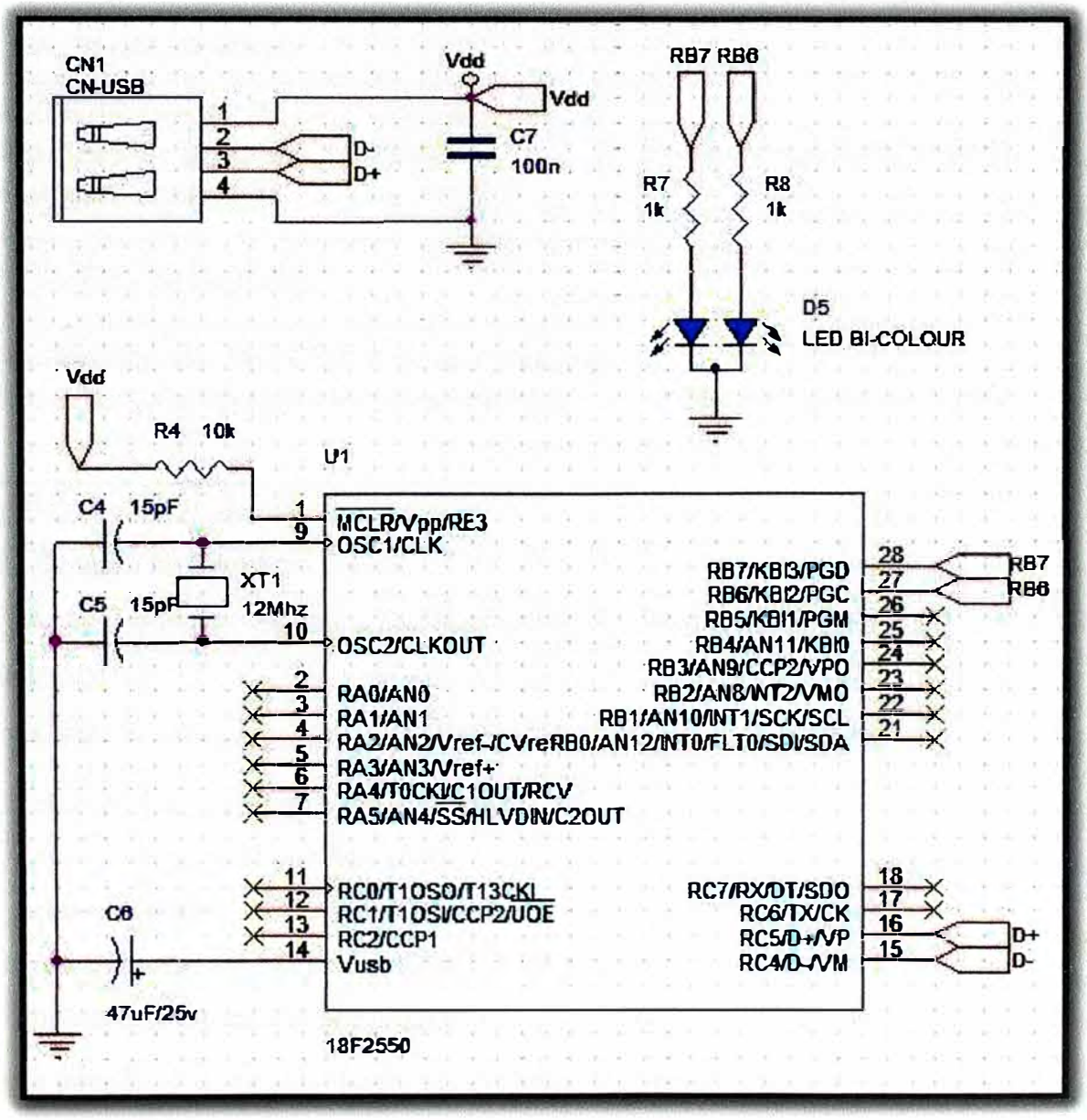


Fig.1.2 Pines del PIC 18F2550 [5]

Después se procede a la instalación del programa MATLAB, en el se establecen los siguientes parámetros de lectura y escritura.

- **Pipe de Lectura:**

```
[my_out_pipe]=calllib('libreria','MPUSBOpe
n',uint8(0),vid_pid_norm,out_pipe,uint8(0),
uint8(0));
```

- **Pipe de Escritura**

```
[my_in_pipe]=calllib('libreria','MPUSBOpen'
,uint8(0),vid_pid_norm,in_pipe,uint8(1),
uint8(0));
```

- **Lectura de los Datos:**

```
[aa,bb,data_in,dd]=calllib('libreria','MPUSB
Read',my_in_pipe,data_in,uint8(64),
uint8(64), uint8(10));
```

data_in=eye(1,64,'uint8')

- **Escritura de los Datos:**

```
[calllib('libreria','MPUSBWrite',my_out_pip
e,data_out,uint8(64), uint8(64), uint8(10));
```

data_out=eye(1,64,'uint8')

1.2 Objetivos del problema.

1.2.1 Tiempo de muestreo 0.1seg

El TIMER2 se puede emplear como base de tiempos para la modulación en ancho de pulso (PWM).

El TIMER2 es un módulo temporizador con las siguientes características:

- Temporizador de 8 bits (registro TMR2).
- Registro de periodo de 8 bits (PR2).
- Ambos registros pueden leer o escribir.
- Prescaler programable por programa (1:1,1:4,1:16).
- Postscaler programable por programa (1:1 a 1:16).

Para generar el tiempo de muestreo con bastante precisión como se requiere se utiliza el temporizador TMR2 con el tiempo de desbordamiento que se calcula según la siguiente ecuación del TMR2:

$T=T_{CM}$.[Prescaler .(Carga TMR2 + 1).Postscaler] .....(1.1)

donde:

T_{CM}=Es el tiempo de ciclo de máquina y se puede calcular mediante la ecuación:

$T_{CM}=4/F_{osc}$

$$T_{CM} = \frac{4}{48 .10^6} = \frac{10^{-6}}{12} \text{ seg} \quad \text{.....(1.2)}$$

Reemplazando en fórmula (1.1) se obtiene:

- Carga TMR2 =249
- Prescaler =16
- Postscaler =15

$T = \frac{10^{-6}}{12}$ [16. (249 + 1).15]

T= 5ms

Para una frecuencia del oscilador Fosc= 48Mhz, debemos que obtener un T=0.1seg, para ello se propone la siguiente solución como el PR2=Carga del TMR2 es un numero de 8bits, es decir de 0-255, le asignamos el valor de 249, luego a nuestro PRESCALER le asignamos un tiempo de división de 16, y al postscaler un valor de 15.

Haciendo estas consideraciones obtenemos 5 mseg exactos, pero deseamos obtener 100 mseg.- Entonces, creamos una variable cont_muestreo y además utilizando un bucle if hacemos que se repita esto 20 veces, de esta forma obtenemos los 0.1 seg. Habilitando en este instante en la interrupción una bandera que nos indicara que debemos obtener el número de pulsos en ese instante.

```
#INT_TIMER2
      Void interrupción_timer2(void)
      {
            cont_muestreo++;
            if (cont_muestreo= =20)
            {
                  bandera_envia=1;
                  cont_muestreo=0;
            }
      }
```

```
int cont  muestreo=0;
```

En la función principal tenemos:

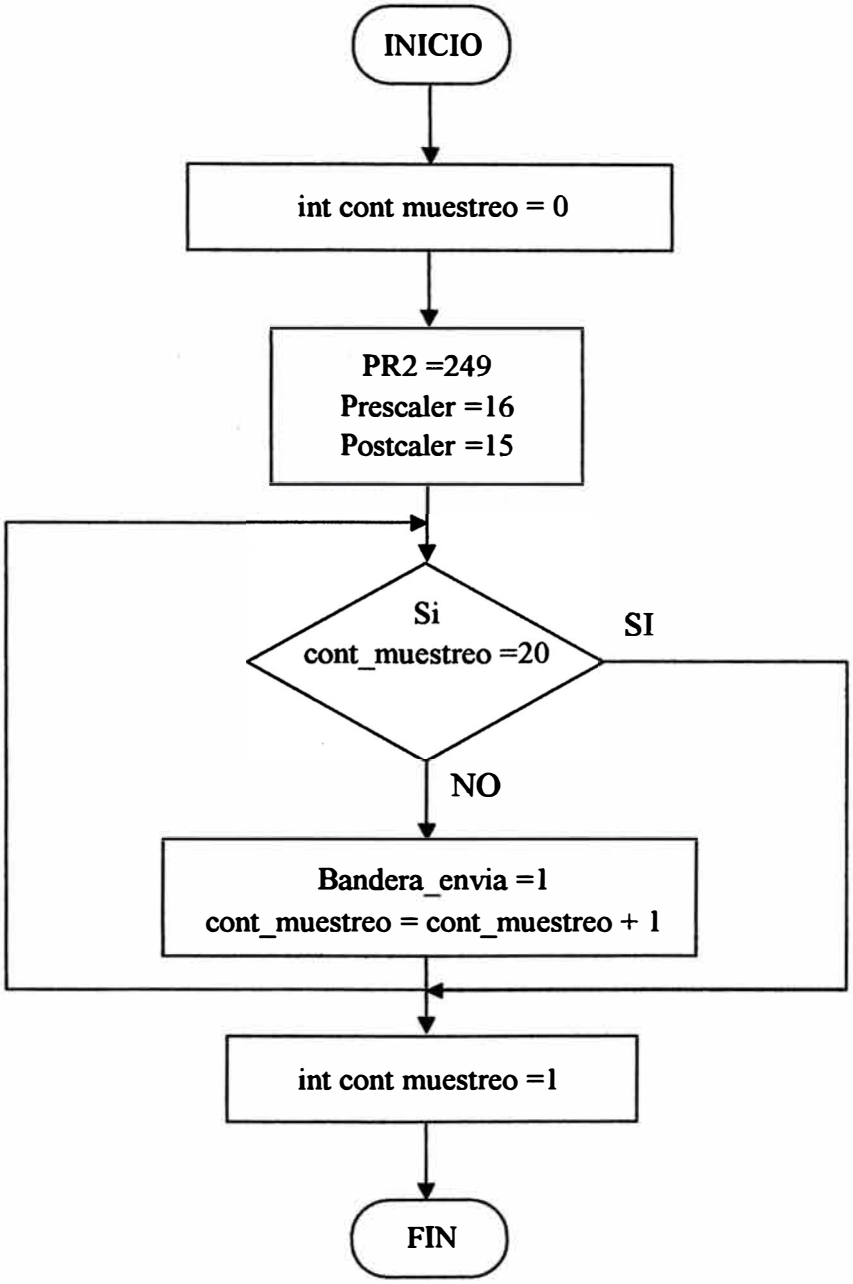
```
Void main (void) {
                  setup _timer_2(T2_DIV_BY_16,249,15) ;
```

1.2.2 Generación de PWM1 Y PWM2

Para la generación de los PWM se usan los módulos CCP, la ventaja de trabajar con el PIC18F2550, es que contamos con dos módulos números suficientes para llevar a cabo esta

tarea, como control de velocidad de un motor DC.

DIAGRAMA DE FLUJO DE INT- TIMER2



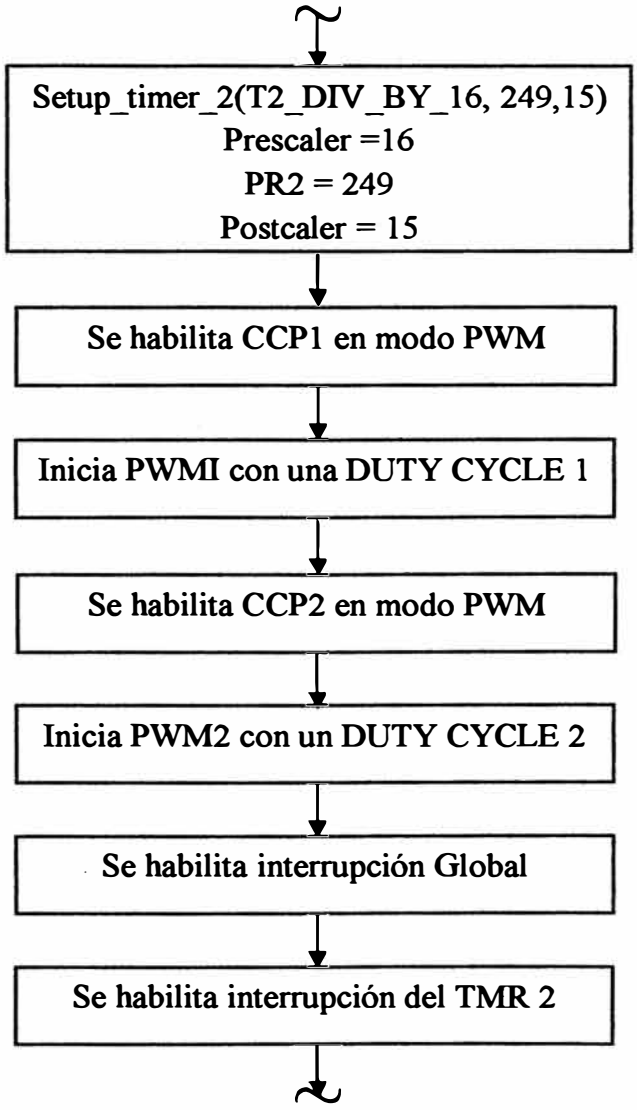
La configuración la hacemos en la función principal, pero antes debemos definir las siguientes variables:

```
int ancho_pulso1=145;
int ancho_pulso2=120;
```

Estos valores serán generados por la PC después de haber sido procesados siguiendo el Algoritmo de Control, el valor que indica la variable ancho de pulso es igual al Duty Cicle, como tenemos dos PWM definimos los valores PWM1 y PWM2 respectivamente.

```
void main (void)     {
/ / esto también es nuevo
                     setup_timer_2(T2_DIV_BY_16,249,15);
                     setup_ccpl(CCP_PWM) ;
                     SET_PWM1_DUTY(ancho_pulso1) ;
                     setup_ccp2{CCP_PWM)
                     SET_PWM1_DUTY(ancho_pulso2) ;
                     enable_interrupts(GLOBAL) ;
                     enable_interrupts(INT_TIMER2) ;
```

DIAGRAMA DE FLUJO PARA PWM1 Y PWM2



La primera sentencia se usa para configurar el TMR2 como temporizador, y a la vez nos sirve para indicar lo siguiente:

setup_timer_2 (modo,periodo,postscaler) donde:

- Periodo es un valor entero de 8 bits (0-255) para el registro PR2;
- Postscaler es el valor del postscaler (1 a 16). Afecta a los bits 6.3 del registro T2CON;
- Modo afecta a los bits 2:0 del registro T2CON

El periodo de la señal de PWM, que se ha configurado está determinado por la ecuación:

PWMT= (PR2+1).4.T_{OSC}.(Valor del Preescaler del TMR2).....(1.3)

Reemplazando valores obtenemos:

$$PWM = (249 + 1).\frac{4.\ 10^{-6}}{48}.(16) = 1/3 \text{ mseg.}$$

El *registro PR2=249*.

CAPITULO II
DESCRIPCIÓN DE LOS ELEMENTOS DE HARDWARE Y SOFTWARE

2.1 Antecedentes del Problema.

2.1.1Arquitectura del Puerto USB

El puerto USB nos ofrece más ventajas que sus predecesores, pero su complejidad para implementarlo es enorme ya que su funcionamiento está basado en protocolos de software.

Solo puede haber un Host en el bus que maneja a todos los componentes conectados. La topología física del USB es de tipo estrella jerarquizada con un máximo de 7 niveles de jerarquía y permite el funcionamiento simultáneo de 127 dispositivos a la vez como se observa en la Figura 2.1

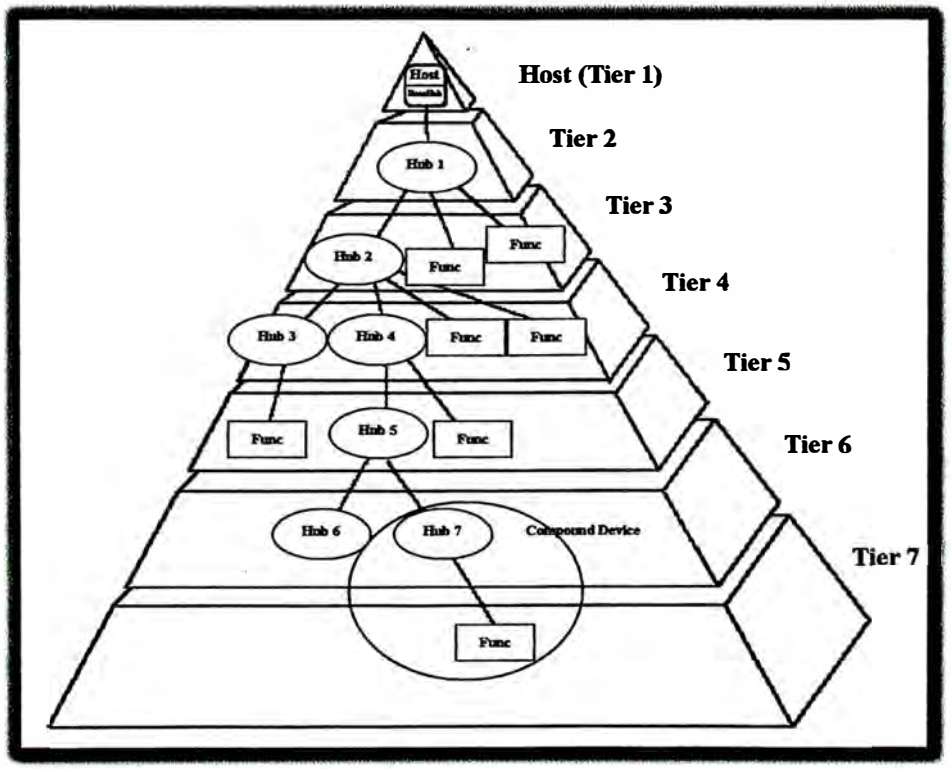


Fig.2.1 Topología del bus [1]

"El HUB es un elemento plug and play en la estructura USB y es un concentrador al cual se le puede agregar más dispositivos USB (Figura 2.2), incluyendo otro HUB .La velocidad de transferencia depende del HUB que se esté utilizando" [1].

De la información [1] se tiene las velocidades típicas (Figura 2.3) son:

- Low- Speed: 1.5Mbps.
- Full-Speed: 12Mbps.
- High Speed: 480Mbps.

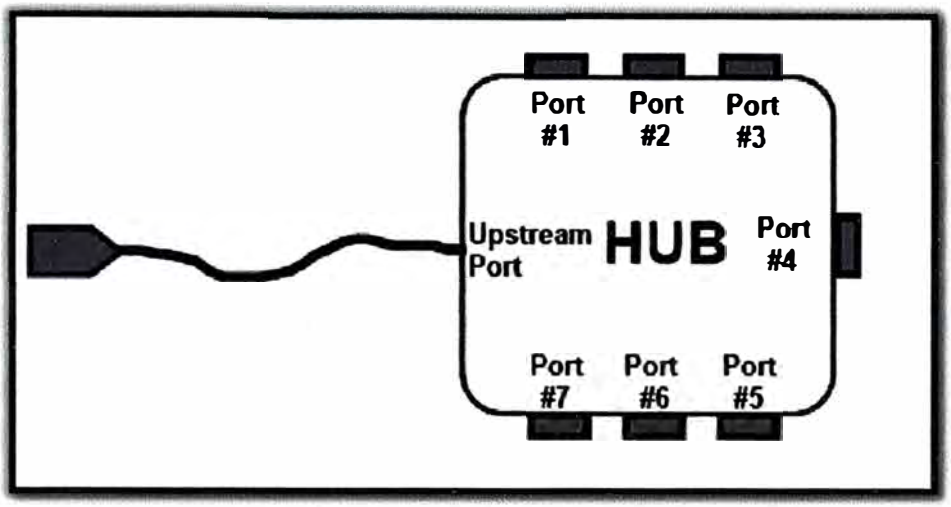


Fig.2.2 HUB USB

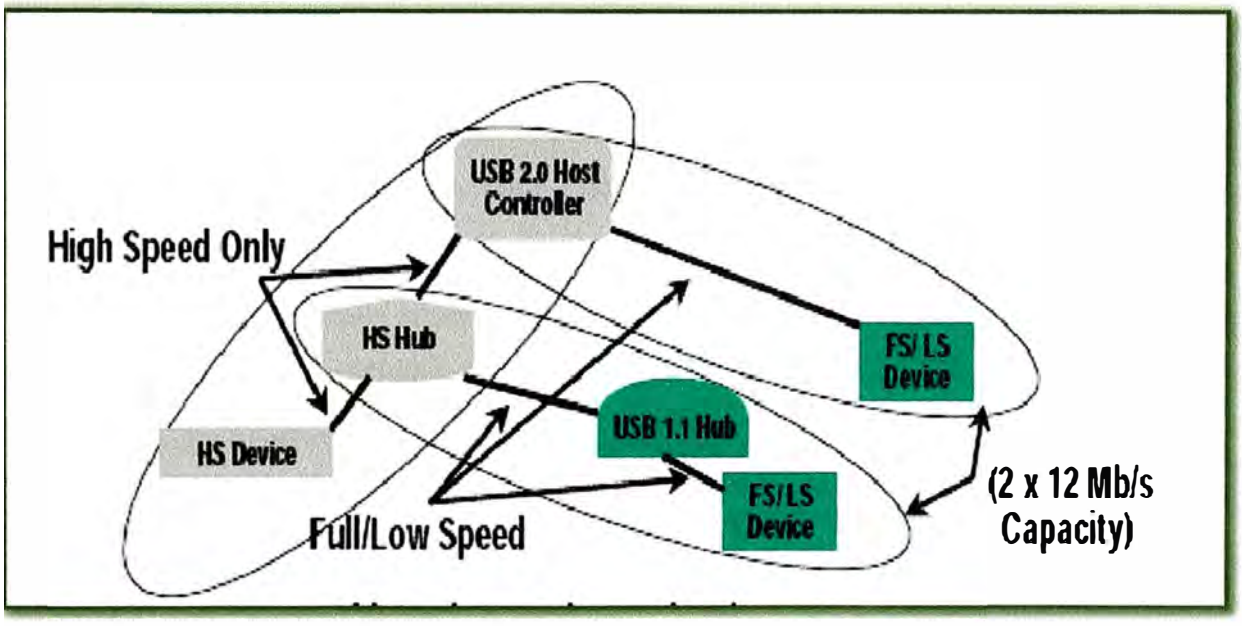


Fig. 2.3 Múltiple Velocidad en un BUS

Estas tasas de transferencias tienen ciertas características propias de funcionamiento y configuración.

El cable que típicamente es usado no es largo, debido a la velocidad de transferencia y tiene la estructura mostrada en la Figura 2.4 de la información [1].

Sus terminales son de tipo diferencial y consta de 4 hilos. Físicamente los datos del USB se transmiten por un par trenzado (+D y-D) además masa y alimentación.

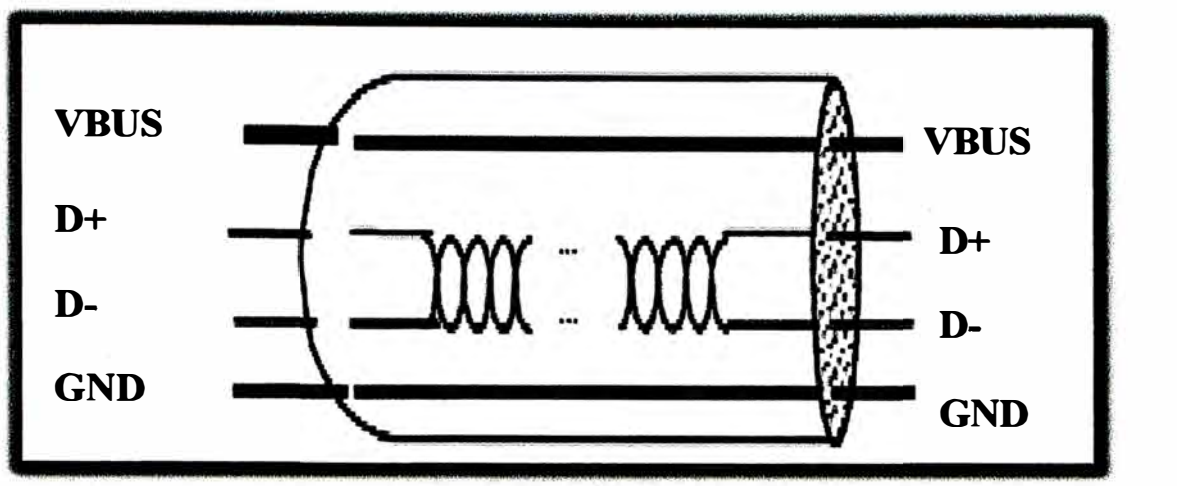


Fig. 2.4 Cable USB

La arquitectura USB comprende algunos tipos básicos de transferencia de datos:

•**Control transfers:** Es usado para configurar un dispositivo al momento de que se conectan. Los datos entregados pueden perderse.

•**Bulk data transfers:** Entrega de datos por volumen el ancho de banda puede variar. Es usado en escáner. La ráfaga de datos es secuencial.

El PIC no puede funcionar como host, ya que se requiere un gran capacidad de manejo de datos para administrar a cada componente del BUS, es suficiente que se le pueda administrar como un dispositivo, para esto se requiere ``memorizarle´´ los protocolos necesarios para enlazarse al host.

Estos protocolos se llaman descriptores y sirve para informarle al host todo lo necesario (Identificación del Endpoint) para que pueda administrarlo.

Los PIC de la serie 18Fxx5x tienen tres modos de funcionamiento.

- USB Human Interface Device (HID): Velocidad baja no requiere driver.
  USB Comuncation Device Class (CDC): Velocidad media requiere driver (Full-speed)es decir 12Mb/s. Crea un Puerto Serie Virtual.
- USB Custom Driver: velocidad alta, este es el modo que usa WinUSB (para Windows vista) y el mphusbapi (Windows 2000 y posterior).

Dentro los protocolos hay que especificar el tipo de transferencia de datos a usar (endpoints), VID&PID, nombre y serie del producto que se conecta para que el host identifique al driver y pueda instalarlo con el fin de que el dispositivo pueda formar las ``pipes´´ o túneles para su comunicación con el host como puede observar en la grafica de enlace virtual. Figura (2.5). [1]

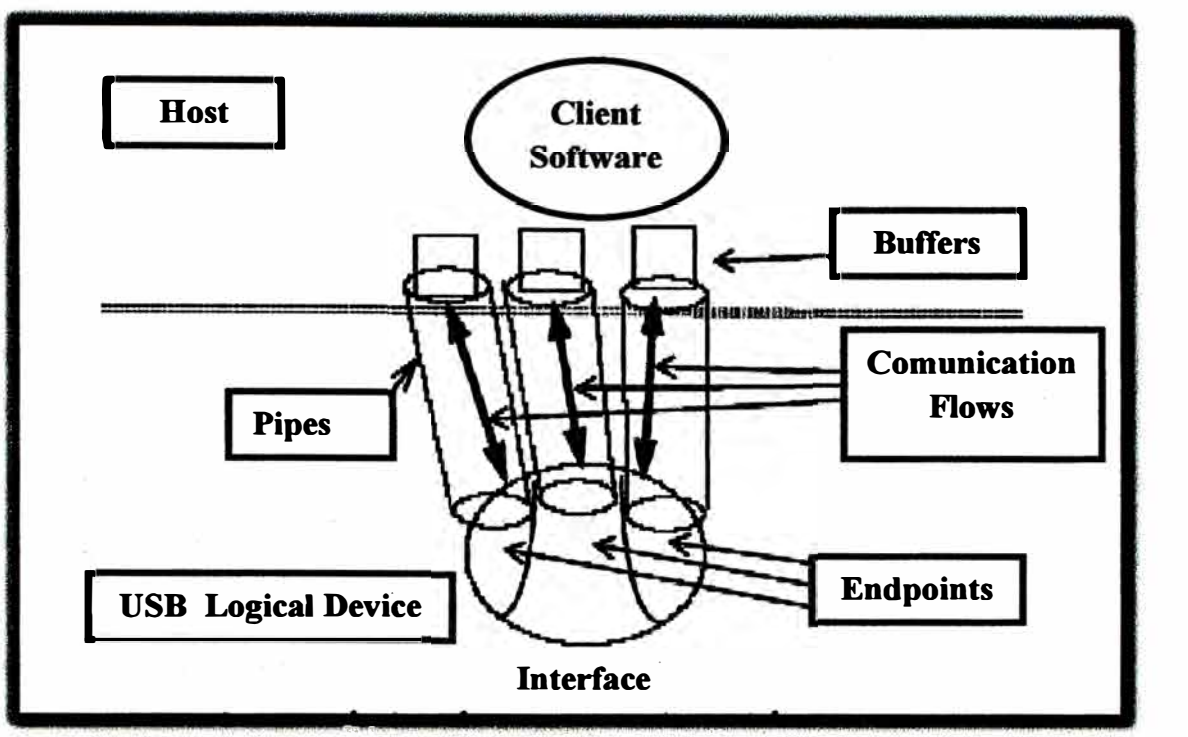


Fig. 2.5 Flujo de Comunicación USB

2.2 Bases teóricas

2.2.1 La MPUSBAPI.DLL de Microchip

Para una mayor facilidad de desarrollo y explicaciones basada en el bus USB Microchip ha creado un archivo de dll en el que proporciona las funciones de acceso al puerto USB con un microcontrolador de la familia PIC18Fxx5x para un funcionamiento correcto se necesita el driver mchpusb.sys .El modo de abrir la pipe es la siguiente:

Primero se identifica si hay un dispositivo con el nombre VID&PID conectado a la PC con la instrucción:

(*MPUSBGetDeviceCount)(PCHARpVID_PID)

La variable pVID&PID, es una entrada de cadena de caracteres que da como resultado el número de dispositivos conectados al host que tiene asignado el mismo pVID&PID.

Seguidamente con la instrucción:

(*MPUSBOpen)

(DWORD instance, // input

PCHAR pVID_PID, // input

PCHAR pEP, // input

DWORD dwDir, // input

Esta instrucción devuelve el acceso al pipe del endpoint con el VID_PID asignado. Todas las pipes se abren con el atributo FILE_FLAG_OVERLAPPED contenida en la DLL, esto permite que la MPUSBRead , MPUSBWrite y MPUSBReandInt tenga un valor de time-out.

El valor de time-out no tiene sentido en una pipe síncrona.

- **Instance: Input:** un número de dispositivo para abrir. Normalmente se utiliza primero la llamada de MPUSBGetDeviceCount para saber cuántos dispositivos hay.

Es importante entender que el driver lo comparten distintos dispositivos. El número devuelto por el MPUSBGetDeviceCount tiene que ser igual o menor que el número de todos los dispositivos actualmente conectados y usando el driver genérico.

Ejemplo: si hay tres dispositivos con los siguientes PID_VID conectados:

- Dispositivo tipo 0,VID 0x04d8,PID 0x0001
- Dispositivo tipo 1,VID 0x04d8,PID 0x0002
- Dispositivo tipo 2,VID 0x04d8,PID 0x0003

y el dispositivo que nos interesa tiene VID=0x04d8 y PID=0x0002 el MPUSBGetDeviceCount devolverá un `1´. Al llamar la función tiene que haber un mecanismo que intente llamar MPUSBOpen() desde 0 hasta MAX_NUM_MPUSB_DEV. Se tiene que contar el número de llamadas exitosas. Cuando este número sea igual al número devuelto por MPUSBGetDeviceCount, hay que dejar de hacer las llamadas porque no puede haber más dispositivos con el mismo VID_PID.

- **pVIP_PID: Input:** String que contiene el VID&PID del dispositivo objetivo. El formato es ``vid_xxxx&pid_yyyy´´ donde xxxx es el valor de VID y el yyyy es el valor del PID los dos en hexadecimal.

Ejemplo: si un dispositivo tiene un VID=0x04d8 y un PID=0x000b, el string de entradas es. ``vid_0x04d8&pid_0x000b´´.

- **pED: Input :** String con el número de endpoint que se va abrir. El formato es ``\\MCHP_EPz´´ o ``\MCHP_EPz´´ dependiendo del lenguaje de programación donde z es el número del Endpoint en decimal.

Ejemplo: ``\\MCHP_EP1´´ o ``\MCHP_EP1´´ Este argumento puede ser NULL (nulo) para crear lazos con Endpoints de funciones no especificas. Las funciones especificas que utilizan este parámetro son: MPUSBRead, MPUSBWrite, MPUSBReadint.

Para utilizar MPUSBReadInt(), el formatode pEP tiene que ser ``\\MCHP_EPz_ASYNC''. Esta operación solo está disponible para un Endpoint interrupción IN. La pipe de datos abierta con ``_ASNYC'' debe almacenar datos con el intervalo especificado en el Endpoint descriptor con un máximo de 100 recepciones. Cualquier otro dato recibido después de llenar el buffer del drive se ignora. La aplicación del usuario tiene que llamar MPUSBReadInt() a menudo sin superar el máximo de 100.

- **dwDir:** Especifica la dirección del Endpoint:
- MP_READ: Para MPUSBRead y MPUSBReadInt.
- MP_Write:para MPUSBWrite.

Se abre un pipe a la vez (hay que usar dos veces esta instrucción) con dwDir=1 se abre la pipe para leer y con dwDir=0 se abre la pipe para escribir al PIC, el resultado que nos arroja esta instrucción es el número de pipe que nos asigna el sistema operativo.

- **dwReserved:** No asignado por el momento el valor por omisión es cero.

El formato típico de la instrucción es: MPUSBOpen (0, vid_pid,out_pipe,dwDir,0).

Como tercer procedimiento, se lee el dato aplicando el número de pipe asignado por medio de la instrucción (*MPUSBRead).

(HANDLE handle, //Input

PVOID pData, //Output

DWORD dwLen, //Input

PDWORD pLength, //Output

DWORD dwMilliseconds) ; //Input

- **handle: Input:** Identifica la pipe del Endpoint que se va a leer. La pipe unido tiene que crearse con el atributo de acceso MP_READ.

En conclusión, ``handle'' es el número de pipe que nos arrojo la instrucción anterior dwDir=1.

- **pData: Output:** Puntero al buffer que recibe el dato leído de la pipe. El formato de dato es un arreglo de N bytes, donde N es el número de bytes que maneja el ``device'' en el arreglo que envía a la PC, generalmente se declara al inicio del programa en el PIC.
- **dwLen: Input:** Especifica el número de bytes que se espera leer de la pipe.

- **pLenght: Output:** Puntero al número de bytes leídos. MPUSBRead pone este valor a cero antes de cualquier lectura o de chequear un error.
- **dwMilliseconds: Input:** Especifica el intervalo de time-out en milisegundos. La función vuelve si transcurre el intervalo aunque no se complete la operación. Si dwMilliseconds=0, la función comprueba los datos de la pipe y vuelve inmediatamente. Si dwMilliseconds es infinito el intervalo de time-out nunca termina.

El formato típico de la instrucción es: MPUSBRead(myInPipe, VarPtr(s(0)), DatosDeseados,Datos,1000)

Para enviar los datos al PIC se hace de la misma manera con la instrucción:

(*MPUSBWrite)

(HANDLE handle, //Input

PVOID pData, //Input

DWORD dwLen, //Input

PDWORD pLength, //Output

DWORD dwMilliseconds); //Input

- **handle: Input:** Identifica la pipe del Endpoint que se va a escribir. La pipe unida tiene que crearse con el atributo de acceso MP_WRITE. En conclusión, ``handle´´ es el número de pipe que nos arrojó la instrucción anterior dwDir=0.
- **pData: Input:** Puntero al buffer que contiene los datos que se van a escribir en la pipe. El formato del dato es un arreglo de N bytes, donde N es el número de bytes que maneja el ``device´´ en el arreglo que recibe de la PC, generalmente se declara al inicio del programa en el PIC.
- **dwLen: Input:** Especifica el número de bytes que se van a escribir en la pipe.
- **plenght: Output:** Puntero que proporciona el número de bytes que se escriben al llamar esta función. MPUSBWrite pone este valor a cero antes de cualquier lectura o de chequear un error.
- **dwMilliseconds: Input:** Especifica el intervalo de time-out en milisegundos. La función vuelve si transcurre el intervalo aunque no se complete la operación. Si dwMilliseconds=0, la función comprueba los datos de la pipe y vuelve inmediatamente. Si dwMilliseconds es infinito, el intervalo de time-out nunca termina.[6]

El formato típico de la instrucción es MPUSBWrite(myOutPipe,VarPtr(SendData(0)), bytes , VarPtr(bytes),1000).

Por último, se requiere cerrar las pipes, porque después de usarlos caducan, ya no es posible leer /escribir de nuevo.

Para cerrarlos basta ejecutar la instrucción: (*MPUSBClose) (*HANDLE handle); de donde handle: Input: Identifica la pipe del Endpoint que se va a cerrar.

El formato típico de la instrucción es: MPUSBClose(myOutPipe) [1]

Existen otras dos instrucciones que no se implementan en este desarrollo, pero es necesario conocerlos, estas son:

MPUSBGETDLLVERSION (VOID)

Lee el nivel de revisión del MPUSAPI.dll. Esta función devuelve la versión del código de la dll en formato hexadecimal de 32bits, no realiza nada con el puerto USB.

El formato típico de la instrucción es: MPUSBGetDLLVersion().

MPUSBREADINT(HANDLE, PDATA, DWLEN, PLENGTH,DWMILLISECONDS)

- **handle: Input:** Identifica la pipe del Endpoint que se va a leer. La pipe unida tiene que crearse con el atributo de acceso MP READ.
- **pData: Output:**Puntero al buffer que recibe el dato leído de la pipe.
- **dwlen: Input:** Especifica el número de bytes que hay que leer de la pipe.
- **pLenght: Output:** Puntero al número de bytes leídos. MPUSBRead pone este valor a cero antes de cualquier lectura o de chequear un error.
- **dwMilliseconds: Input:** Especifica el intervalo de time-out en milisegundos. La función vuelve si transcurre el intervalo aunque no se complete la operación. Si dwMilliseconds=0, la función comprueba los datos de la pipe y vuelve inmediatamente. Si dwMilliseconds es infinito, el intervalo de time-out nunca termina.

El formato típico de la instrucción es: MPUSBReadInt(myOutPipe,VarPtr(SendData(0)), bytes , VarPtr(bytes),1000).

2.2.2 Tipos de Transferencias soportado por estas Instrucciones

En este apartado se recomienda que función utilizar dependiendo del tipo de transferencia.

Tipo	Función	¿Aplicable time-out?
Interrupt IN	MPUSRead, MPUSReadInt	si

Interrupt OUT	MPUSBWrite	si
Bulk IN	MPUSBRead	si
Bulk OUT	MPUSWrite	si
Isochronous IN	MPUSBRead	no
Isochronous OUT	MPUSBWrite	no

Interrupt: tipo interrupción. - Isochronous : tipo síncrono.

`Input´´ y ``output´´ se refiere a los dos parámetros designados en las llamadas a estas funciones, que son lo opuesto a los sentidos comunes desde la perspectiva de una aplicación haciendo llamadas.

2.2.3 Declaración de constantes y variables

Aquí aparecen las constantes y variables que el fabricante recomienda usar. Todas son optativas, dejando la elección al programador. También se comentan las pequeñas variaciones que existen al declarar estas variables en los distintos lenguajes.

MPUS FAIL=0

MPUSB SUCCESS=1

MP WRITE=0

MP READ=1

MAX NUM MPUSB DEV=127

Vid_pid=``vid-04d8&pid-0011´´

En Visual Basic:

Out_pipe=``\MCHP_EPx´´

In_pipe=``\MCHP_EPy´´

En C y Delphi:

Out_pipe=``\\MCHP_EPx´´

In_pipe=``\\MCHP_EPy´´

Siendo x e y números del Endpoint por los que se van a realizar las transmisiones. Está dll se llama de acuerdo a la convención del lenguaje C, NO funciona si es llamada con el formato de STDLIB.[6]

2.3 Definición de Términos

- **USB (Universal Serial Bus).**

Fue ideado para ser un Bus de extensión para la PC con el objetivo de llegar a ser un estándar de la industria.-Las posibles tasas de transferencias lo hace útil para aplicaciones que van desde periféricos de la PC hasta dispositivos de videos u otros que requieren alta tasa de transferencia de datos.

El Bus es independiente de la plataforma de Hardware utilizada, al depender gran parte de su funcionamiento de software.-Otro motivo del Bus es la reducción de su costo dándose esto con la reducción de cables de cobre (por ser un bus de tipo serial).

En la arquitectura USB las capas de Hardware y Software tienen a cargo distintas funciones tanto en el esclavo como en el Host.

- **Elementos del USB**

El USB es un bus ideado para el intercambio de datos entre un computador anfitrión (Host) y dispositivos conectados a él (Esclavos). Los periféricos conectados al USB comparten el ancho de Banda del Bus mediante un protocolo basado en mensajes (tokens).

Un sistema USB consta de tres partes:

- Anfitrión USB (USB Host o Host).
- Dispositivos USB (USB devices).
- Interconexión USB (USB interconnect).

Existe un solo host en el sistema USB, los dispositivos USB proveen servicios o funciones al host y la interconexión USB es la que soporta el trafico entre host y los dispositivos USB es el canal de comunicación.

- **Direccionamiento**

El direccionamiento de dispositivos esclavos USB se hace mediante un número de 7 bits, lo que hace posible direccionar hasta 128 dispositivos y en caso de utilizar HUB (Concentrador) en teoría pueden conectarse hasta 127 dispositivos USB.

Dentro de cada dispositivo lógico USB existe uno o más Endpoints, que son receptores independientes de datos. El ruteo de los datos es de tipo Broadcast, es decir todos los esclavos dentro de una jerarquía reciben los datos pero solo la procesa el esclavo al que están destinados dichos datos.

- **Acceso al Canal de Comunicación**

El protocolo de acceso al canal de comunicación (media access protocol) es de tipo polled, es decir cuando el host se comunica con un esclavo, lo hace mediante el envío de un token con la dirección de dicho esclavo al canal de comunicación (bus) y solo el esclavo con esta dirección procesa el pedido del host.

- **Transmisión y Codificación**

Los datos son transmitidos en forma serie, en 2 líneas de datos complementarias que son denominadas D+ y D-. Además se proveen dos líneas de alimentación y de masa respectivamente, las cuales pueden servir para que el dispositivo tome alimentación del host (5 V, 500mA) de esta forma habilitar al PIC para su respectivo trabajo.

Para transmitir los datos en forma serie se utiliza la codificación Non – Return – To- Zero – Inverted (NRZI), en este tipo de codificación, un 0 (cero) se representa sin un cambio de nivel de tensión, y un 1(uno) se representa con un cambio de nivel de tensión.

Conjuntamente se utiliza el bit stuffing, técnica que consiste en insertar un 0 (cero) cada 6 (seis) 1s (unos) consecutivos en el flujo de bits además, del bit stuffing y de la codificación NRZI se utilizan CRCs. Los CRCs se generan después del bit stuffing.

- **Protocolos USB**

En el USB los datos se envían en paquetes, a su vez los `paquetes se agrupan para formar las transacciones y las transacciones se agrupan para formar las transferencias.

Las transferencias son las estructuras de datos que tienen sentido para el software client que corre en el Host y que es destinatario final de los datos enviados o recibidos desde el dispositivo lógico.

- **Endpoints, Buffer y Pipes**

Cada canal lógico de comunicación entre el host y el esclavo está formado por una triada de componentes (Endpoints- Pipe- Buffer).

El Endpoint pertenece al esclavo USB el Buffer pertenece al Host y la Pipe es la conexión en un enlace virtual entre ambos.

Existen dos básicos de Endpoints: Los de control y los de datos.

Los de control son utilizados para transferir información de configuración y estado entre esclavo y el Host, los de datos son utilizados para transferir datos que utiliza el software

client que corre en el Host, siendo cada Endpoint identificado por un número.

Los Endpoint de Control pueden transferir datos en ambas direcciones.

Los Endpoints de datos según la dirección en la que transfieren los datos se clasifican en:

•**IN:** Los datos van del esclavo al Host.

•**OUT:** Los datos van del Host al esclavo.

Los Endpoint se agrupan para formar interfaces, una interface representa a un dispositivo lógico USB. Cada tipo de endpoint está asociado a un tipo de transferencia.

- **Transferencias**

Una transferencia es un bloque de datos que conforma una estructura comprensible para el Host o el Esclavo, existiendo 2 tipos básicos de transferencias que están directamente relacionados a los tipos de Endpoints: Las de Control y las de Datos.

Dentro de las transferencias de datos existen 3 tipos:

- Transferencia de Interrupción (Interrupt Data Transfers): Realiza baja tasa de Transferencia de datos.
- Transferencia de Bultos (Bulk Data Transfers): Transfiere cantidades relativamente grande de datos en un solo envío. El ancho de banda disponible para este tipo de transferencia varía en función de la disponibilidad de éste .
- Transferencias Isócronas (Isochronous Data Transfers): Ocupan un ancho de banda del USB, cuya latencia es negociada en la configuración y sirven para transmitir datos a intervalos regulares de tiempo.

Cada Endpoint de un dispositivo USB está asociado a un y solo un tipo de transferencia. El Endpoint0 siempre realiza control transfers y todos los esclavos USB lo tienen. A la Pipe asociada al Endpoint0 de la denomina Default Control Pipe.

El Host obtiene la información sobre el tipo de transferencia que realizará cada Endpoint durante el proceso de enumeración cuando lee los Descriptores.

Las transferencias de Control siempre inician con un paquete SETUP, el cual tiene información sobre la función de configuración que el Host desee que el esclavo realice. Las transferencias de datos siempre comienzan con paquetes IN u OUT.

- **Comportamiento de esclavos USB**

Un esclavo USB tiene una cantidad finita de estados posibles, estos pueden ser:

- **Attached:** Es cuando el Esclavo se encuentra apenas conectado físicamente al bus USB.
- **Powered:** Es cuando se ha aplicado tensión al dispositivo o cuando se ha alimentado con una fuente externa.
- **Default:** Un esclavo USB está en este estado después haber recibido una señal de Reset USB. Después de resetearse, éste puede comunicarse con el Host mediante el Endpoint0. Cuando el proceso de reset ha sido completado con éxito, el dispositivo es capaz de comunicarse a la velocidad correcta. Después de haber sido reseteado el dispositivo debe ser capaz de responder correctamente y devolver al Host los Device Descriptor y Configuración Descriptor que contienen la información sobre el Esclavo
- **Addressed:** Un Esclavo USB se encuentra en este estado una vez que le ha sido asignada una dirección USB por el Host.
- **Configured:** Antes de que el Esclavo USB pueda prestar su utilidad, éste debe ser configurado. Desde el punto de vista del dispositivo, éste estará configurado una vez que se haya podido procesar correctamente un Request del tipo SetConfiguration().
- **Suspended:** Este estado es usado para ahorrar energía. Un esclavo entra en este estado cuando no ha recibido un paquete de información durante 3ms. Un dispositivo debe salir de este estado cuando detecta actividad en el bus. Durante este estado se mantiene la dirección USB que fue asignada al dispositivo USB.

▪ **Enumeración**

Se denomina Enumeración (bus enumeration) al proceso, por el cual el Host identifica a un dispositivo USB leyendo sus descriptores y luego le asigna una dirección USB. Además de detectar cuando un dispositivo es removido del bus liberar la dirección USB que este tenía asignada y liberar los demás procesos asociados a las Pipes creadas para comunicarse con dicho dispositivo.

▪ **USB Device Requests**

Todos los dispositivos USB responden a los Requests del Host mediante la Default Control Pipe. Los Requests se manifiestan en las llamadas Control Transfers (Transferencias de Control). Una Transferencia de Control se reconoce porque comienza con un paquete de tipo SETUP a diferencia de los otros tipos de Transferencias que comienzan siempre con paquetes tipo IN u OUT.

Los Requests se dividen en:

- Standard Device Requests: Son comunes a todos los tipos de dispositivos USB sirven para que el Host identifique y configure un dispositivo durante el proceso de Enumeración.
- Class Requests: Son los relativos a cualquier clase particular de dispositivo USB cada clase de dispositivo (excepto las clases genéricas o definidas por los fabricantes, vendor specific) está definida en una especificación de clase USB. Cada clase tiene sus requisitos propios tales como tipo y número mínimo de Endpoints además de otro tipo de Descriptor que es llamado Report Descriptor (Descriptor de Reporte), el cual indica al Host cómo deben interpretarse los datos que envía el Esclavo al Host según la función del Esclavo.
- **Descriptores**

Los Descriptores son tablas en las que los Esclavos almacenan información sobre sus características. Dichas tablas no son modificables por el Host (grabadas en ROM). Los Descriptores son jerárquicos algunos pueden contener información relativa a String Descriptors que tienen información para que el Host muestre al usuario.

El Host solicita los Descriptores a los Esclavos USB mediante las Control Transfers.

Los Descriptores que están en los niveles más altos de la jerarquía de Descriptores, informan al Host sobre la existencia de los Descriptores que están en los niveles más bajos de la jerarquía de Descriptores.

Los Descriptores comunes a todos los tipos de Esclavos USB son:

- Device Descriptor: Contiene información sobre el máximo tamaño de paquete que soporta el Endpoint0, cuántas configuraciones soporta el Esclavo y es el primero que lee el Host. .
- Configuration Descriptor: Existe uno por cada posible forma de operar del Esclavo (solo puede haber una configuración activa en un determinado momento). Tiene información sobre cuántas Interfaces existen por configuración.
- Interface Descriptor: Tiene información sobre el número de Endpoints (excepto el Endpoint0) que utiliza al Interface y sobre la Clase a la que pertenece.
- Endpoint Descriptor: Existe uno por cada Endpoint de una Interface. Tiene información sobre el número de Endpoint. También sobre el tipo de Transferencia que realiza (Control, Interrupt, Bulk o Isochronous) y también tiene información sobre el tamaño máximo de paquete del Endpoint.

Los Descriptores básicos de las Clases USB son:

- Class Descriptor: Especifica a qué Clase USB pertenece una Interface. También da información sobre la longitud del Report Descriptor.
- Report Descriptor: Tiene información sobre cómo debe el Host interpretar las Transferencias del Esclavo. Su estructura es totalmente diferente al del resto de los Descriptores. En el contexto del USB una Transferencia es equivalente a un Report y es un bloque de datos que el Host puede interpretar.

- **Modelo de Capas de USB**

El USB tiene su modelo particular en tres capas. Por lo general, las dos capas superiores se implementan en software y la inferior en hardware tanto en el Esclavo como en el Host.

La Function Layer (Capa de Función) está formada por las Interfaces, cada Interface está formada por un grupo de Endpoints. Los datos que se mueven no tienen formato USB tienen la estructura definida en el Report Descriptor; son los datos que corresponden al par Client Software-Function.

En el USB Device Layer (Capa de Dispositivo USB), la información transmitida entre los pares son las Transacciones que son entre Endpoints y Buffers (estos últimos implementados en software en el Host). En el USB Bus Interface Layer los datos intercambiados entre los pares son Transacciones. Estas Transacciones están encuadradas dentro de los Frames generados a intervalos regulares de tiempo por el Host Controller.

- **Clase HID**

El nombre HID es la abreviatura de "Human Interface Devices". Esta Clase cuya versión actual es el estándar HID 1.11 fue ideada con el propósito de englobar a dispositivos que permitan la interacción del usuario (ser humano) con el Host. Por lo tanto, los requerimientos de ancho de banda son mínimos y la transferencia de datos debe ser confiable.

Los datos que los dispositivos HID envían al Host son interpretados por el HID Class Driver del sistema operativo, para luego ser utilizados por la aplicación que los requiera (Client Software). Los dispositivos HID se comunican con el HID Class Driver mediante la (Default) Control Pipe o mediante Interrupt Pipe.

Los requisitos para la implementación de un dispositivo HID son:

- Control Endpoint (Endpoint0): obligatorio

- Interrupt IN Endpoint: obligatorio.
- Interrupt OUT Endpoint: opcional.

- **Implementación de descriptores**

Cada tipo de Descriptor se declaró como una estructura y luego se inicializaron con sus correspondientes valores, excepto el Report Descriptor que se declaró como un arreglo de caracteres.

Ya que cuando el Host solicita el Configuration Descriptor, el Esclavo debe enviar este y todos sus Descriptores subordinados en forma concatenada, se creó una estructura que los contuviera a todos y es esta estructura la que se devuelve cuando se solicita el Configuration Descriptor.

- **Implementacion de Standard Requests**

Cada Standard Request se implementó como una función separada en el firmware. Para identificar el tipo de Standard Request enviado por el Host, se leen los campos de bits del paquete SETUP de la Control Transfer que contiene el Request.

La determinación de los parámetros del Standard Request solicitado también se hace en base a una variable que forma parte del SETUP Packet. Como ejemplo, puede citarse que al Standard Request GetDescriptor() le corresponde un parámetro que es el Descriptor solicitado. Los parámetros son dependientes del Request.

Existe un flag en los registros de control del módulo USB del Esclavo que indica cuando un paquete es de SETUP.

- **Implementación de Class Requests**

La decodificación de los Class Requests se hace con el mismo método que la decodificación de los Standard Requests. Con la única salvedad que los Class Requests y sus parámetros son con referencia a la Clase USB implementada.

- **Reports**

Todos los Reports (enviados al Host cuando este los solicita mediante el Endpoint correspondiente) deben tener siempre el tamaño y estructura declarados en el Report Descriptor, los cuales dependen de la función a implementar en el Esclavo en caso de no cumplirse esto el Host los recibirá pero los descartará. [1]

CAPITULO III
METODOLOGIA PARA LA SOLUCION DEL PROBLEMA.

3.1. Alternativas de Solución

El objetivo principal del presente trabajo es integrar el software de MATLAB con el PIC18F2550 de Microchip con el fin de diseñar una tarjeta de adquisición de datos ajustada a la necesidad personalizada de cada desarrollo.

Se tiene tres objetivos particulares y son:

- Enlace a la PC mediante USB.
- Enlace de MATLAB - PIC.
- Pruebas finales y correcciones.

3.1.1 Enlace a la PC mediante USB.

Para lograr el enlace a USB se utilizaron las funciones USB incorporadas en el lenguaje C del programa CCS para PICs, dichas funciones están preparadas para que el microcontrolador sea reconocido como un dispositivo personalizado usando los descriptores que incluye el mismo lenguaje se estableció la cantidad de datos a 64 bytes (8 bits por byte) de envió y recepción hacia la PC, luego en la PC se descarga los driver que nos proporciona Microchip en su página Web.

Por parte del hardware, el mismo compilador trae en la ayuda un diagrama esquemático para conectar al PIC dispositivos adicionales como el teclado, display genéricos de dos líneas y gráficos, el puerto serie, USB, I2C.[1]

3.1.2 Configurando el Hardware.

1. Conecte el PIC como se muestra en el diagrama esquemático al final del capítulo.

2. Antes de compilar el código de programa <<daq.c>> adjunto en este archivo comprimido con PCWH Compiler de CCS versión 3.246 ó posterior, primero se escoge el PIC18F2550, en la sección #include al inicio del programa.

3. Verifique que la configuración del PLL corresponda a la frecuencia del Xtal que utiliza.

Ejemplo:

PLL1…para Xtal de 4Mhz.

PLL2…para Xtal de 8Mhz.

PLL3…para Xtal de 12Mhz.

PLL4…para Xtal de 20Mhz.

4. Abra el archivo C:\Archivo de programa\PICC\Drivers\usb-desc_scope.h (donde se instaló el compilador de CCS) que es el descriptor del USB ubicado en su PC, avance hasta la sección start device descriptors (aproximadamente en la línea 132) y reemplace los valores del vendor id, el product id y el device release number como sigue (puede copiar las tres líneas siguiente y pegar en el archivo del descriptor <<usb_desc_scope.h>>:

0xD8,0x04, //vendor id (0x04D8 es Microchip)

0x0B, 0x00, //product id

0x01, 0x00, //device release number

5. Compile el programa y grábelo en el PIC.

Nota: De no completar estos pasos la PC **NO** detectará al PIC.

3.1.3 Configurando el Software

1. La DLL que proporciona Microchip se puede descargar desde su sitio web. (www.microchip.com).Asegúrese de obtener la versión más actual. En la dirección web que se menciona en la bibliografía se descarga el driver (link de acceso directo), en caso de haber caducado busque en la sección Desing→Aplication Desing Center→Wired Conectivity→ USB→Software/Tools→<<MCHPFSUSB Framework v2.4>> ó simplemente escriba ``usb software/tools´´, en la ventanita de búsqueda y dar un click en el botón ``site search´´, generalmente en el acceso a la pagina queda en los primeros resultados de la búsqueda el cual, al darle click lleva directamente al driver. En el mismo paquete existenejemplos que incluye el programa fuente para la comprensión de su uso.

2. Ejecute el driver descargado en el paso anterior e instale en la dirección que trae ya predeterminada. Este ejecutable trae muchos ejemplos de aplicación, entre ellos trae el driver que queda ubicado en:

``C: \MICROCHIP SOLUTIONS\USB TOOLS\MCHPUSB CUSTOM DRIVER \MCHPUSB DRIVER\ RELEASE\´´

3. Instale el hardware a la PC de manera similar al que se instala un dispositivo USB que adquiere en el mercadoconecte el dispositivo a la PC, en cuanto le solicite los driver solo proporcione la dirección donde fue descomprimido el driver (la misma dirección del paso anterior).

Si todo es correcto debemos observar en el administrador de dispositivos un nuevo hardware que se agrego tal como se muestra en la Figura 3.1.

Si se olvida ó no se sustituye correctamente en el descriptor las 3 líneas que se comentan en el código del programa el compilador CCS compilara correctamente pero al conectarse el PIC en la PC, este no reconocerá el driver.

Todo esto se desarrolla basándose en ejemplos del compilador que trae incluidos en la carpeta de ``examples''. Cabe aclarar que los diseños en USB con PIC día a día se van incrementando. Solo basta investigar un poco en la red para ver los resultados. El código del programa del PIC se encuentra al final del capítulo con sus respectivos comentarios. [1]

Fig. 3.1 Instalación del PIC en la PC

4. En las propiedades del dispositivo instalado se puede observar en la Figura 3.2 el número PID&VID que se configuró en el PIC.

Fig. 3.2 En propiedades del PIC instalado en la PC se observa el numero de PID&VID.

3.2. Enlace de MATLAB al PIC para Transferencia de Datos

Para poder iniciar el enlace con el PIC es necesario que se haya concluido satisfactoriamente la instalación del paquete de drives de Microchip. En el mismo paquete que se descarga de la red obtenemos instrucciones y ejemplos en C, que muestran como manipular el driver y los parámetros requeridos para aplicarlo. Sin embargo aun no es suficiente la información, porque la dificultad está en identificar el tipo de variable que acepta cada software de desarrollo y que también sea aceptada por la DLL. Por lo que de nuevo se sugiere consultar las notas de aplicaciones de Microchip y ejemplos publicados en la web (con cualquier software de desarrollo).

Para hacer la conexión con MATLAB simplemente se ``copiaron'' las instrucciones al lenguaje de MATLB y se acondicionaron las variables para que la DLL pueda reconocerlo.

Nota: Adjunto a este archivo comprimido se encuentra el código de MATLAB usb.m que es un demo de cómo el programa envía y recibe datos del PIC los archivos-mpusbapi.c y mpusbapi.dll son necesarios para la ejecución en MATLAB y deben estar en la misma carpeta que el archivo usb.m.

Después se explica el desarrollo de la comunicación de ida y vuelta; primero es necesario conocer la función de MATLAB dentro del proyecto para implementar su respectivo software y tareas a cumplir.

Existe varios métodos pero el que se utilizó fue manipular directamente la dll con la instrucción ``loadlibrary´´; ya que manipula la librería de una manera directa y sin intermediarios.

Para iniciar con la implementación del código primero inicie el programa de MATLAB.

Las instrucciones que se requieren implementar tienen la siguiente secuencia y formato en MATLAB y son:

a. Primero copiar los archivos-mpusbapi.c y mpusbapi.dll en la misma carpeta de trabajo (se obtienen de la descarga del driver en la página de microchip (``Microchip MCHPFSUSB v2.4 installer.zip´´)), al instalarse queda ubicado en X:\Microchip Solutions\USB Tools\MCHPUSB Custom Driver\Mpusbapi\Dll\Borland_C, en caso de descargar una versión de driver más reciente reemplace estos archivos por los más nuevos.

b.Se abre el editor de MATLAB y comenzamos cargando la librería en memoria.

Formato:

Loadlibrary mpusbapi_ mpusbapi.h alias librería.

c.Luego se identifica el número de dispositivos conectados con el PID&VID y ubicar el que corresponde al hardware de su desarrollo.

Formato:

[Conectado]= calllib (`librería´,`MPUSBGetDeviceCount´, vid_pid_norm)

donde:

Vid_pid_norm=libpointer('int8Ptr',[uint8('vid_04d8&pid_000b')0]);

d.Seguidamente abra la pipe para leer.

Formato:

[my_out_pie] = calllib('librería'.'MPUSBOpen',unit8 (0), vid_pid_norm,

out_pipe,int8(0), uint8(0));

donde:

Vid_pid_norm=libpointer('int8Ptr',[uint8('vid_04d8&pid_000b')0]);

Out_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1')0]);

e.Siguiendo con la secuencia, abra la pipe para escribir.

Formato:

[my_in_pipe] = calllib('librería'.'MPUSBOpen', unit8 (0), vid_pid_norm,in_pipe,uint8(1),uint8(0));

donde:

vid_pid_norm=libpointer('int8Ptr',[uint8('vid_04d8&pid_000b')0]);

in _pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1')0]);

f.Lea los datos de la pipe (solamente si la pipe está abierta).

Formato

[aa,bb,data-in,dd] = calllib('librería','MPUSBRead', my_in_pipe, data_in,uint8(64),uint8(10));

De donde:

data_in=eye(1,64,'uint8');

g.Escriba los datos de la pipe (solamente si la pipe está abierta).

Formato:**calllib('librería','MPUSBWrite', my_out_pipe, data_out,uint8(64) uint8(64),uint8(10));**

donde:

data_out=eye(1,64,'uint8');

h.Cierre la(s) pipe(s) abierta(s), cada vez que finalice el programa, ya que si quedan abierta Windows genera errores y se pierde la comunicación.

Formato:

calllib('librería','MPUSBClose', my_in_pipe);

calllib('librería','MPUSBClose', my_out_pipe);

NOTA: Al terminar el programa descargue la librería de memoria ya que no se puede cargar más de una vez.

Formato:

Unloadlibrary libreria;

Una vez enlazado con el PIC los datos pueden fluir las veces que sea necesario de un sentido a otro y manipularlos como desee ya que se tiene el completo control el software del PIC (por parte del Compilador C) y en la PC (por parte de MATLAB).En caso de perderse la comunicación con el PIC (en casos donde el programa MATLAB genere errores por cuestiones ajenas a la comunicación) desinstale del dispositivo ``Microchip Custom USB Device´´ desde el administrador de dispositivos, desconecte el PIC del puerto USB, descargue la librería de memoria desde el MATLAB con ``unloadlibrary librería´´ en la línea de comandos de MATLAB y resetee el PIC.

Conecte de nuevo el PIC al puerto USB de su computadora y con eso es suficiente para restaurar las comunicaciones entre el MATLAB y el PIC. [1]

3.3. Pruebas Finales y Correcciones

Para el enlace con la PC y USB es muy importante conectar el capacitor (C4) de 0.1uF como se indica en el diagrama, ya que, si se omite generará un error al momento del enlace. También es muy importante habilitar el fusible VREGEN para que Vusb sea activado.Cuidar de no invertir las terminales D- y D+ del conector USB al momento de implementar el Hardware.

Es importante considerar la función del PLL del PIC ya que de acuerdo al valor de la frecuencia del Cristal depende el valor de la multiplicación de la frecuencia del reloj para generar los 48MHZ necesarios para el USB.

Respecto al descriptor del USB es importante considerar el valor del VID&PID ya que si no es configurado correctamente, el driver que proporciona Microchip no lo reconocerá y en consecuencia, no funcionara. Para esto no olvidarse sustituir en el descriptor las 3 líneas que se comentan en el código del programa que anexa en este capítulo.

La DLL que proporciona Microchip se puede descargar desde su sitio web.Asegúrese de obtener la versión más actual. El mismo paquete incluye ejemplos que incluyen el programa fuente para la comprensión de su uso.Se recomienda verificar el archivo que nos proporciona Microchip en el driver en la siguiente dirección:

"C:\MICROCHIP SOLUTIONS\USB TOOLS\MCHPUSB CUSTOM DRIVER\MCHPUSBDRIVER\MCHPUSB DRIVER RELEASE NOTES.HTM" [1]

3.4 Software (simulación)

Considerando la dirección electrónica www.microchip.com, procedemos descargar los

drivers a trabajar.

Se busco la versión más actualizada (MCHPFSUSB Framework V2.8) la cual nos da elarchivo que contiene los drivers (MCH_App_Lib_v2010_10_19), para luego guardarse en la carpeta USBPIC18F2550 (carpeta de trabajo) tanto drivers como programas de trabajos tienen que estar en la misma carpeta.

Luego mediante MPLAB IDE V8.5 se crea un nuevo proyecto llamado E001 colocándolo en la carpeta del proyecto USBPIC18F2550 listo para su requerimiento, también se crea un programa en C llamado C001.

Estando en MPLAB se trabaja el programa en C se realiza su respectiva compilación para luego ser utilizado para su respectiva grabación del PIC mediante codificación hexadecimal en el PIC18F2550.En este mismo entorno de trabajo mediante MPSIM se acondiciona el programa en C para su simulación en PROTEUS.

Luego se construye el circuito en PROTEUS para su simulación se edita propiedades y se abre el programa en C llamado C001.c.

Se abre MATLAB y en comando Windows se escribe el programa que va administrar la comunicación PC-PIC mediante MATLAB este programa tiene tres etapas de trabajo.

Una etapa inicial que garantiza la comunicación USB –PIC mediante la activación de Leds, a una segunda etapa del programa que realiza tareas de acuerdo a ciertas condiciones de programación y una tercera etapa que es necesaria es un cierre del programa.

3.5 Procedimientos para la realización del software

Se procede a instalar los drivers de Microchip MCH_App_Lib_v2010_10_19 base para las necesidades en siguientes etapas tanto para el programa en C (Figura 3.3) como para el programa en MATLAB (Figura3.4).

MPLAB en su entorno de desarrollo se tiene le código fuente para luego llevarlo a ensamblador con lo cual se creara un programa ejecutable para el microcontrolador.

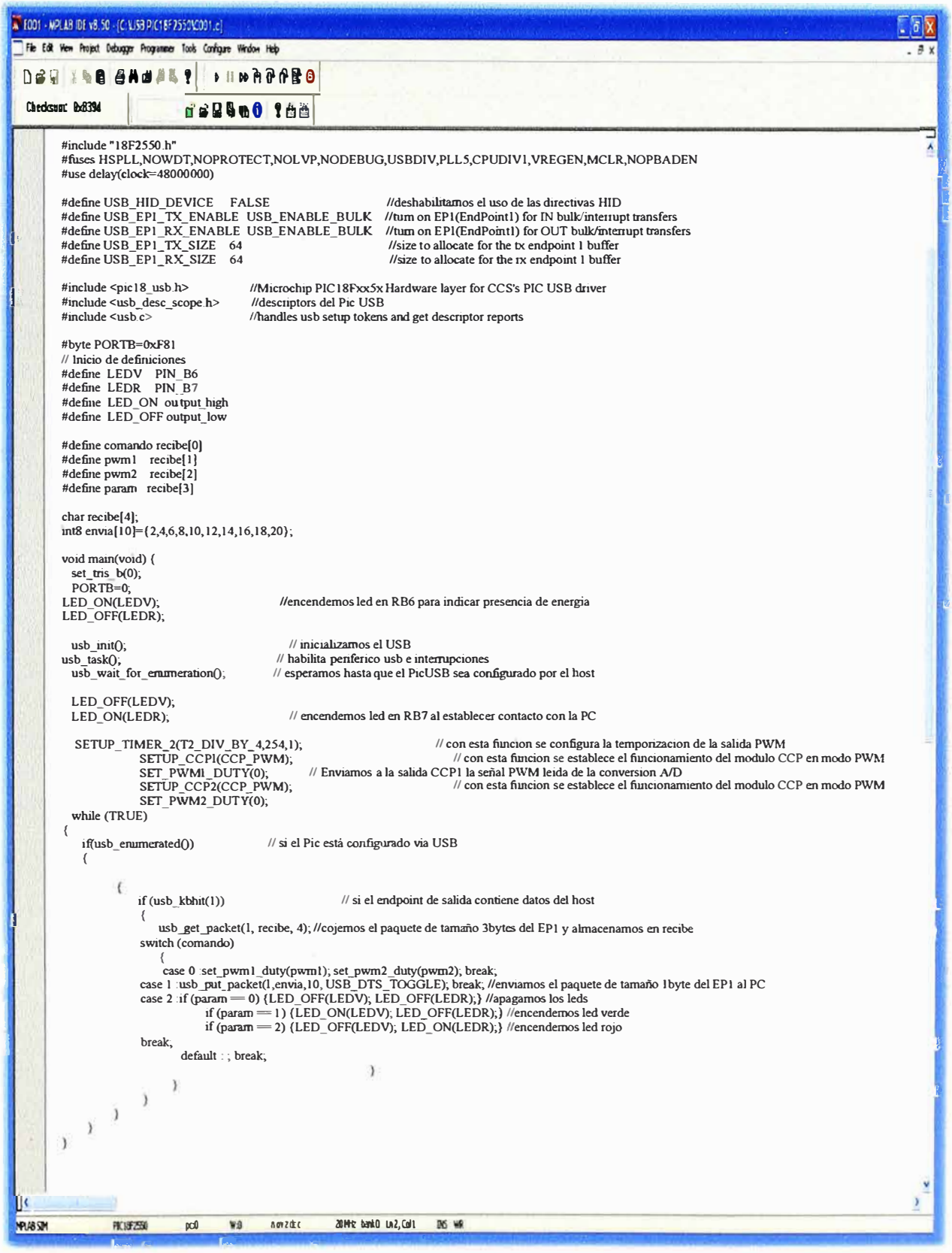
Se crean dos ficheros ejecutables: Uno para depurar el código desarrollado (Debbug Code) u otro el final del programa (Release Code).Se trabaja también para la simulación mediante el simulador MPSIM o depuramos nuestro código mediante dispositivos Hardware, en esta parte de depuración se corrigen los errores de programación.

En MPLAB mediante Proyect Wizard se sigue los siguientes pasos:

- Se escoge el PIC que se va a trabajar para nuestro caso el PIC18F2550.

- El CCS.C Compiler se guarda en tool site .
- Se reconfigura el proyecto y finalmente se tiene el directorio del proyecto.

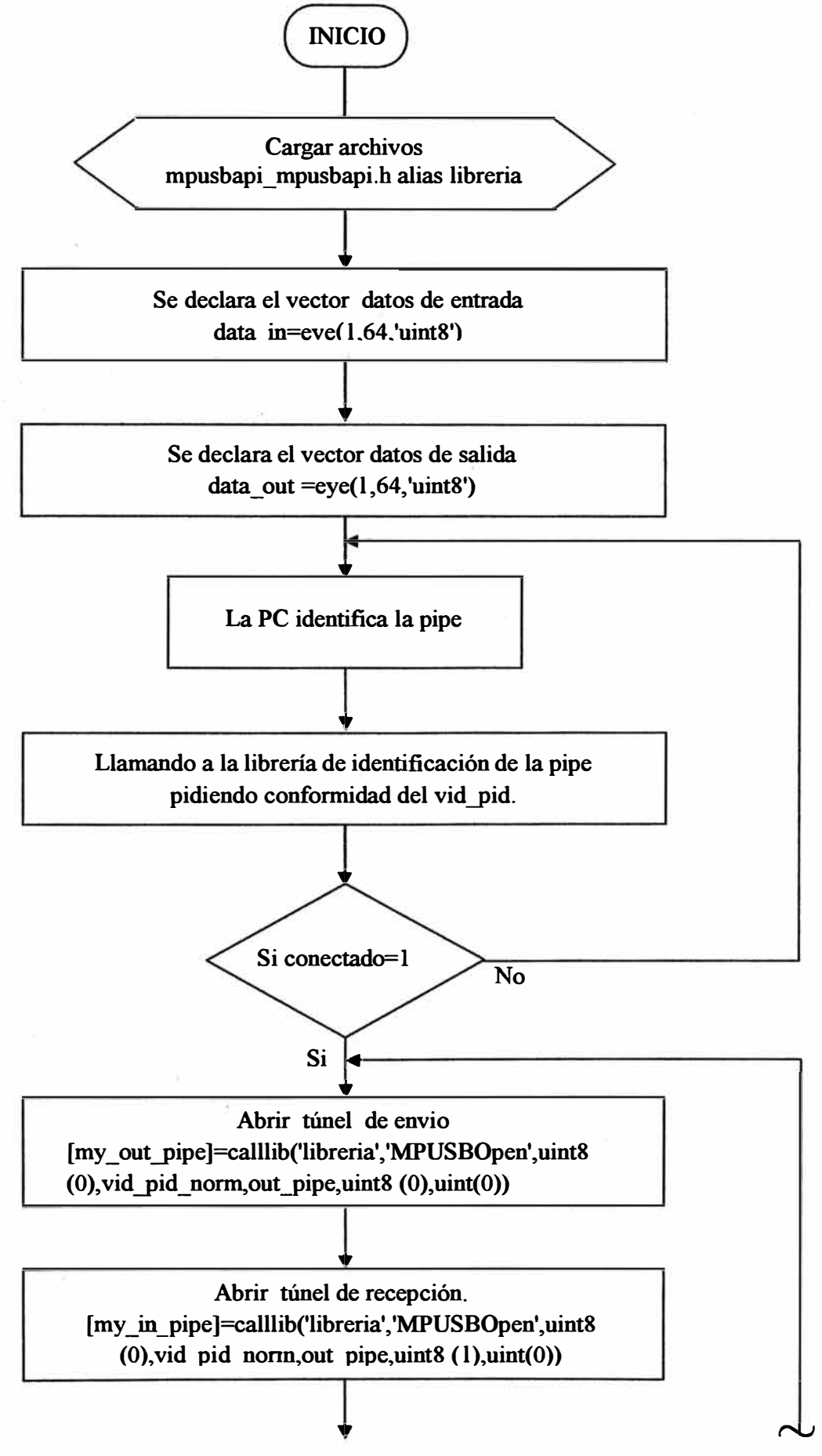
El directorio del proyecto se guarda en la misma carpeta USBPIC18F2550.

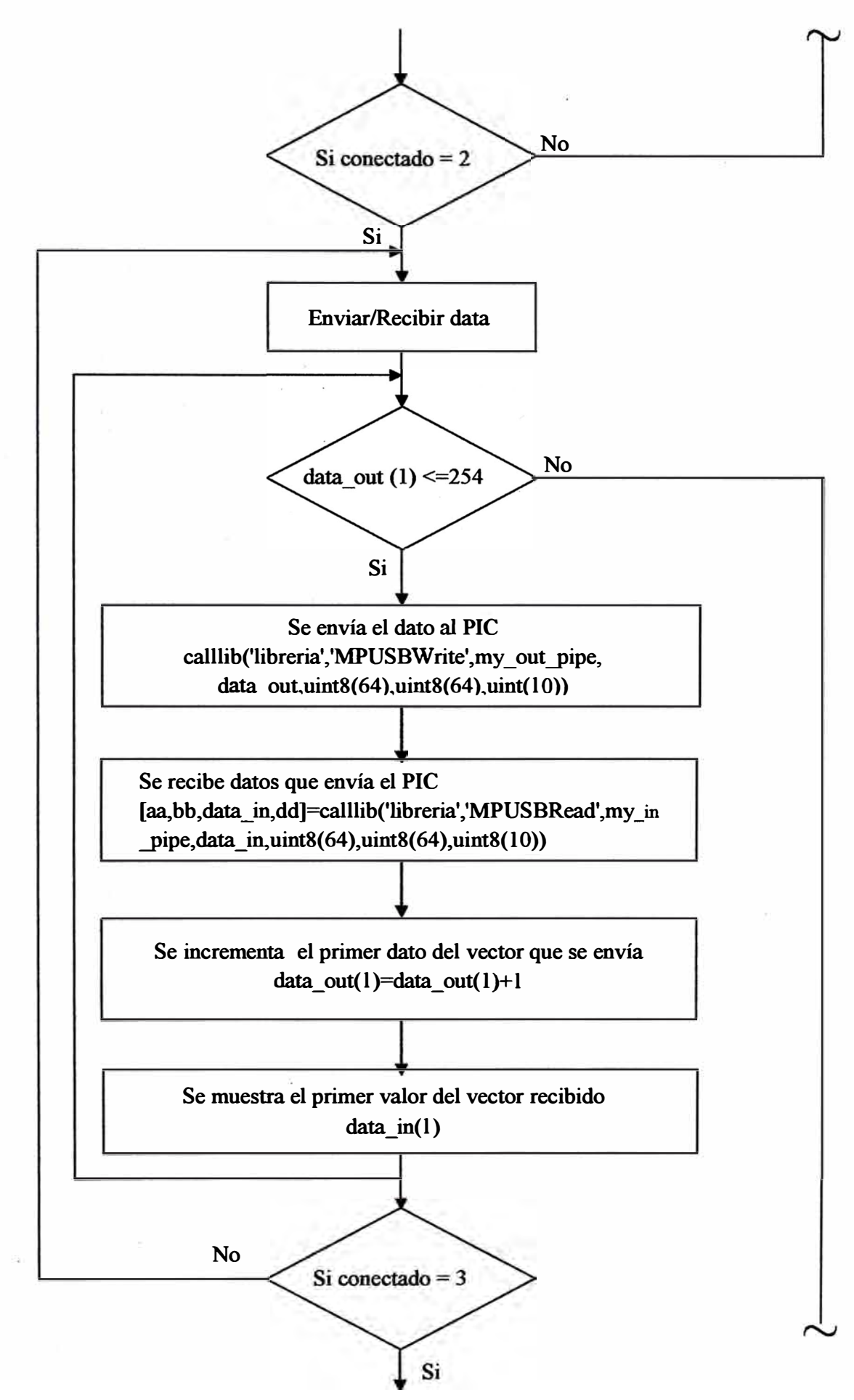


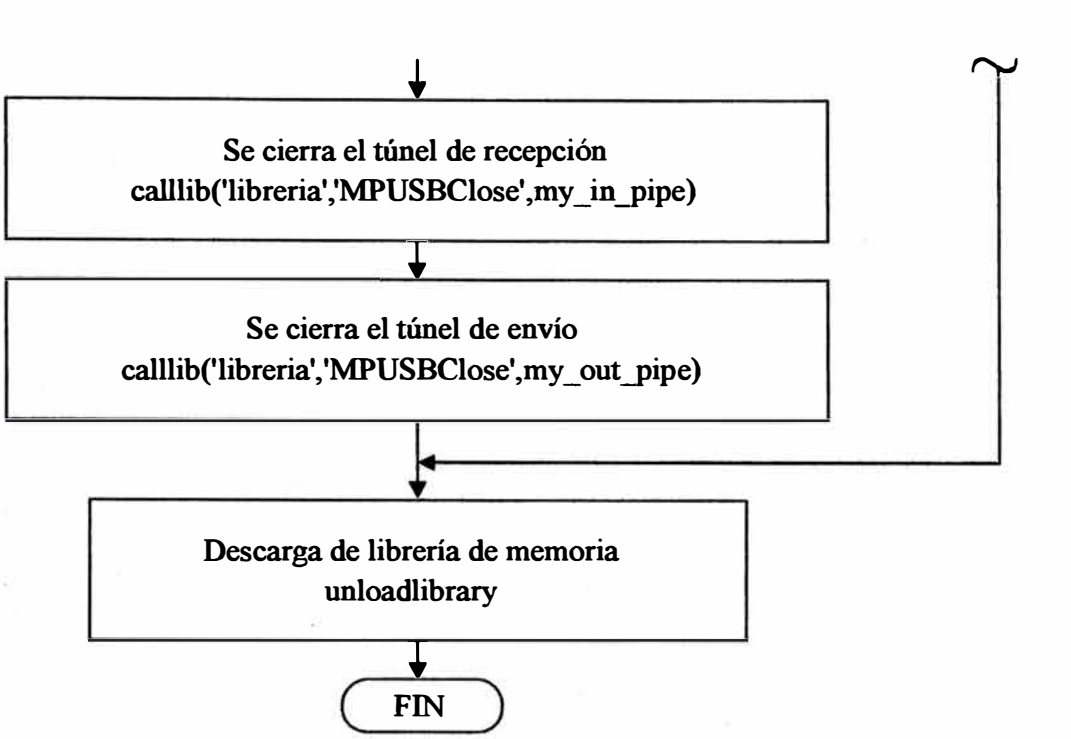
```
#include "18F2550.h"
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN,MCLR,NOPBADEN
#use delay(clock=48000000)

#define USB_HID_DEVICE    FALSE                    //deshabilitamos el uso de las directivas HID
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK   //turn on EP1(EndPoint1) for IN bulk/interrupt transfers
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK   //turn on EP1(EndPoint1) for OUT bulk/interrupt transfers
#define USB_EP1_TX_SIZE   64                    //size to allocate for the tx endpoint 1 buffer
#define USB_EP1_RX_SIZE   64                    //size to allocate for the rx endpoint 1 buffer

#include <pic18_usb.h>          //Microchip PIC18Fxx5x Hardware layer for CCS's PIC USB driver
#include <usb_desc_scope.h>     //descriptors del Pic USB
#include <usb.c>                //handles usb setup tokens and get descriptor reports

#byte PORTB=0xF81
// Inicio de definiciones
#define  LEDV   PIN_B6
#define  LEDR   PIN_B7
#define  LED_ON  output_high
#define  LED_OFF output_low

#define comando recibe[0]
#define pwm1    recibe[1]
#define pwm2    recibe[2]
#define param   recibe[3]

char recibe[4];
int8 envia[10]={2,4,6,8,10,12,14,16,18,20};

void main(void) {
  set_tris_b(0);
  PORTB=0;
LED_ON(LEDV);                   //encendemos led en RB6 para indicar presencia de energia
LED_OFF(LEDR);

  usb_init();                   // inicializamos el USB
usb_task();                     // habilita periferico usb e interrupciones
  usb_wait_for_enumeration();   // esperamos hasta que el PicUSB sea configurado por el host

  LED_OFF(LEDV);
  LED_ON(LEDR);                 // encendemos led en RB7 al establecer contacto con la PC

   SETUP_TIMER_2(T2_DIV_BY_4,254,1);                    // con esta funcion se configura la temporizacion de la salida PWM
              SETUP_CCP1(CCP_PWM);                      // con esta funcion se establece el funcionamiento del modulo CCP en modo PWM
              SET_PWM1_DUTY(0);       // Enviamos a la salida CCP1 la señal PWM leida de la conversion A/D
              SETUP_CCP2(CCP_PWM);                      // con esta funcion se establece el funcionamiento del modulo CCP en modo PWM
              SET_PWM2_DUTY(0);
  while (TRUE)
{
    if(usb_enumerated())        // si el Pic está configurado via USB
    {

        {
            if (usb_kbhit(1))               // si el endpoint de salida contiene datos del host
            {
                usb_get_packet(1, recibe, 4); //cojemos el paquete de tamaño 3bytes del EP1 y almacenamos en recibe
            switch (comando)
                {
                 case 0 :set_pwm1_duty(pwm1); set_pwm2_duty(pwm2); break;
            case 1 :usb_put_packet(1,envia,10, USB_DTS_TOGGLE); break; //enviamos el paquete de tamaño 1byte del EP1 al PC
            case 2 :if (param == 0) {LED_OFF(LEDV); LED_OFF(LEDR);} //apagamos los leds
                        if (param == 1) {LED_ON(LEDV); LED_OFF(LEDR);} //encendemos led verde
                        if (param == 2) {LED_OFF(LEDV); LED_ON(LEDR);} //encendemos led rojo
            break;
                    default : ; break;
                                                    }
                }
            }
        }
    }
}
```

Fig.3.3 Programa en Lenguaje en C (MPLAB)

DIAGRAMA DE FLUJO MATLAB – PIC18F2550




Si conectado = 2
No
Si
Enviar/Recibir data
data_out (1) <=254
No
Si
Se envía el dato al PIC
calllib('libreria','MPUSBWrite',my_out_pipe,
data out,uint8(64),uint8(64),uint(10))
Se recibe datos que envía el PIC
[aa,bb,data_in,dd]=calllib('libreria','MPUSBRead',my_in
_pipe,data_in,uint8(64),uint8(64),uint8(10))
Se incrementa el primer dato del vector que se envía
data_out(1)=data_out(1)+1
Se muestra el primer valor del vector recibido
data_in(1)
No
Si conectado = 3
Si



Se procede a la compilación del programa y después de compilar el programa se va al menú vista para observar lo siguiente:

Proyect: Esta ventana presenta la lista de archivos que actualmente hay en el proyecto.

Output: Esta ventana ofrece información de salida del programa por ejemplo cuando se compila va apareciendo la información sobre la compilación, indicando si hay errores .

Disassembly Listing: En esta ventana se puede ver el código maquina paralelamente al código ensamblador.

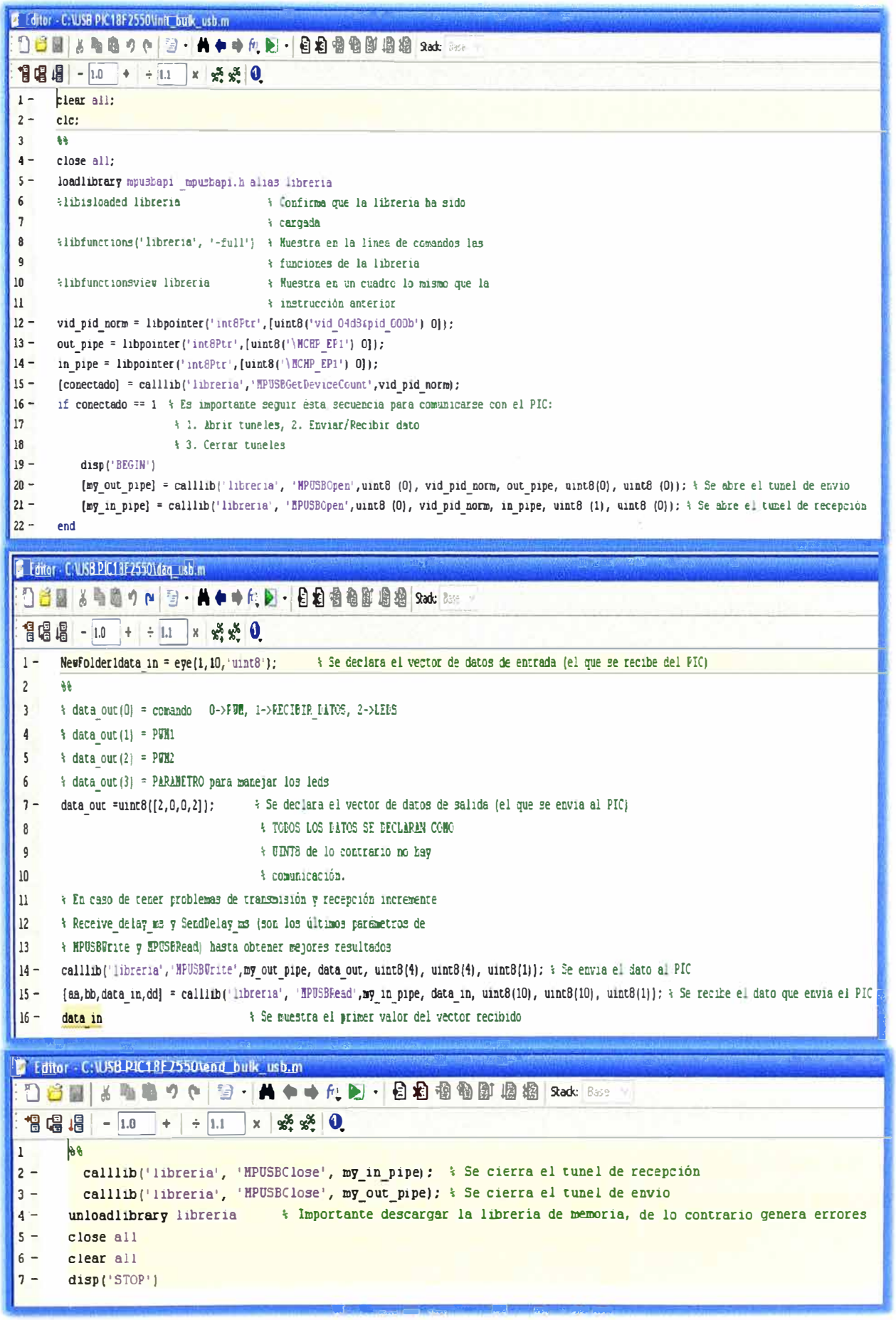
Hardware Stack: Muestra el contenido de la pila.Si la pila se desborda MPLAB indica su desbordamiento con el mensaje Underflow.

Program Memory: En esta ventana se puede apreciar las posiciones de memoria que ocupa cada una de las instrucciones.Puede utilizarse para seguir paso a paso la ejecución del programa.Si pulsamos en cada uno de los tres botones de la parte inferior de esta ventana podemos seleccionar tres formas de ver la memoria del programa:

- **Opcode hex:** Representa la memoria del programa con los datos en Hexadecimal.

Esta opción es muy útil al usar el programador del dispositivo y comprobar si se grabaron bien los datos.

- **Machine:** Esta opción representa el código maquina ensamblado con la información de las etiquetas y direcciones de memoria que tiene asignadas.
- **Symbolic:** Despliega el código hexadecimal desensamblado con los símbolos (etiquetas utilizados en el programa).



```
Editor - C:\USB PIC18F2550\init_bulk_usb.m

clear all;
clc;
%%
close all;
loadlibrary mpusbapi _mpusbapi.h alias libreria
%libisloaded libreria              % Confirma que la libreria ha sido
                                   % cargada
%libfunctions('libreria', '-full') % Muestra en la linea de comandos las
                                   % funciones de la libreria
%libfunctionsview libreria         % Muestra en un cuadro lo mismo que la
                                   % instrucción anterior
vid_pid_norm = libpointer('int8Ptr',[uint8('vid_04d8&pid_000b') 0]);
out_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);
in_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);
[conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm);
if conectado == 1  % Es importante seguir ésta secuencia para comunicarse con el PIC:
                   % 1. Abrir tuneles, 2. Enviar/Recibir dato
                   % 3. Cerrar tuneles
    disp('BEGIN')
    [my_out_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, out_pipe, uint8(0), uint8 (0)); % Se abre el tunel de envio
    [my_in_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, in_pipe, uint8 (1), uint8 (0)); % Se abre el tunel de recepción
end
```

```
Editor - C:\USB PIC18F2550\daq_usb.m

NewFolder1data_in = eye(1,10,'uint8');      % Se declara el vector de datos de entrada (el que se recibe del PIC)
%%
% data_out(0) = comando   0->PWM, 1->RECIBIR DATOS, 2->LEDS
% data_out(1) = PWM1
% data_out(2) = PWM2
% data_out(3) = PARAMETRO para manejar los leds
data_out =uint8([2,0,0,2]);       % Se declara el vector de datos de salida (el que se envia al PIC)
                                  % TODOS LOS DATOS SE DECLARAN COMO
                                  % UINT8 de lo contrario no hay
                                  % comunicación.
% En caso de tener problemas de transmisión y recepción incremente
% Receive_delay_ms y SendDelay_ms (son los últimos parametros de
% MPUSBWrite y MPUSBRead) hasta obtener mejores resultados
calllib('libreria','MPUSBWrite',my_out_pipe, data_out, uint8(4), uint8(4), uint8(1)); % Se envia el dato al PIC
[aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(10), uint8(10), uint8(1)); % Se recibe el dato que envia el PIC
data_in                           % Se muestra el primer valor del vector recibido
```

```
Editor - C:\USB PIC18F2550\end_bulk_usb.m

%%
  calllib('libreria', 'MPUSBClose', my_in_pipe);  % Se cierra el tunel de recepción
  calllib('libreria', 'MPUSBClose', my_out_pipe); % Se cierra el tunel de envio
unloadlibrary libreria      % Importante descargar la libreria de memoria, de lo contrario genera errores
close all
clear all
disp('STOP')
```

Fig.3.4 Programa en MATLAB

3.6 Simulacion en PROTEUS

Se realiza el esquema del circuito (dibujo) para su simulación luego procedemos a grabar virtualmente al PIC del siguiente modo:

- Seleccionamos el microcontrolador.
- Buscamos el menú herramientas source y seleccionamos Add/Remove Source file...
- En el recuadro que aparece hacemos click en la opción new y seleccionamos el archivo a grabar en PROTEUS.
- En la misma herramienta source hacemos click en la opción Build All, para concluir con la grabación del PIC en el entorno PROTEUS.
- Pulsando Play da inicio a la simulación bajo ciertas condiciones.

3.7 Comunicación fluida MATLAB - PROTEUS

El programa MATLAB se ha dividido en tres partes:

init_bult-usb.m : Su tarea principal es cargar las librerías mpusbapi_mpusb.h alias .

El Host da los descriptores para la identificación de los endpoints.Se abre los tuneles de comunicacion se envía/recibe los datos y finalmente se cierra los tuneles de comunicación de datos según los valores de conectado en el programa.

daq_usb.m : Se declara el vector de datos de entrada (el que se recibe del PIC) .

Se define condiciones para las diferentes tareas del data_out que va a cumplir de acuerdo a los requerimientos del programa MATLAB que posteriormente se van a ver reflejados en simulación PROTEUS ante un Play.

end _bulk _usb.m: Se encarga de cerrar los túneles de recepción y de envió para finalmente descargar la librería de memoria cierra el comando windons y origina un STOP.

Estas tareas se consiguen variando algunos parámetros y utilizando opción RUN y son:

- Inicialmente se debe comprobar el encendido de los leds verde para indicar que el prototipo (PIC) está con energía , después se da el encendido del leds rojo para dar una confirmación que se ha originado un enlace de comunicación PIC-PC.
- Variando el data _out se tiene una serie de funciones distintas entre los dos leds de encendido y apagado.

- Variando el data out se tiene una ráfaga de 10 datos en la salida del PIC pero ingresando a la PC.
- Variando el data_out se tiene una modulación de ancho de pulso utilizada para el control de velocidad de motor DC.

Para la implementacion se tiene el circuitosegún figura 3.6

- Variando el data_out se tiene una modulación de ancho de pulso (PWM) utilizada para el sentido de giro de motor DC.
- Finalmente se cierra la comunicación y aparece STOP. (Figura3.5).

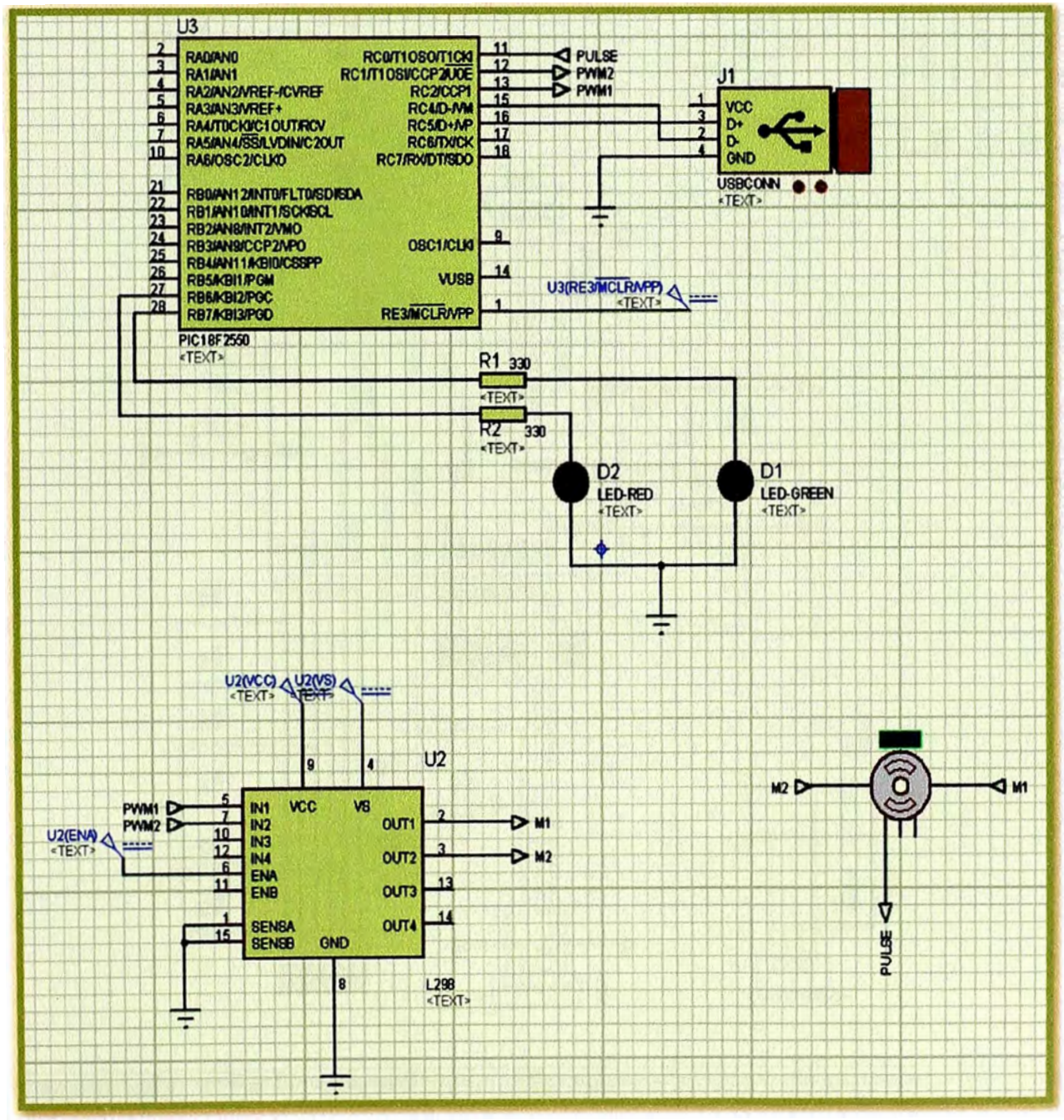
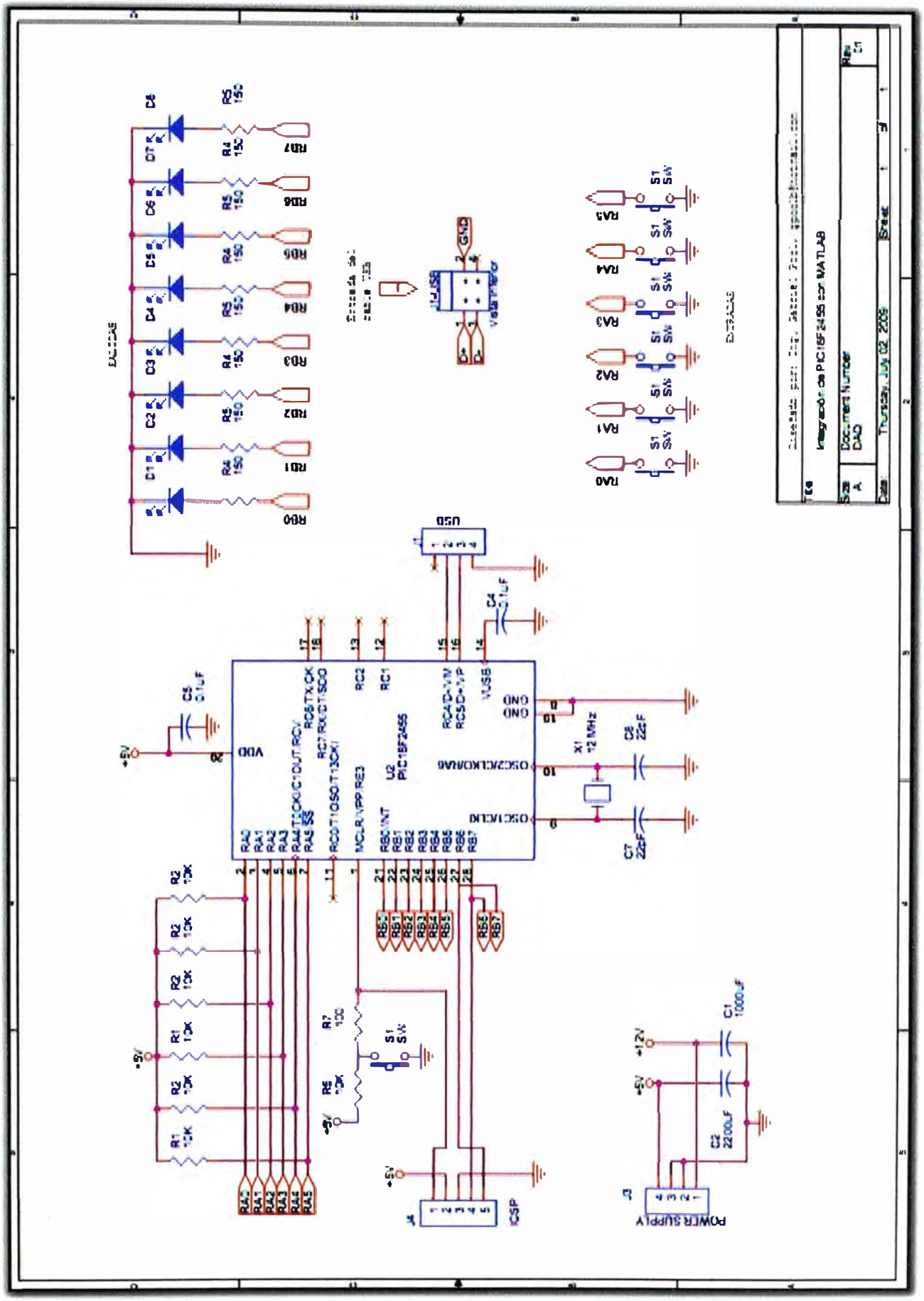


Fig.3.5 Simulacion en PROTEUS



3.6 Esquema para la Implementación.[1]

El proyecto mediante simulación PROTEUS (VMS) muestra resultados importantes que deben comprobarse en la implementación de la tarjeta de adquisición de datos estos resultados son:

- Prueba del enlace de comunicación PIC-PC mediante el encendido de leds.
- En el programa MATLAB variando el vector data _out () de cuatro variables se obtiene diferente alternativas de encendido o apagado de los dos leds.
- También variando el vector data_out() y bajo condiciones de comando se recibe en la PC una ráfaga de 10datos.
- Si al vector data_out() de cuatro variables se varia las variables centrales se obtiene la Modulación de Ancho de Pulso (PWM) variando de esta manera la velocidad del motor DC .
- Si variamos la entrada del Drivers Bidireccional (CI L298) se invierte el sentido de giro del motor DC.

CAPITULO IV
ANALISIS Y PRESENTACION DE RESULTADOS

4.1 Análisis Descriptivo

El objetivo es obtener el modelo matemático del Motor DC que es de la forma:

$$\frac{\dot{\theta} m_{(s)}}{V_{a(s)}} = \frac{K_t}{S[(J_m S + b)](LaS + Ra) + K_t K_e]} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.1)$$

Las ecuaciones de estado se obtienen de la Figura 4.1.

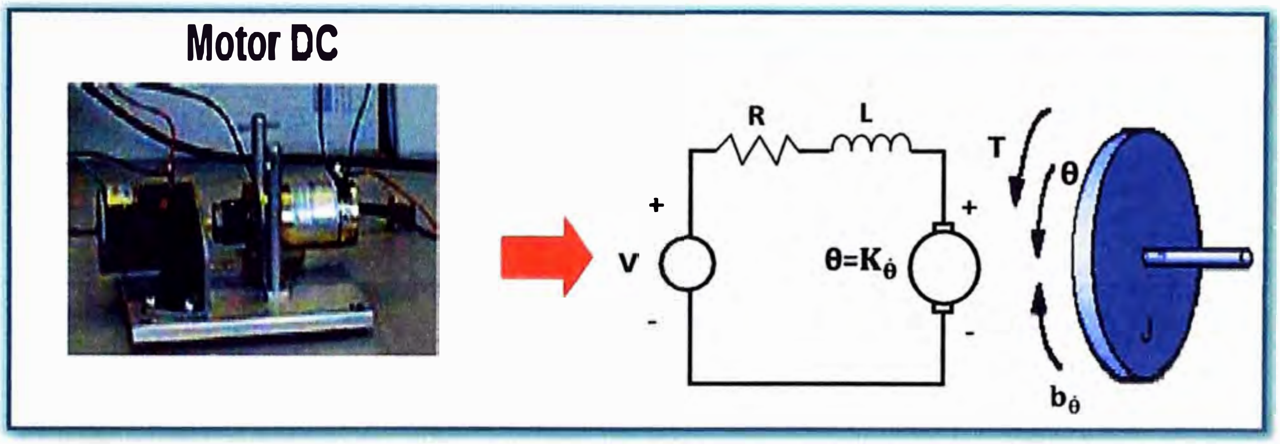


Fig.4.1 Motor DC-Esquema Eléctrico del Motor DC.

Ecuaciones del Sistema

$$J\ddot{\theta} + b\dot{\theta} = T = K_t i \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.2)$$

$$Li + Ri = V - K_e\dot{\theta} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4.3)$$

4.1.1 Parámetros del Motor DC

- Momento de inercia del rotor J=0.01 [kg .m. s^2/rad].
- Coeficiente de amortiguamiento del sistema mecánico b=0.1 [N. m. s].
- Constante de fuerza electromotriz =constante de armadura Ke=Kt=0.01 [Nm/Amp].
- Resistencia eléctrica R=1Ω
- Inductancia eléctrica L = 0.5 µH.

Para obtener un buen modelo se aplica el método de **Prueba - Error,** empezando por definir un primer modelo a un voltaje de alimentación del motor DC. Este modelo será

base de prueba para luego comparar la respuesta del sistema en estado estacionario con la respuesta obtenida en la simulación realizada en Simulink (MATLAB). Para la simulación se emplean escalones de 5, 10,15,..... 20, 25 y 30 Voltios aplicados al motor DC que nos permiten decidir si el modelo es aceptable o no.

Considerando los parámetros del motor DC se obtiene la función de transferencia para 5voltios según las muestras obtenidas (Anexo B).

$$G(s) = \frac{Kip}{(1+T_{p1}*s)(1+T_{p2}*s)} \quad \text{.....(4.4)}$$

Kip = 9.8045

Tp1= 0.017196

Tp1= 0.0024118

4.2 Adquisición de datos

Consiste en obtener datos del motor DC mediante el encoder, que envía pulsos digitales a la tarjeta de adquisición de datos, como puede apreciarse en la figura 4.3.

Fig. 4.3 Descripción del sistema de Transferencia de Datos

El motor DC que se utiliza para el proyecto es de Imán Permanente (PM) con las siguientes características:

Marca : Hitachi DC Motor

Volts : 30

Amp´s : 2

RPM : 2750

Ouput : 42 W.

Encoder : 400 P/R.

Considerando una revolución por minuto se tiene:

(2πradx 1/60s) es decir velocidad angular para una sola vuelta.

Para las 2750 revoluciones por minuto la velocidad angular seria:

(2πradx 1/60s) x2750=288 rad/s. que es la velocidad angular del motor DC trabajando a tensión máxima continua es decir a 30V.

Para una vuelta también se tiene:

(400pulsos/2πrad) pulsos digitales que salen del encoder de 400 ranuras.

Si el tiempo de muestreo es de 5 ms (Ts= 5ms) se estaría dando: (nπrad/s).

Otra manera de conseguir esta relación es:

((n pulsos)/(5. ⟦10⟧ ^(-3) s)) . (2πrad/(400 pulsos)) = (nπ rad/s) donde n cantidad de pulsos en un determinado tiempo de muestreo.

El tiempo de muestreo puede ser: Ts=5ms ó Ts= 10ms.pero es adecuado escoger un tiempo de muestreo de 5ms.

El tiempo de muestreo es importante porque en base a la información **prueba- error** se obtiene la mejor función de transferencia del motor DC, para luego diseñar un buen controlador.

La **prueba** se obtiene cuando el PIC18F2550 envía a la PC una ráfaga de datos propios del muestreo al escalón que se le aplica al motor DC y mediante MATLAB se visualiza esta información en la PC. Los datos recibidos en la PC son traducidos a velocidad angular de arranque del motor. El **error** se obtiene cuando el resultado práctico se compara con el resultado teórico que nos ofrece SIMULINK ante la misma entrada escalón a determinada función de transferencia del motor DC.

La mejor función de transferencia se da cuando velocidad angular del motor DC tanto teórico (SIMULINK) como práctico (Valores de la tarjeta de adquisición de datos) son casi iguales.

Para ello se adquiere datos del arranque del motor DC para diferentes niveles de tensión.

También se puede utilizar un CI L298 (Figura 4.4) que trabaja como Driver en la etapa de potencia.

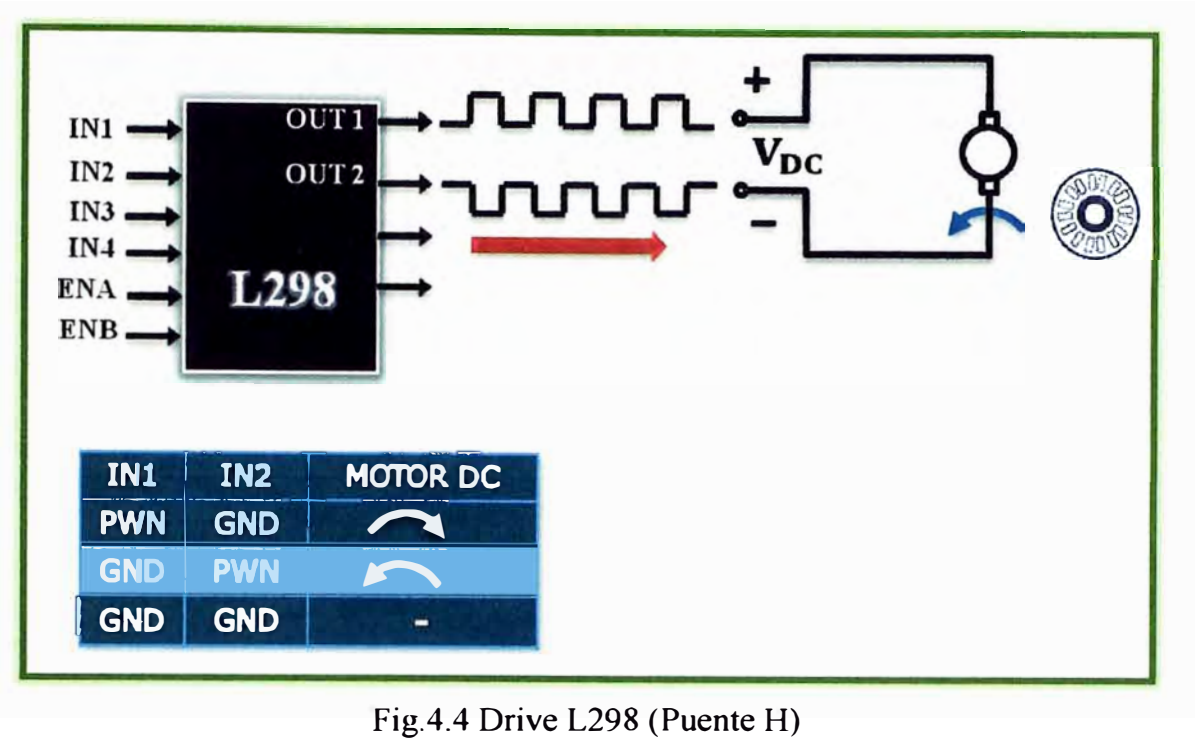


Fig.4.4 Drive L298 (Puente H)

Enlace de comunicación MATLAB -PIC

En MATLAB después de declarar los vectores:

data _in= eye (1,64,`uint8´) vector de datos de entrada (el que se recibe del PIC).

data _out= eye (1,64,`uint8´) vector de datos de salida (el que se envía ala PIC).

Luego se pone la condición:

If conectado ==1 que significa abrir los túneles tanto de transmisión como de recepción.

[my_out_pipe] = Se abre el túnel de transmisión

[my_in_pipe] = Se abre el túnel de recepción.

While(data _out(1)<=254) nos lleva a un proceso iterativo hasta que la condición de la desigualdad se cumpla.

Calllib() la PC envía al PIC el data_ out

MPUSB Write se escribe al PIC el data out

MPUSB Read se lee del PIC data in .

data_out(1)=data_out(1)+1.

data_in (1)=[1]

end

end

El programa de adquisición en C tiene en su desarrollo:

usb_get_packet (1,dato ,64) para el PIC dato para MATLAB data esta instrucción nos dice que se debe " coger" el paquete de datos que le envía la PC.

Luego se almacena el dato en el puerto b.

Portb= dato [0].

Se lee el puerto a y se almacena en el dato[1] mediante la instrucción .

dato[1]=porta .

usb_put_packet (1,dato ,64) el PIC responde poniendo un paquete de datos para enviarle a la PC de retorno.

data_in=eye (1,64, `uint8´) .

data_out=eye (1,64, `uint8´) dato del PIC que corresponde a un vector fila de 64datos.

MATLAB domina la PC y el PIC domina la tarjeta de adquisición de datos.

CONCLUSIONES Y RECOMENDACIONES

1. Se ha obtenido el modelo de la planta (Motor DC) en base a una respuesta escalón (Sep point). La Función de Transferencia equivalente, se logra realizando los procedimientos de prueba-error para los diferentes niveles de tensión continua aplicados al motor DC.
2. Obtenido el modelo del motor DC, se puede aplicar cualquier tipo de control y si el problema se analiza por Espacio de Estado permite controlar algunas variables como: posición, velocidad, ángulo y sentido de giro. Se puede obtener la función de transferencia de cualquier otro motor DC de mayor potencia, siguiendo los mismos procedimientos.
3. Se utiliza el PIC18F2550 porque cumple los requerimientos técnicos del proyecto como: enlace USB,y temporizadores, TMR1 para la detección de los pulsos del encoder y el TMR2 el control de velocidad del motor DC utilizando la configuración PWM.
4. El microcontrolador se utiliza para obtener datos es decir toma (envía) datos digitales normalizados del encoder a la PC, por lo tanto el PIC trabaja en sistema full-duplex.
   MATLAB determina la lógica de control del motor DC y se enlaza con comunicación de protocolo USB a la PC. El PIC recibe datos de la PC (MATLAB) y este lo convierte a un control PWM variando el duty cycle .
   La potencia del motor DC debe ser limitado o especifico ya que esto implica limitaciones para los drives de control (CI .L298).
5. La transferencia de datos PC-PIC se realiza a alta velocidad, esta es la razón por la cual encuentra restringida la distancia de transmisión mediante el cable conexión USB entre PC-PIC. Para un mejor entendimiento del proyecto y sus aplicaciones se recomienda incidir en software avanzado para Lenguaje de programación en C con aplicación específica al PIC18Fxxxx conocer a profundidad PROTEUS (ISIS, VMS, ARES) y tener un buen domino del software MATLAB.

ANEXO A
ANÁLISIS TEÓRICO

ANEXO A
ANÁLISIS TEÓRICO

Compilador C CCS y Simulador PROTEUS para Microcontroladores PIC

El estudio de los microcontroladores PIC no consiste solo en dominar su arquitectura interna o el código maquina sino también en conocer programas auxiliares que faciliten el diseño de los sistemas donde intervienen.

Entre los programas para el desarrollo de sistemas con PIC destacan por su potencia; el

PROTEUS VSM Labcenter Electronics y el compilador C de Custom Computer Services Incorporated(CCS).

El programa PROTEUS VMS es una herramienta que sirve para la verificación vía software y permite comprobar en cualquier diseño la eficacia del programa desarrollado. Su potencia de trabajo es magnífica.

PROTEUS es una aplicación CAD compuesta de tres módulos:

ISIS (Intelling Schematic Input System): Es el módulo de captura de esquemas.
VMS (Virtual System Modelling): Es el módulo de simulación.
ARES (Advanced Routing Modelling): Es el módulo para la realización de circuitos impresos.

También tenemos el compilador C de CCS ya que después de conocer y "dominar "el lenguaje ensamblador es muy útil aprender a programar con un lenguaje de alto nivel como el C. El compilador CCS C permite desarrollar programas en C enfocado a PIC con las ventajas que supone tener un lenguaje desarrollado específicamente para un microcontrolador concreto. Dado que tiene una relativa facilidad de uso un entorno de trabajo y la posibilidad de compilar en las tres familias de gama baja, media y alta le confieren una versatilidad y potencia muy elevada.[2]

➢ ISIS de PROTEUS VSM

El módulo ISIS es un programa que nos permite dibujar sobre un área de trabajo un circuito que posteriormente podremos simular. El entorno de diseño electrónico PROTEUS

VSM de LABCENTER ELECTRONICS ofrece la posibilidad de simular código microcontrolador de alto y bajo nivel.

Esto permite el diseño tanto a nivel de hardware como de software y realizar la simulación en un mismo y único entorno. Para ello, se suministran tres potentes subentornos como son: el ISIS para el diseño grafico, VSM (Virtual SystemModelling) para la simulación y el ARES para el diseño de placas.

Con las herramientas tradicionales de diseño, el desarrollo del software y la comprobación del sistema no puede realizarse hasta que se desarrolla un prototipo real, esto puede suponer semanas de retraso.

Además, si se localiza algún error en el diseño de hardware la totalidad del proceso debe repetirse. Usando Proteus VMS el desarrollo del software puede comenzar tanto como el esquemático es dibujado y la combinación de software y hardware puede ser testeada antes de montar el prototipo.

ISIS es un potente programa de diseño electrónico que permite realizar esquemas que pueden ser simulados en el entorno VSM. Posee una muy buena colección de librerías de modelos tanto para dibujar, simular y para la implementación de placas.

El programa ISIS posee un entorno de trabajo formado por distintas barras de herramientas y la ventana de trabajo. Para los microcontroladores la ventana de edición aporta mucha información, tal vez la más importante es que permite cargar en el microcontrolador el archivo de programa (HEX) generado en la compilación; también se puede modificar la frecuencia de reloj (por lo tanto no es necesario el uso de cristales externos en la simulación).

PROTEUS VMS permite visualizar elementos internos del PIC como son: La memoria de datos RAM, los registros especiales (FSR) y la pila (Stack). Además permite compilar programas fuente en código ensamblador directamente para ello se utiliza el comando SOURCE. Para ejecutar el programa desde ISIS se debe abrir la ventana de edición del microcontrolador y en el ítem PROGRAM FILE se puede indicar el fichero de código fuente utilizado. La simulación No es en tiempo real y dependerá de la carga de trabajo de la PC.

➢ **Compilador CCS C (Custom Computer Service Incorporated)**

El compilador de C de CCS ha sido desarrollado específicamente para PIC obteniendo la

máxima optimización del compilador con estos dispositivos. Dispone de una amplia librería de funciones predefinidas, comandos de preprocesado, además suministra los controladores (drivers) para diversos dispositivos como LCD, convertidores A/D, relojes en tiempo real.

Un compilador convierte el lenguaje de alto nivel a instrucciones en código maquina, un cross-compiler es un compilador que funciona en un procesador (normalmente en una PC) diferente al procesador objeto. El compilador CCS C es un cross-compiler, los programas son editados y compilados a instrucciones maquina en entorno de trabajo de la PC y el código maquina puede ser cargado de la PC al sistema PIC.

El CCS es C estándar y además de las directivas específicas estándar (#include), suministra unas directivas especificas para PIC (#device), además incluye funciones especificas (bit_ set ()). Se suministra un editor que permite controlar la sintáxis del programa.

➢ **Estructura de un programa**

Para escribir un programa en C con el CCS se debe tener en cuenta los siguientes elementos básicos:

Directivas de Preprocesado: Controlan la conversión del programa a código maquina por parte del compilador.

Programas o Funciones: Conjunto de instrucciones, puede haber uno o varios en cualquier caso hay uno definido como principal mediante la inclusión de la llamada main().

Instrucciones: Indican cómo debe comportarse el PIC en todo momento.

Comentarios: Permiten describir lo que significa cada línea del programa.

Directivas: Las directivas de preprocesado comienzan con el símbolo # y continúan con un comando específico, la sintaxis depende del comando se comenta algunos

. **#DEVICE chip**: Permite definir el PIC con el que se realizara la compilación.

#device PIC18F2550

. **#FUSES options** : Permite definir la palabra de configuración para programar un PIC . Por ejemplo, en el PIC 18F2550 las opciones son:

HSPLL,NOWDT,NOPROTECT,NOLVT,NODEBUG,USBDIV,PLL3,CPUDIV1,REGEN ,MCLR,NOPBADEN .[2]

. **#INCLUDE "filename"**: Permite incluir fichero en el programa

. **#USE DELAY (CLOCK=SPEED):** Permite definir la frecuencia del oscilador del PIC, el compilador lo utiliza para realizar cálculos de tiempo.

. **#ASM y #ENDASM:** Permiten utilizar código ensamblador en el programa en C. Se utiliza en el inicio y al final del bloque ensamblador.

Empleando las siguientes directivas se tiene:

#include <18F2550>

#device ADC=10

#fuses HSPLL, NOWDT, NOPROTECT, NOLVP, NODEBUG, USBDIV, PLL3, CPUDIV1, VREGEN, MCLR, NOPBADEN.

PLL3=Para cristal de 12MHZ

#use delay (clock=48000000)

➢ **Gestión de los Puertos**

Los microcontroladores PIC tienen terminales de entrada/ salida divididos en puertos que se encuentran nombrados alfabéticamente A,B,C.D,etc. Cada puerto puede tener hasta 8 terminales que de forma básica se comportan como una entrada /salida digital, los puertos se caracterizan por ser independientes, es decir se puede programar cada terminal del puerto para que se comporte como una entrada o salida digital.

La habilitación como entrada o salida se realiza a través del registro TRISx (TRISA, TRISB, TRISC, TRISD, TRISE).

Un valor **0** en estos registros indica que el terminal correspondiente del puerto es de salida, mientras que un valor **1** indica que será de entrada.

La gestión del bus de datos se realiza a través de los registros PORTx (PORTA, PORTB, PORTC, PORTD, PORTE).

En lenguaje C se gestionan los puertos de la siguiente forma:

Se declaran los registros TRISX y PORTx definiendo su posición en la memoria RAM como variable de C. La directiva utilizada es #BYTE:

#BYTEvariable =constante,

#BYTE TRISA=0x0F92 //Variable TRISA en 0F92

```
#BYTE TRISB=0x0F93          //Variable TRISB en 0F93
#BYTE PORTA=0x0F80          //Variable PORTA en 0F80
#BYTE PORTB=0x0F81          //Variable PORTB en 0F81
```

Los TIMER o temporizadores son módulos integrados en el PIC que permiten realizar cuentas tanto de eventos internos como externos. Cuando la cuenta es interna se habla de temporización y cuando la cuenta es externa se habla de contador.

Al producirse una interrupción, el PIC salta automáticamente a la dirección del vector interrupción de la memoria de programa y ejecuta la porción del programa correspondiente a la atención de la interrupción hasta encontrar la instrucción REFTFIE. Al encontrar dicha instrucción, abandona la interrupción y retorna a la posición de memoria del programa principal desde la que salto al producirse la interrupción. Las fuentes de interrupción dependen de la gama del PIC utilizado.

La interrupción exterior por RB0 es una interrupción básica y común en la mayoría de los PIC; permite generar una interrupción tras el cambio de nivel de alto a bajo o debajo a alto en la entrada RB0 . La directiva utilizada es #INT_EXT.

TIMER0

El bloque funcional TMER0/WATCHDOG es un contador (registro) de 8 bits incrementado por hardware y programable. La cuenta máxima es de 254 (el incremento es constante e independiente).

-Contador: cuenta eventos externos.

-Temporizador: cuenta los pulsos internos de reloj.

Se pueden insertar un prescaler, es decir un divisor de frecuencia programable que puede dividir por 2, 4, 16, 32, 64, 128, y 256.

El bloque del TIMER0 puede funcionar como WATCHDOG , permite que durante el funcionamiento normal del microcontrolador , un desbordamiento(o time off) del watchdog provoque un reset (watchdog timer reset). Para evitar el desbordamiento se debe cada cierto tiempo y antes que llegue al límite, ejecutar una instrucción CLRWDT que borra el watchdog y que hace comenzar un nuevo conteo desde cero.

La función para configurar el TIMER0 es:

Setup_timer_0 (modo) donde modo está definido en el fichero de cabecera.

Para activar el watchdog se utiliza los bits de configuración mediante la directiva #FUSES:

#fuses WDT Activado.

#fuses NOWDT Desactivado

El compilador C suministra una serie de funciones para leer o escribir en el TIMER/WDT.

Set_timer0 (valor): Para escribir un valor en el registro.

Valor: Entero de 8 bits.

Valor= get_timer0 (): Para leer el valor actual del registro.

El módulo **TIMER1** es otro registro /contador con las siguientes características:

-Trabaja con 16 bits, se puede leer o escribir.

-Tiene interrupción por desbordamiento y reset por disparo mediante el módulo CCP.

El **TIMER 2** (Figura A1) es un módulo temporizador con las siguientes características:

-Temporizador de 8bits (registro TMR2).

-Registro de periodo 8 bits (PR2).

-Ambos registros se pueden leer o escribir.

-Prescaler programable por programa (1:1,1:4,1:16).

-Postcaler programable por programa (1:1 a 1:16).

-Interrupción controlada por PR2.

El TIMER2 se puede emplear como base de tiempos para la modulación de ancho de pulso (PWM) mediante la utilización del módulo CCP.

El TIMER2 se puede leer o escribir y es borrado en el reset.

La configuración del módulo TMR2 en el compilador C se realiza con la función:

Set_timer _2 (modo, periodo, postscaler) donde :

Periodo: es un valor entero de 8 bits (0-254) para el registro PR2.

Postscaler : es el valor del postscaler (1 a 16).

La lectura escritura en el modulo TMR2 se realiza con la ayuda de las siguientes funciones:

Valor=get_timer 2();

Set_timer2 (valor) donde valor es un entero de 8 bits.[2]

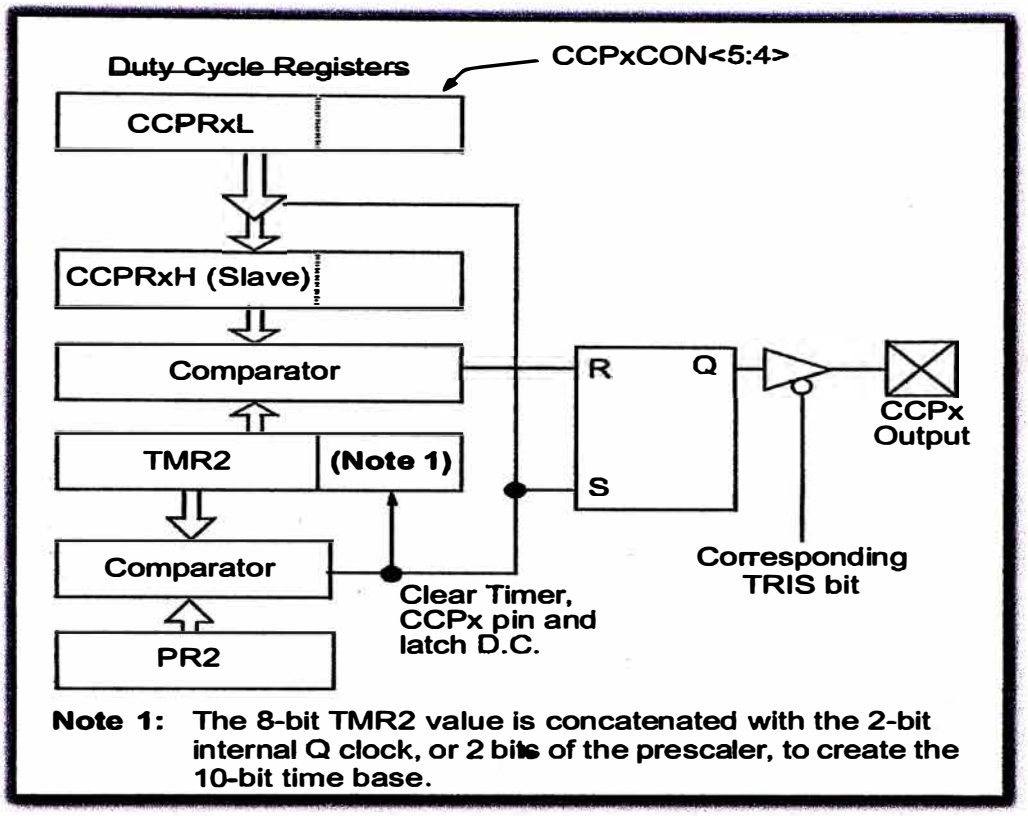


Fig. A.1 Diagrama de bloques simplificado (PWM).
(Hoja de datos de Microchip) [3]

➢ **Módulo CCP Comparador, Captura y PWM**

Los módulos CCP permiten realizar tres funciones básicas basadas en el manejo de los temporizadores (timer):

- Comparador: Compara el valor del temporizador con el valor de un registro y provoca una acción en el PIC.
- Captura: Obtiene el valor del temporizador en un momento dado fijado por la acción de un terminal del PIC.
- PWM: Genera una señal modulada en ancho de pulso.

Tras producirse un reset el módulo CCP se encuentra deshabilitado.

Cada módulo CCP posee un registro de 16 bits que puede utilizarse de tres siguientes formas:

- Registro de 16 bits para capturar el valor del temporizador al producirse un evento (Captura).
- Registro de 16 bits para comparar su valor con el valor del temporizador TMR1 pudiendo provocar un evento cuando se alcanza el valor contenido en este registro (Comparador).

- Registro de 10bits para el ciclo de trabajo de una señal PWM (PWM).

Modo PWM: Son los dos bits menos significativos de los 10bits utilizados para el Duty Cycle del PWM. Los ocho bits de mayor peso del Duty Cycle se encuentran en el registro CCPRxL (Byte bajo del registro de 16bits del CCP).

El modo PWM (Pulse Width Modulation) o Modulacion de Ancho de Pulso permite obtener en los pines CCPx una señal periódica en la que se puede modificar su ciclo de trabajo (Duty Cicle) es decir puede variarse el tiempo en el cual la señal esta a nivel alto (Ton) frente al tiempo que esta a nivel bajo (Toff).

La tensión media aplicada a la carga es proporcional al Ton, controlando por ejemplo la velocidad de motores, luminosidad de lámparas.

La resolución de salida es de hasta 10 bits. Para que este módulo funcione correctamente el pin debe estar configurado como salida; mediante la configuración del TRIS correspondiente. El periodo de la señal PWM se obtiene de configurar el TIMER2 y el contenido del registro PR2. Este registro de 8bits contiene, 8bits más significativos de una cuenta de 10 bits.

Cuando el valor del TIMER2 se iguala al valor PR2 puede ocurrir lo siguiente:

- TIMER2 se borra.
- El pin CCPx se pone a 1 (excepción: si el Duty Cicle es 0%).
- El valor de CCPRxL se carga en el CCPRxH,el cuál es el que se compara con el TIMER2 para fijar el Duty Cicle.

El compilador C suministra una serie de funciones para el manejo del módulo CCP.

Configuración del módulo CCPx:

Setup_ccpx(modo).

Definicion del ciclo de trabajo para PWM.

Set_pwm_duty (valor).

Valor: dato de 8 o16 bits que determina el ciclo de trabajo.

➢ Transmisión Serie

Los PIC utilizan entre otros dos modos de transmisión en serie:

- El puerto serie síncrono (SSP).[2]

- La interfaz de comunicación serie (SCI) o receptor transmisor serie síncrono asíncrono universal (USART).

El SSP se suele utilizar en la comunicación con otros microcontroladores o con periféricos. Las dos interfaces de trabajo son:

- Interfaz serie de periféricos (SPI): desarrollada por Motorola para la comunicación entre microcontroladores de la misma o diferente familia en modo maestro esclavo full-duplex.
- Interfaz Inter-Circuitos (I^2C): Interfaz desarrollada por Philips con una gran capacidad para comunicar microcontroladores y periféricos half-duplex.

➢ **PIC18 (Gama Alta)**

Microchip ha lanzado varias gamas de PIC con elevadas prestaciones los PIC18, los PIC24 y los dsPIC. Con la gama alta PIC18 Microchip mantiene la arquitectura básica que tan buenos resultados ha obtenido con la gama baja y media.

Los PIC18 tienen una arquitectura RISC avanzada Harvard con 16bits de bus de programa y 8bits de bus de datos.

La memoria de programa aumenta hasta 1Mbyte (en realidad se manejan hasta 64KBytes). La memoria de datos RAM puede llegar hasta 16x256 (4KBytes) y hasta 1KBytes de EEPROM.

La pila aumenta hasta 31niveles, incluyen tres punteros FSR que permiten direccionar la memoria de datos de forma indirecta y sin bancos. El juego de instrucciones aumenta hasta las 75 instrucciones. Introduce un multiplicador hardware8x8.

La frecuencia máxima de reloj es de 40MHZ incluye periféricos de comunicación avanzados (CAN y USB). Los PIC18 son compatibles con los PIC16CXX y PIC17CXX, además ha desarrollado un compilador C especifico para esta gama alta, elC18. El PIC que se utiliza es el PIC18F2550 porque cumple con las condiciones de diseño del proyecto.

Se escogió el PIC18F2550 por las siguientes razones:

- Tiene el flash de 32Kbytes para la memoria del programa válido para los dispositivos de PIC18FX550, en comparación con 24Kbytes para los dispositivos de PIC18FX455.
- Tiene 3puertos bidireccionales I/O y 1 puerto de entrada suficiente para los fines que se pretende para los dispositivos de 28pines, en comparación con 5 puertos bidireccionales en dispositivos de 40/44pines.[2]

- Tiene la capacidad de generar PWM1 y PWM2 (Modulación de ancho de pulso) gracias a que cuenta con 2 módulos CCP(Captura –Comparación -PWM) los cuales son útiles para regular la velocidad de un motor DC.
- Menor Hardware porque es un dispositivo de 28pines en comparación con 40/44pines dentro de la familia PIC18F.

➢ **Organización de la Memoria**

El PIC18 (FiguraA.2) dispone de las siguientes memorias:

- **Memoria de programa**: Memoria FLASH interna de 32.768bytes.

Almacena instrucciones y constantes /datos. Puede ser escrita /leída mediante un programador externo o durante la ejecución del programa mediante unos punteros.

- **Memoria RAM de datos: M**emoria SRAM interna de 1.536bytes en la que están incluidos los registros de función especial
- Almacena datos de forma temporal durante la ejecución del programa.
- Puede ser escrita/leída en tiempo de ejecución mediante diversas instrucciones.
- **Memoria EEPROM de datos:** Memoria no volátil de 256 bytes.
- Almacena datos que se deben conservar aun en ausencia de tensión de alimentación.
- Puede ser escrita/leída en tiempo de ejecución a través de registros.
- **Pila:** Bloque de 31 palabras de 21 bits.
- Almacena la dirección de la instrucción que debe ser ejecutada después de una interrupción o subrutina.
- **Memoria de configuración:** Memoria en la que se incluyen los bits de configuración (12 bytes de memoria flash) y los registros de identificación (2 bytes de memoria de solo lectura).

➢ **Arquitectura HARVARD**

El PIC18 dispone de buses diferentes para el acceso a la memoria de programa y a la memoria de datos (arquitectura Harvard):

- Bus de la memoria de programa:
- 21 líneas de dirección.
- 16/8 líneas de datos (16 líneas para instrucciones/8 líneas para datos).
- Bus de la memoria de datos.
- 12 líneas de dirección.
- 8 líneas e datos.

Esto permite acceder simultáneamente a la memoria de programa y a la memoria de datos. Es decir, se puede ejecutar una instrucción (lo que por lo general requiere acceso a la memoria de datos) mientras se lee de la memoria de programa la siguiente instrucción (proceso pipeline).

➢ **Memoria de Programa**

EL PIC18 dispone de una memoria de programa de 32.768 bytes. Las instrucciones ocupan 2 bytes (excepto las instrucciones CALL, MOVFF, GOTO y LSFR que ocupan 4).

Por lo tanto, la memoria de programa puede almacenar hasta 16.384 instrucciones.

Primero se almacena la parte baja de la instrucción y luego la parte alta (para las instrucciones de 4 bytes primero los bytes menos significativos y luego los más significativos).(Figura A.3).

Las instrucciones siempre empiezan en direcciones pares.

Direcciones especiales de la memoria de programa:

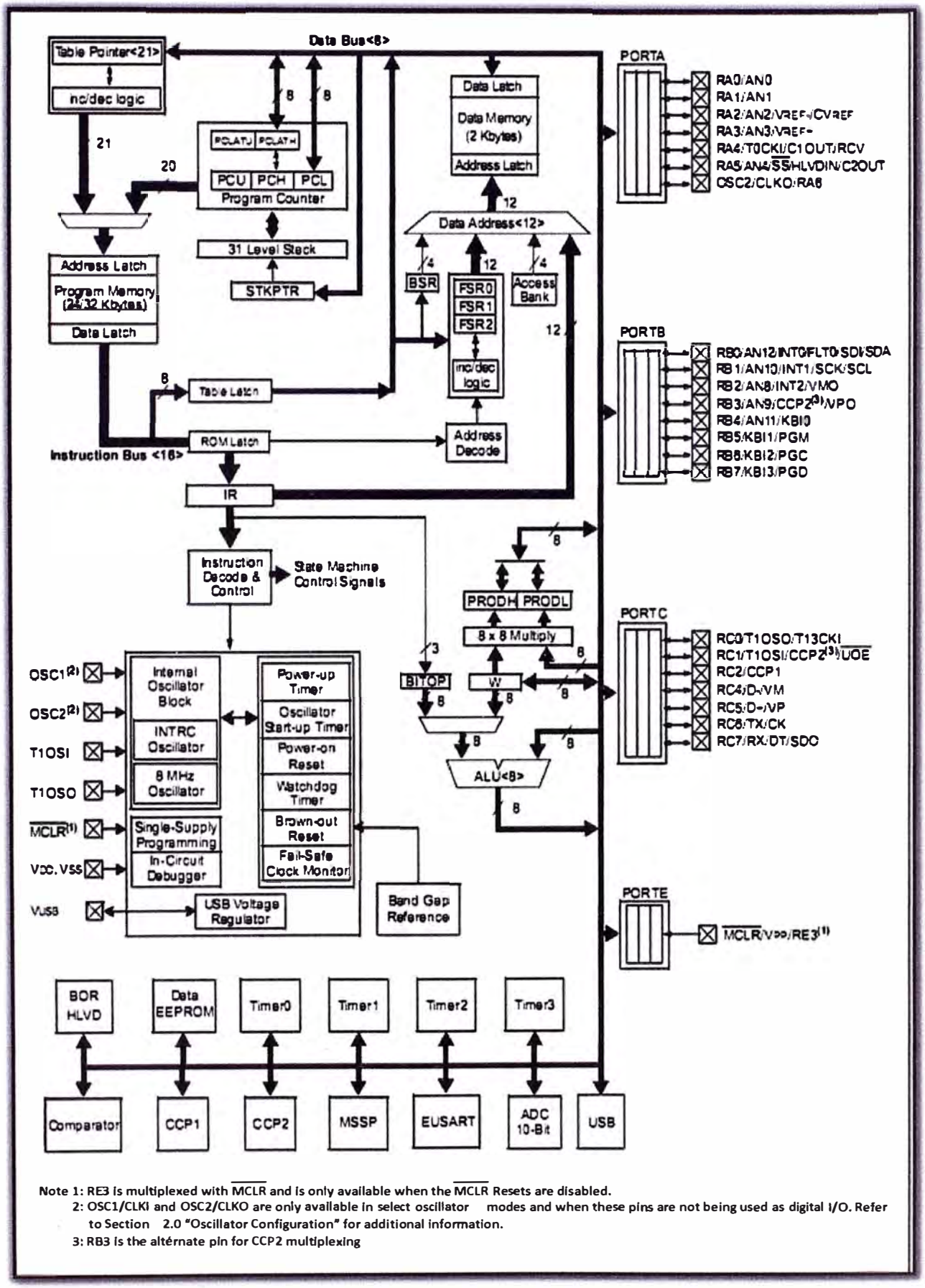
- Vectorización del reset.
- Vectorización de las interrupciones de alta prioridad.
- Vectorización de las interrupciones de baja prioridad.

La memoria de programa puede ser leída, borrada y escrita durante la ejecución del programa. La operación que se utiliza normalmente en tiempo de ejecución es la de lectura de tablas o datos almacenados en memoria de programa.

➢ **Contador de Programa**

El PC (contador de programa) tiene 21 bits (PCU, PCH y PCL). El bit menos significativo del PC apunta a BYTEs, no a WORDs, por lo que es ``0''. El PC incrementa de 2 en 2.

- **PCU:** Parte superior del PC registro no directamente accesible; las operaciones de lectura/escritura sobre este registro se hacen a través del registro PCLATU.
- **PCH:** Parte alta del PC registro no directamente accesible; las operaciones de lectura/escritura sobre este registro se hacen a través del registro PCLATH.
- **PCL:** Parte baja del PC, registro directamente accesible. Una operación de lectura sobre PC provoca que los valores de PCU y PCH pasen a los registros PCLATU y PCLATH, respectivamente.



Note 1: RE3 is multiplexed with MCLR and is only available when the MCLR Resets are disabled.
2: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to Section 2.0 "Oscillator Configuration" for additional information.
3: RB3 is the altérnate pin for CCP2 multiplexing

Fig. A.2 Diagrama de Bloques del PIC18F2550
(Hoja de datos de Microchip) [3]

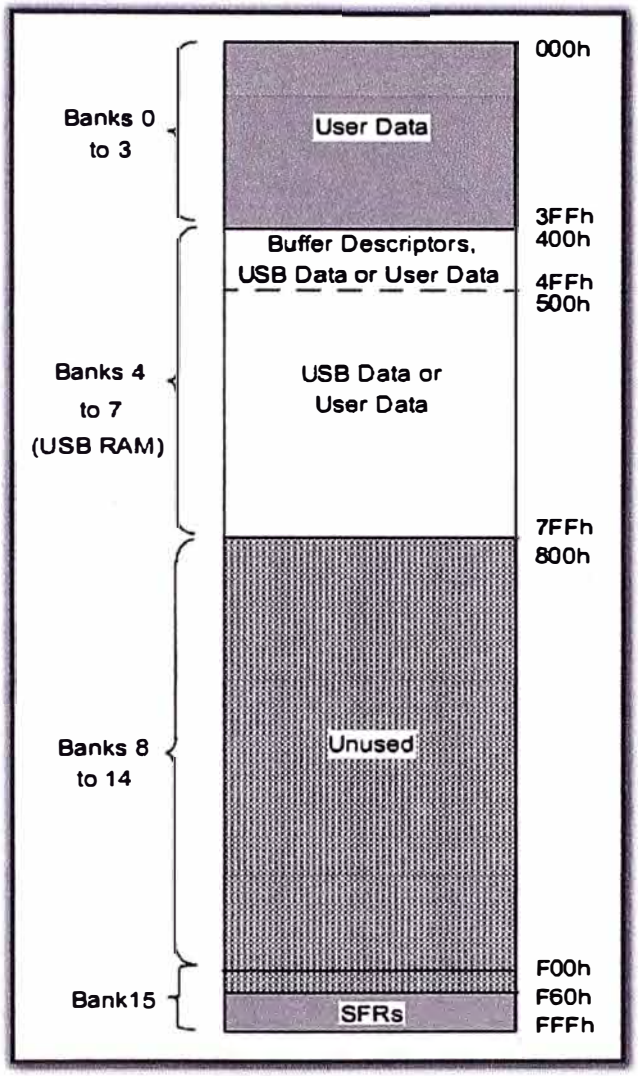


Fig. A.3 Implementación de la Memoria de Datos.
(Hoja de datos de Microchip) [3]

Memoria de Configuración

En esta memoria de configuración incluyen:

- **Bits de configuración:** contenidos en 12 bytes de memoria flash permiten la configuración de algunas operaciones del PIC como:
- Opciones de oscilador.
- Opciones de reset.
- Opciones de watchdog.
- Opciones de la circuitería de depuración y programación.
- Opciones de protección contra escritura de la memoria de programa y de la memoria

EEPROM de datos.

Estos bits se configuran generalmente durante la programación C, aunque también pueden ser leídos y modificados durante la ejecución del programa.

- **Registros de identificación:** Se trata de dos registros situados en las direcciones que contienen información del modelo y revisión del dispositivo. Son registros de solo lectura y no pueden ser modificados por el usuario.

➢ **Pila**

La pila es un bloque de memoria RAM independiente de 31 palabras de 21 bits y un puntero de 5 bits, que sirve para almacenar temporalmente el valor del PC cuando se produce una llamada a una subrutina o interrupción. El ``top of stack''' es accesible, se puede leer y escribir (será conveniente quitar previamente las interrupciones).

El puntero de pila (contenido en el registro STKPTR) es un contador de 5 bits que indica la posición actual final de la pila . El contenido del final de pila es accesible mediante los registros TOSU, TOSH, TOSL. Cuando se procesa una interrupción o se ejecutan las instrucciones CALL o RCALL (el PC está apuntando a la siguiente instrucción) se incrementa el STKPTR y se almacena el valor del PC en el final de la pila. Cuando se ejecutan las instrucciones RETURN, RETLW RETFIE se copia en el PC el valor almacenado en la cima de pila y se decrementa el STKPTR. [2]

➢ **Memoria de Datos**

Los PIC18 tienen hasta un total de 4 KBytes agrupados en 16 bancos con 256 bytes cada uno. Como en el resto de las gamas, existen los registros de propósito general GPR y los SFRs éstos últimos se sitúan en la zona más alta.

El PIC18 dispone de una memoria RAM de datos 1.536 bytes (6 bancos de 256 bytes). Además dispone de 126 bytes dedicados a los registros de función especial (SFRs) registro de funciones especiales.

Para acceder a un byte de la memoria RAM de datos primero se debe seleccionar el banco al que pertenece el byte mediante el registro de selección de banco (BSR) y a continuación, direccionar el byte dentro del banco.

Además existe una modalidad de acceso rápido a las 126 posiciones de la parte baja del banco 0 y a los 126 bytes de SFR (banco de acceso rápido). La memoria RAM de datos se compone de registros de propósito general (GPRs) y de registros de función especial (SFRs). Los SFRs son los registros mediante los cuales se les puede monitorizar / controlar el funcionamiento de la CPU y de las unidades funcionales del PIC, cumplen una función importante. (Figura A.4).

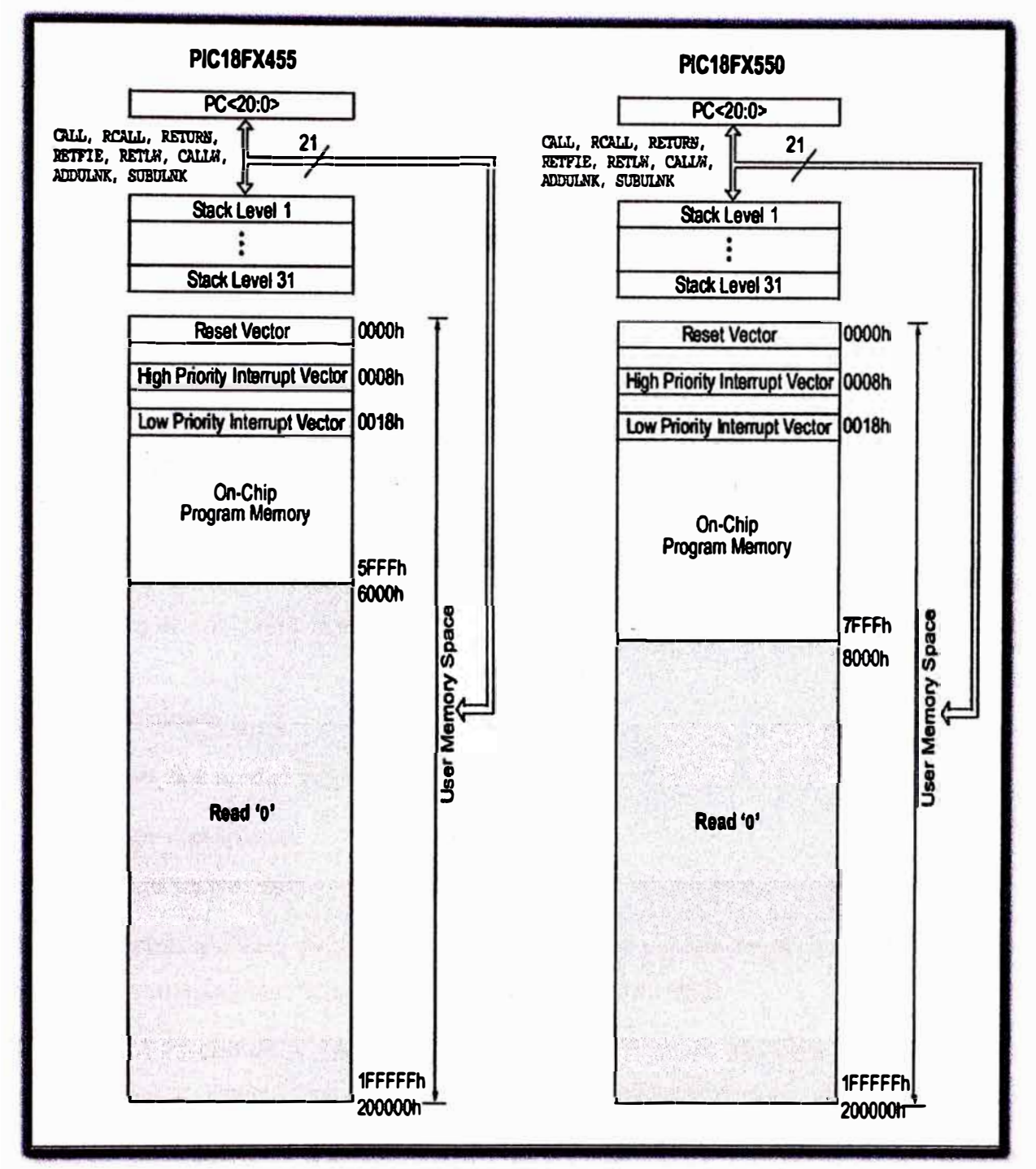


Fig. A.4 Mapa de Programación de Memoria y Pila del PIC18F2550

(Hoja de datos de Microchip) [3]

➢ Memoria EEPROM

Esta memoria permite hasta 1.000.000 de ciclos de borrado /escritura. Se puede leer/escribir de forma individual en cada una de las 256 posiciones de memoria. Cuando se realiza una operación de escritura, la circuitería interna del PIC se encarga de borrar previamente la posición en la que se desea escribir. La duración de un ciclo completo de borrado /escritura de un byte en la memoria EEPROM suele ser de unos 4ms.

➢ **Modo de Direccionamiento**

El modo de direccionamiento es la forma en la que se obtienen los datos que van a ser utilizados en la instrucción. Existen 4 modos de direccionamiento: Inherente, Literal, Directo e Indirecto.

- **Modo de direccionamiento inherente**: En este modo, o bien la instrucción no tiene operando o bien el operando viene especificado en el propio código de operación de la instrucción.
- **Modo de direccionamiento literal**: En este modo, el valor del operando viene indicado de forma explícita en la instrucción.
- **Modo de direccionamiento directo**: En este modo, la dirección en la que se encuentra el valor del operando viene indicada de forma explícita en la instrucción.
- **Modo de direccionamiento indirectamente:** en este modo, la dirección de memoria en la que se encuentra el dato viene especificado en uno de los registros FSR0, FRS1y FRS2.

➢ **Interrupciones**

Se dispone de dos niveles de prioridad:

- Nivel alto vectorizado.
- Nivel bajo vectorizado.

Todas las interrupciones pueden ser programadas con cualquiera de las dos prioridades, salvo la interrupción externa 0(que siempre tiene alta prioridad).

GIE/GIEH&PEIE/GIEL Controlan los respectivos permisos globales. Cuando se sirve una interrupción, automáticamente se quita su correspondiente permiso global.

Todas las interrupciones disponen de 3 bits de configuración (excepto la interrupción externa 0 que tiene dos):

- **Bit de habilitación de interrupción**: Permite habilitar a nivel individual la interrupción.
- **Fag de interrupción**: Se pone a "1" cuando se produce la condición de interrupción independientemente de si la interrupción está habilitada o no. Este flag debe ponerse a "0"por software cuando se procesa la interrupción.
- **Bit de prioridad de interrupción**: Establece si la interrupción es de alta o de baja prioridad (este bit no está disponible para la interrupción externa 0).

Se distinguen dos grupos de interrupciones:

- Grupo general de interrupciones :

Interrupción de temporizador 0.

Interrupción por cambio en PORTB.

Interrupción externa 0.

Interrupción externa 1.

Interrupción externa 2.

- Grupo de interrupciones de periféricos :

Interrupción del SSP.

Interrupción del A/D.

Interrupción de recepción de la EUSART.

Interrupción de transmisión de la EUSART.

Interrupción del MMSP.

Interrupción del CCP1

Interrupción del temporizador 1.

Interrupción del temporizador 3.

Interrupción del fallo del oscilador.

Interrupción del comparador.

Interrupción del CCP2.

Interrupción de escritura en flash/EEPROM.

Interrupción de colisión de bus (MSSP).

Interrupción de anomalías en V_{DD}

Interrupción del temporizador 2.

En el compilador C se modifica la directiva #INT_XXXX de tal forma que se pueden incluir las palabras clave HIGH y FAST.

Una prioridad HIGH puede interrumpir a otra interrupción, mientras que una prioridad FAST se realiza para salvar o restaurar registros.

Así en los PIC18 se pueden dar las siguientes interrupciones en C:

- #INT_XXXX: Prioridad normal (baja) de interrupción. El compilador guarda y restaura los registros clave. Esta interrupción no interrumpe a otras en progreso, porque es de baja prioridad.
- #INT XXXXFAST: Interrupción de alta prioridad. El compilador NO guarda y restaura los registros clave. Esta interrupción puede interrumpir a otras en progreso. Solo se permite una en el programa.
- #INT_XXXXHIGH: Interrupción de alta prioridad. El compilador guarda y restaura los registros clave. Esta interrupción puede interrumpir a otras en progreso.

WREG es un registro direccionable por lo que se puede utilizar de forma explícita.

➢ **Oscilador**

LP: Cristal Baja- Potencia (máx. 200KHZ).

XT: Cristal Resonador (máx. 4MHZ).

HS: Cristal Resonador Alta -Velocidad (máx. 40MHZ).

HSPLL: Cristal Resonador Alta- Velocidad con habilitación de PLL (máx.10MHZ).

RC: Externa R-C con FOSC/4(max. 4MHZ).

RCIO: Externa R-C con I/O.

INTIO1: Oscilador Interno FOSC/4 con I/O.

El PIC18 dispone de una serie de unidades funcionales que le permiten:

- Realizar tareas especificas especializadas (conversión A/D, transmisión /recepción de datos, generación de señales digitales con temporizadores programables).
- Optimizar el rendimiento del PIC, ya que estas unidades trabajan en paralelo a la CPU permitiendo que ésta se centre en otras tareas como el procesado de datos, cálculos movimientos de datos.

Las unidades funcionales más importantes son:

Puertos de Entrada Salida El PIC18F2550 dispone de puertos Entrada/Salida tales como se aprecia en la Figura (A.5), Figura (A.6) y Figura (A.7).

Pin Name	Pin Number PDIP, SOIC	Pin Type	Buffer Type	Description
				PORTA is a bidirectional I/O port.
RA0/AN0	2			
RA0		I/O	TTL	Digital I/O.
AN0		I	Analog	Analog input 0.
RA1/AN1	3			
RA1		I/O	TTL	Digital I/O.
AN1		I	Analog	Analog input 1.
RA2/AN2/VREF-/CVREF	4			
RA2		I/O	TTL	Digital I/O.
AN2		I	Analog	Analog input 2.
VREF-		I	Analog	A/D reference voltage (low) input.
CVREF		O	Analog	Analog comparator reference output.
RA3/AN3/VREF+	5			
RA3		I/O	TTL	Digital I/O.
AN3		I	Analog	Analog input 3.
VREF+		I	Analog	A/D reference voltage (high) input.
RA4/T0CKI/C1OUT/RCV	6			
RA4		I/O	ST	Digital I/O.
T0CKI		I	ST	Timer0 external clock input.
C1OUT		O	—	Comparator 1 output.
RCV		I	TTL	External USB transceiver RCV input.
RA5/AN4/$\overline{SS}$/ HLVDIN/C2OUT	7			
RA5		I/O	TTL	Digital I/O.
AN4		I	Analog	Analog input 4.
$\overline{SS}$		I	TTL	SPI slave select input.
HLVDIN		I	Analog	High/Low-Voltage Detect input.
C2OUT		O	—	Comparator 2 output.
RA6	—	—	—	See the OSC2/CLKO/RA6 pin.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; I = Input; O = Output; P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX Configuration bit is cleared.

2: Default assignment for CCP2 when CCP2MX Configuration bit is set.

Fig. A.5 Descripción de Pines Entrada-Salida
(Hoja de datos de Microchip)[3]

Puerto de E/S	**Unidad de Comparación /Captura/PWM mejorada (ECCP)**
Temporizador 0	**Canal de comunicación serie EUSART**
Temporizador 1	**Canal de comunicación serie MSSP**
Temporizador 2	**Modulo analógico de comparación**
Temporizador 3	**Canal de transmisión de datos en paralelo (SPP)**
Convertidor A/D	**Acceso a memoria externa (EMA)**

Pin Name	Pin Number PDIP, SOIC	Pin Type	Buffer Type	Description
				PORTC is a bidirectional I/O port.
RC0/T1OSO/T13CKI	11			
RC0		I/O	ST	Digital I/O.
T1OSO		O	—	Timer1 oscillator output.
T13CKI		I	ST	Timer1/Timer3 external clock input.
RC1/T1OSI/CCP2/$\overline{UOE}$	12			
RC1		I/O	ST	Digital I/O.
T1OSI		I	CMOS	Timer1 oscillator input.
CCP2(2)		I/O	ST	Capture 2 input/Compare 2 output/PWM 2 output.
$\overline{UOE}$		—	—	External USB transceiver $\overline{OE}$ output.
RC2/CCP1	13			
RC2		I/O	ST	Digital I/O.
CCP1		I/O	ST	Capture 1 input/Compare 1 output/PWM 1 output.
RC4/D-/VM	15			
RC4		I	TTL	Digital input.
D-		I/O	—	USB differential minus line (input/output).
VM		I	TTL	External USB transceiver VM input.
RC5/D+/VP	16			
RC5		I	TTL	Digital input.
D+		I/O	—	USB differential plus line (input/output).
VP		O	TTL	External USB transceiver VP input.
RC6/TX/CK	17			
RC6		I/O	ST	Digital I/O.
TX		O	—	EUSART asynchronous transmit.
CK		I/O	ST	EUSART synchronous clock (see RX/DT).
RC7/RX/DT/SDO	18			
RC7		I/O	ST	Digital I/O.
RX		I	ST	EUSART asynchronous receive.
DT		I/O	ST	EUSART synchronous data (see TX/CK).
SDO		O	—	SPI data out.
RE3	—	—	—	See $\overline{MCLR}$/VPP/RE3 pin.
VUSB	14	O	—	Internal USB 3.3V voltage regulator.
VSS	8, 19	P	—	Ground reference for logic and I/O pins.
VDD	20	P	—	Positive supply for logic and I/O pins.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; I = Input; O = Output; P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX Configuration bit is cleared.

2: Default assignment for CCP2 when CCP2MX Configuration bit is set.

Fig. A.6 Descripción de Pines Entrada-Salida (continuación)

(Hoja de datos de Microchip)[3]

PUERTO	LINEAS DE ENTRADA/SALIDA
PORT A	8 LINEAS DE ENTRADA/SALIDA
PORT B	8 LINEAS DE ENTRADA/SALIDA
PORT C	6 LINEAS DE ENTRADA /SALIDA+2 LINEAS DE ENTRADA
PORT D	8 LINEAS DE ENTRADA/SALIDA
PORT E	3 LINEAS DE ENTRADA/SALIDA+1 LINEA DE ENTRADA

Pin Name	Pin Number: PDIP, SOIC	Pin Type	Buffer Type	Description
				PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0/AN12/INT0/FLT0/ SDI/SDA	21			
RB0		I/O	TTL	Digital I/O.
AN12		I	Analog	Analog input 12.
INT0		I	ST	External interrupt 0.
FLT0		I	ST	PWM Fault input (CCP1 module).
SDI		I	ST	SPI data in.
SDA		I/O	ST	I^2C™ data I/O.
RB1/AN10/INT1/SCK/ SCL	22			
RB1		I/O	TTL	Digital I/O.
AN10		I	Analog	Analog input 10.
INT1		I	ST	External interrupt 1.
SCK		I/O	ST	Synchronous serial clock input/output for SPI mode.
SCL		I/O	ST	Synchronous serial clock input/output for I^2C mode.
RB2/AN8/INT2/VMO	23			
RB2		I/O	TTL	Digital I/O.
AN8		I	Analog	Analog input 8.
INT2		I	ST	External interrupt 2.
VMO		O	—	External USB transceiver VMO output.
RB3/AN9/CCP2/VPO	24			
RB3		I/O	TTL	Digital I/O.
AN9		I	Analog	Analog input 9.
CCP2(1)		I/O	ST	Capture 2 input/Compare 2 output/PWM 2 output.
VPO		O	—	External USB transceiver VPO output.
RB4/AN11/KBI0	25			
RB4		I/O	TTL	Digital I/O.
AN11		I	Analog	Analog input 11.
KBI0		I	TTL	Interrupt-on-change pin.
RB5/KBI1/PGM	26			
RB5		I/O	TTL	Digital I/O.
KBI1		I	TTL	Interrupt-on-change pin.
PGM		I/O	ST	Low-Voltage ICSP™ Programming enable pin.
RB6/KBI2/PGC	27			
RB6		I/O	TTL	Digital I/O.
KBI2		I	TTL	Interrupt-on-change pin.
PGC		I/O	ST	In-Circuit Debugger and ICSP programming clock pin.
RB7/KBI3/PGD	28			
RB7		I/O	TTL	Digital I/O.
KBI3		I	TTL	Interrupt-on-change pin.
PGD		I/O	ST	In-Circuit Debugger and ICSP programming data pin.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; I = Input; O = Output; P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX Configuration bit is cleared.
2: Default assignment for CCP2 when CCP2MX Configuration bit is set.

Fig. A.7 Descripción de Pines Entrada-Salida (continuación)
(Hoja de datos de Microchip)[3]

Todas las líneas digitales de E/S disponen como mínimo de una función alternativa asociada a alguna circuitería especifica del PIC.

Cuando una línea trabaja en el modo alternativo no puede ser utilizada como línea digital de E/S estándar.

Cada puerto de E/S tiene asociado 3 registros:

- Registro TRIS: mediante este registro se configuran las líneas de E/S del puerto ENTRADA (bit correspondiente a"1"), como SALIDA (bit correspondiente a "0").
- Registro PORT: mediante este registro se puede leer el nivel del pin de E/S y se puede establecer el valor del latch de salida.
- Registro LAT: mediante este registro se pude leer o establecer el valor del latch de salida.

➢ **Temporizadores**

- Configurable como temporizador /contador de 8bits/16bits.
- Pre-escaler de 8bits programable.
- Interrupción de desbordamiento.

. Temporizador1:

- Configurable como temporizador /contador de 16bits.
- Dispone de un oscilador propio que puede funcionar como :
- Señal de reloj del temporizador 1.
- Señal de reloj del PIC en modo de bajo consumo.
- Pre-escaler de 3bits programable.
- Interrupción por desbordamiento

. Temporizador2:

- Temporizador de 8bits como se aprecia en la figura A.8 (registro TMR2).
- Registro de periodo PR2.
- Pre-escaler de 2bits programable (1:1,1:4,1:16).
- Post-escaler de 4bits (1:1…… 1:16).
- Interrupción por igualdad entre TMR2 y PR2.
- Se puede utilizar junto con los módulos CCP y ECCP.
- Se puede utilizar como señal de reloj del modulo MSSP en modo SPI.

➢ **Convertidor Análogo Digital**

- 10bits de resolución.
- 13 canales multiplexados.
- Señal de reloj configurable.

- Tiempo de adquisición programable.
- Posibilidad de establecer el rango de tensiones de conversión mediante tensiones de referencia externas.[2]

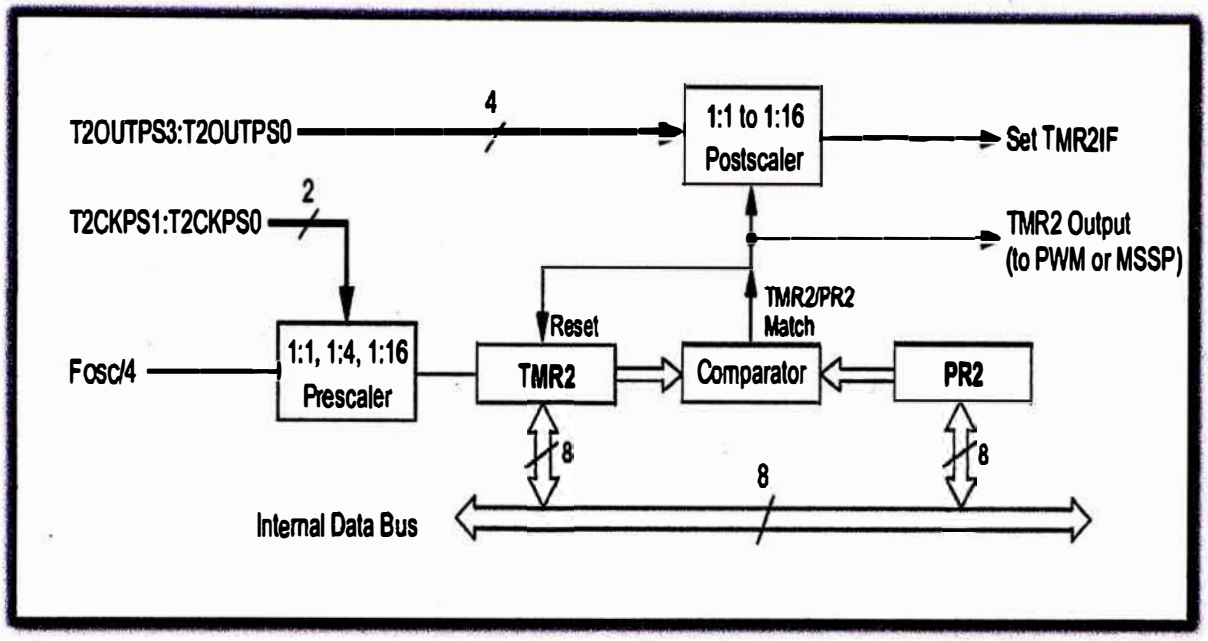


Fig. A.8 Diagrama de Bloques del TIMER 2
(Hoja de datos de Microchip)[3]

➢ **Canal de Comunicación Serie (EUSART)**

Características fundamentales:

- Modos de trabajo:
  - Modo asíncrono de 8bits.
  - Modo asíncrono de 9bits.
  - Modo síncrono Maestro.
  - Modo síncrono Esclavo.
- Auto –activación por detección de dato recibido.
- Detección automática de velocidad de comunicación (baudrate).
- Transmisión y detección del carácter de BREAK (bus LIN).

➢ **Módulo Master SSP (MSSP)**

El módulo MSSP es un interfaz serie capaz de comunicarse con periféricos u otro microcontrolador . Puede operar de dos modos :

- Serial Peripheral Interface (SPI).
- Inter-Integrated Circuit (I2C).

La interfaz I2C admite los siguientes modos:

- Master mode.
- Multi-master mode .
- Slave mode.

➢ **Módulo de Comparación/Captura/PWM (CCP)**

Dispone de tres modos de funcionamiento:

- **Modo de captura:** Se utiliza para medir eventos externos como la duración de pulsos digitales.
- **Modo de comparación:** Se utiliza para generar señales digitales con temporizaciones programables. Este tipo de señales son muy útiles para el control de etapas de potencia.
- **Modo PWM:** Se utiliza para generar señales de modulación de ancho de pulso (PWM).
- **Modo PWM mejorado:** Se utiliza para generar señales PWM complementarias para el control de semipuentes de transistores .También existe un módulo de comparación /captura /PWM mejorado (ECCP).

➢ **Módulo comparador**

El módulo de comparadores analógicos contiene dos comparadores que pueden ser configurados de distintas formas. Las entradas pueden seleccionarse entre entradas analógicas. Las salidas digitales (normal o invertida) son accesibles exteriormente y pueden ser leídas a través de un registro de control.

➢ **Módulo de referencia**

El módulo de referencia está formado por una red de resistencias y permite seleccionar una tensión de referencia.

Módulo detector de (Alto /Bajo) Voltaje

Este módulo programable permite al usuario, definir un punto umbral de tensión y la dirección de cambio. Si el dispositivo experimenta un cambio en la tensión y en la dirección indicada sobre el punto umbral se produce una interrupción.

➢ **Universal Serial Bus**

El Bus Serie Universal (USB) fue creado por los años 90 con la idea de mejorar las técnicas plug-and-play es decir permitir a los dispositivos conectarse y desconectarse sin necesidad de reiniciación configurándose automáticamente al ser conectados ;además se

dotó de transmisión de energía eléctrica para los dispositivos conectados. Este bus tiene una estructura de árbol y se pueden ir conectando dispositivos en cadena, pudiendo conectarse hasta 127dispositivos permitiendo la transferencia síncrona y asíncrona.

Se puede clasificar según su velocidad de transferencia de datos (desde kilobits hasta megabits): Baja Velocidad (1.0) utilizado para los dispositivos de interfaz Humana(HID) como ratones, teclado; Velocidad Completa (1.1) y Alta Velocidad (2.0) para conexiones a internet.

USB es un bus punto a punto, con inicio en el HOST y destino en un dispositivo o en un HUB; solo puede existir un único HOST en la arquitectura USB. HOST se define como el dispositivo maestro que inicia la comunicación y HUB es el dispositivo que contiene uno o más conectores o conexiones hacia otros dispositivos USB, cada conector es un puerto USB. El protocolo de comunicación se basa en el paso de testigo (token), donde el HOST proporciona el testigo al dispositivo seleccionado y este le devuelve el testigo como respuesta.

➢ **Migración de RS232 a USB**

La interfaz serie RS-232 está desapareciendo practicamente de las PC personales y esto supone un problema ya que muchas de las aplicaciones con microcontroladores utilizan este bus para su comunicación con la PC. La solución ideal es migrar a una interfaz USB y existen diferentes formas de hacerlo. El método más sencillo es emular RS-232con el USB, con la ventaja de que la PC verá la conexión USB como una conexión COM RS-232.

Una clase USB es una agrupación de dispositivos de características comunes es decir; utilizan una misma forma de comunicarse con el entorno. La clase de dispositivo permite conocer la forma en que la interfaz se comunica con el sistema, el cual puede localizar el driver que puede controlar la conectividad entre la interfaz y el sistema. (Figura A.9).

USB solo permite al driver comunicarse con el periférico a través de las tuberías (pipes) establecidas entre el sistema USB y los endpoints del periférico. Los tipos de transferencia a través de las pipes dependen del endpoint y pueden ser: Bulk,Control,Interrupt e Isochronous. Una tubería es un enlace virtual entre HOST y el dispositivo USB donde se configura el ancho de banda el tipo de transferencia la dirección del flujo de datos y el tamaño del paquete de datos. Estos enlaces se definen y crean durante la inicialización del USB. Un endpoint es un buffer dentro del dispositivo o periférico donde se almacenan paquetes de información todos los dispositivos deben admitir el endpoint0 el cual recibe

el control y las peticiones de estado durante la enumeración del dispositivo. Cuando se conecta un dispositivo al HOST se produce la enumeración en la cual el HOST interroga al dispositivo sobre sus características principales asignándole una dirección y permitiendo la transferencia de datos. La especificación Clase de Dispositivo de Comunicación (CDC) define algunos modelos de comunicación incluyendo la comunicación serie.

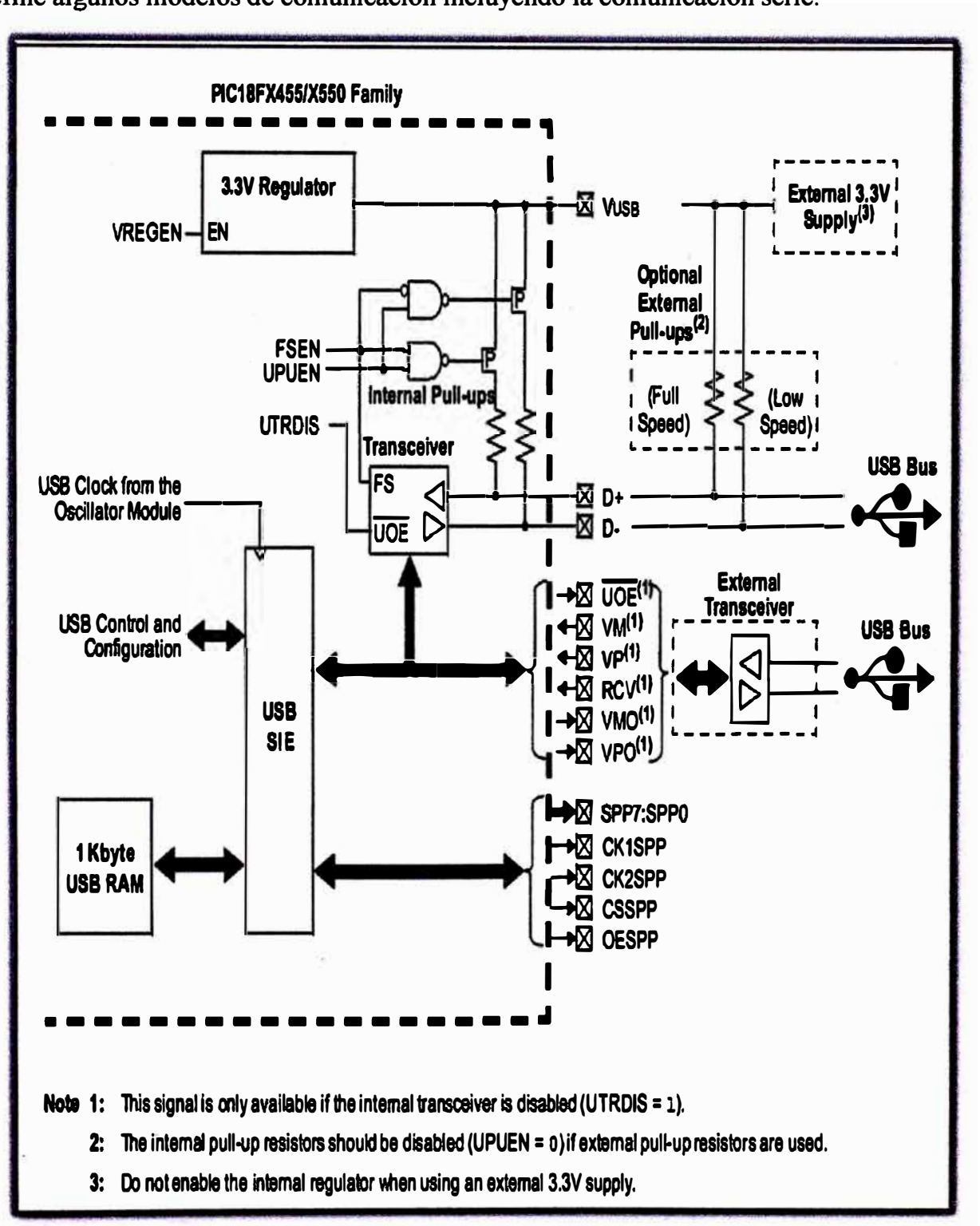


Fig. A.9 Periféricos y Opciones USB
(Hoja de datos de Microchip)[3]

Para la especificación CDC se necesitan dos interfaces USB, primero la interfaz communication class usando IN interrupt endpoint de interrupción y el segundo es la interfaz Data Class usando un OUT bulk endpoint y un IN bulk endpoint. Desde el punto de vista del sistema USB, el dispositivo puede tener distintas configuraciones para una de las cuales puede funcionar de forma distinta.

Los dispositivos suministran la información necesaria al sistema USB a través de los descriptores; estos contienen unos campos que permiten al sistema clasificar al dispositivo y asignarle un driver. La primera información que necesita es la del fabricante y producto (USB vendor ID-VID y el productID-PID). El VID es un número de 16bits asignado por el USB.

Microchip suministra su VID y los PID para cada familia de PIC con USB.

➢ **USB con ISIS y CCS C**

Se ha incluido dos herramientas para la simulación de circuitos con USB:

- El conector USB llamado USBCONN, el cual permite conectar y desconectar el bus.
- El visualizador de USB llamado Analizador de Transiciones USB.

Además del software se incorpora los drivers necesarios para la simulación con USB. Para instalarlos hay que ir a la opción INICIO >PROGRAMAS >PROTEUS 7 PROFESIONAL>VIRTUAL USB>INSTALL USB DRIVERS.

Tras este proceso ya se puede trabajar con el USBCONN.

➢ **USB en CCS C:** CCS suministra librerías para comunicar PIC con la PC utilizando el bus USB, mediante periféricos internos, estas librerías suministradas son:

- **pic_18usb.h:** Driver de capa de hardware de la familia PIC18.
- **usb.h:** definiciones y prototipos utilizados en el driver USB.
- **usb.c:** El USB stack, que maneja las interrupciones USB y el USB Setup Requests en el endpoint 0 .
- **usb_cdc.h:** Driver que permite utilizar una clase de dispositivo CDC USB emulando un dispositivo RS-232.
- **usb_init():**Inicializa el hardware USB. Espera en un bucle hasta que el periférico USB es conectado al bus (aunque esto no significa que ha sido enumerado por la PC). Habilita y utiliza la interrupción USB.
- **usb_task():**Si se utiliza una detección de conexión para la inicialización entonces se

debe llamar periódicamente a esta función para controlar el pin de detección de conexión. Cuando el PIC es conectado desconectado del bus esta función inicializa el periférico o resetea el USB stack y el periférico.

- **usb_enumerated():** Devuelve un TRUE si el dispositivo ha sido enumerado por la PC y en este caso el dispositivo entra en modo de operación normal y puede enviar y recibir paquetes de datos.

Las funciones específicas para CDC (Clase de Dispositivo de Comunicación) son:

- **usb_cdc _putc(c): E**s idéntica a put(c) y envía un caracter. Coloca un carácter en el buffer de transmisión en el caso de que esté lleno esperará hasta que pueda enviarlo.
- **usb_cdc _getc(c):** Es idéntica a get(c) y lee un caracter. Recibe un caracter del buffer de transmisión en caso de estar vacio esperara hasta que se reciba.

Para el PIC18 F es importante configurar el de detección del sentido de transmisión (USB_CON_SENSE_PIN) para poder controlar la conexión o desconexión del PIC al USB. Antes de utilizar el puerto en el programa hay que inicializarlo (usb_cdc_init() y usb_init()) y "muy importante ",comprobar que ha sido enumerado por el HOST(usb_enumerated()). Con el fin de inicializar o resetear la conexión con el USB se debe realizar una llamada periódica a la función usb_task(). En este instante el PIC queda conectado a la PC. [2]

ANEXO B
RESULTADOS DEL TOOL BOX DE IDENTIFICACIÓN DE MATLAB

ANEXO B

RESULTADOS DEL TOOL BOX DE IDENTIFICACIÓN DE MATLAB

- **Muestras cada 5ms**

%V1=5V

V1=pi*[0 3 5 8 10 11 12 14 14 14 15 15 16 15 15 15 15 16 15 15
16 15 15 16 15 16 15 16 15 16 16 15 16 15 16 16 15 16 16 16 15
16 16 15 16 16 15 16 16 15 16 16 15 16 16 15 16 15 16 16 15 16
15 16 15 16 15 16 15 16 15 15 16 15 16 15 16 15 16 15 16 15 16
16 15 16 15 16 16 15 16 16 15 16 16 15 16 15 16 16 15 16 16 15
16 16 15 16 15 16 16 15 16 16 15 16 16 15 16 16 15 16 16 15 16
15 16 16 15 16 16 15 16 15 15 16 15 16 16 15 16 16 15 16 16 15
16 15 16 15 16 16 15 16 15 16 16 15 16 15 16 16 15 16 16 15 16
16 16 15 16 16 15 16 16 15 16 15 16 16 16 15 16 15 16 16 15 16
16 15];

Ve1=5*ones(1,190);
%V(tf)=49.0248
%V(real)=16*pi=50.2656

%V2=10V

V2=pi*[0 1 2 5 10 14 20 24 27 29 30 31 31 32 33 32 33 32 33 33
33 33 33 33 32 33 33 33 32 33 33 33 33 32 33 33 32 33 33 32 33
33 33 33 32 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 34 33
33 33 33 33 34 33 33 34 33 33 33 34 33 33 33 34 33 34 33 34 33
33 34 33 33 33 33 33 33 33 34 33 33 33 33 33 33 33 33 33 33 33
33 33 33 33 32 33 33 33 33 33 33 33 34 33 33 33 33 33 33 33 33
34 33 33 34 33 33 33 34 33 33 34 33 33 34 33 34 33 34 33 33 34
33 33 34 33 33 34 33 33 33 34 33 33 33 33 33 34 33 33 33 33 34
33 33 33 33 33 34 33 33 33 34 33 33 33 34 33 33 33];

Ve2=10*ones(1,184);
%V(tf)=104.1998
%V(real)=33*pi=103.6728

%V3=15V

V3=pi*[0 2 5 9 13 21 26 31 35 36 40 43 45 47 47 49 49 49 50 50
50 50 50 50 50 51 50 50 50 51 50 50 51 50 51 50 51 50 50 51 50
51 50 51 50 50 51 50 50 51 50 51 50 51 50 50 52 50 51 50 51 50
50 51 50 51 50 50 51 50 50 50 51 50 50 50 51 50 50 50 51 50 51
50 50 51 50 51 50 51 50 51 50 50 51 50 50 51 50 51 50 51 50 50

51 50 51 50 50 51 50 50 51 50 50 50 51 50 50 50 50 50 50 50 50
50 51 50 50 50 51 50 50 50 50 51 50 50 51 50 50 50 50 50 50 51
50 51 50 50 50 51 50 50 51 50 50 51 50 50 51 50 50 51 50 50 51
50 50 51 50 50 50 50 50 51 50 51 50 50 51 50 50 52 50 50 51 50
51 50 50 51 50 50 51 50 50 51 50 51 50 50 51 50 50 51 50 51 50
50 51 50 51 50 50 51 50 51 50 50 51 50 50 51 50 51 50 50 51 50
50 51 50 51 50 50 51 50 50 51 50 51 50 50 51 50 51 50 50 51 50
51 50 50 51 50 50 52 50 51 50 50 51 50 51 50 50 51 50 50 51 50
51 50 51 50 50 51 50 51 50 50 51 50 51 50 50 51 50 51 50 50 51
50 50 51 50 51];

Ve3=15*ones(1,298);
%V(tf)=158.2115
%V(real)=50*pi=157.08

%V4=20V

V4=pi*[0 1 4 7 12 18 25 31 34 38 44 49 54 58 60 63 64 65 65 66
66 67 67 67 67 67 68 67 68 67 67 67 68 67 68 68 68 67 68 68 68
68 68 67 68 68 67 68 67 68 68 67 68 67 68 68 67 68 68 67 68 67
68 68 68 67 68 68 67 68 68 68 67 68 68 68 67 68 68 68 67 68 68
68 67 68 68 68 68 67 68 68 68 68 67 68 68 68 67 68 68 68 68 67
68 68 68 68 67 68 68 68 68 67 68 68 68 68 67 68 68 68 68 67 68
68 68 68 68 67 68 68 68 67 68 68 68 68 67 68 68 68 68 67 68 68
68 68 68 67 68 68 68 68 67 68 68 68 68 67 68 68 68 68 68 67 68
68 68 68 67 68 68 68 68 68 67 68 68 68 68 67 67 68 68 68 68 67
68 68 68 68 68 67 68 68 68 68 68 68 68 68 67 68 68 68 68 68 68
68 68 67 68 68 68 68 68 68 67 68 68 68 68 68 68 67 68 68 68 68
67 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
67 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
68 68 68 67 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
68 68 68 68 67 68 68 67 68 68 68 68 68 68 68 68 68 68 68 68 67
68 68 68 68 68 68 68 68 68 68 68 68 68 67 68 69 68 68 68 68 68 68
68 68 68 68 68 68 68 68 68 67 68 68 68 68 68 68 68 68 68 68 68 68

68 68 68 67 68 68 68 68 68 68 68 68 68 68 68 68 68 68 67 69 67
68 68 68 68 68 68 68 68 68 68 68 68 68 67 68 68 68 68 68 68 68
68 68 68 68 68 68 68 68 68 67 68 68 68 68 68 68 68 68 68 68 68
68 68 68 68 68 67 68 68 68 68 68 68 68 68 68 68 68 67 68 68 68
68 68 68 68 68 68 68 68 67 67 68 68 68 68 68 68 68 68 68 68 68
67 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 67 68 68 68 68
68 68 68 68 68 68 68 68 67 68 68 68 68 68 68 68 68 68 68 68 68
68 68 68 67 68 68 68 68 68 68 68 68 68 68 68 68 67 68 68 68 68
68 68 68 68 68 68 68 68 67 68 68 68 68 68 68 68 68 68 68 68 68
68 68 68 67 68 68 68 68 68 68 68 68 68 68 68 68 67 68 68 68 68];

Ve4=20*ones(1,902);
%V(tf)=213.3545
%V(real)=68*pi=213.6288

%V5=25V

Vo61=[1 2 3 7 9 14 18 22 27 32 37 42 48 53 59 64 69 73 75 78
80 82 82 83 84 83 84 84 84 85 85 85 85 85 85 85 85 86 85 85
85 85 86 85 85 86 85 85 86 85 85 85 85 85 85 85 86 85 85 85 85
85 86 85 85 85 85 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85
85 85 85 85 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85
85 86 85 85 85 85 86 85 85 86 85 86 85 85 85 85 86 85 85 85 86
85 85 86 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85
85 86 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86 85
85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 86 85
85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 85 86 85
85 85 85 86 85 86 85 86 85 85 85 85 85 85 85 85 86 85 85 85 86
86 85 85 86 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86
85 85 85 85 86 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85
86 85 85 85 85 86 85 85 85 86 85 85 86 85 85 85 85 86 85 85 85
86 85 85 85 86 85 86 85 85 86 85 85 85 86 85 85 85 85 86 85 85
85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 85 85 86
85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 86 85 87 85 86 85
85 85 85 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86
85 85 85 85 86 85 85 85 86 86 85 85 85 86 85 85 85 85 86 85 85
85 85 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86 85
85 85 85 85 85 85 86 85 85 85 86 85 85 86 86 85 85 85 85 86 85
85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85
86 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85
85 85 86 85 85 85 85 86 85 86 85 85 86 85 85 85 85 85 85 86 85
86 85 85 85 85 86 85 85 85 85 85 85 85 85 85 85 85 85 85 86 85
85 85 85 86 85 85 85 86 85 85 86 85 86 85 86 85 86 85 85 85 85
86 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85
85 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 85 86
85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 85
85 86 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85

85 85 86 85 85 85 86 85 85 85 85 85 85 85 85 86 85 85 85 86 85
85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 85 85
86 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85
85 86 85 85 85 85 86 85 85 85 86 85 85 85 85 85 85 85 85 85 85
85 86 85 85 86 86 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85
85 85 86 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 86
85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85
85 86 85 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 85 85 86
85 85 85 85 86 85 85 85 86 85 85 85 86 85 85 85 86 85 85 85 86
85 85 85 85 86 86 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85
85 86 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85 85 85
86 85 85 85 85 86 85 85 85 85 85 85 85 85 85 85 86 85 85 86 86
85 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85
85 85 86 85 85 86 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85
86 85 85 85 85 86 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85
85 85 85 85 85 85 85 85 85 85 85 85 85 85 85 85 86 86 85 85 85
85 85 85 85 85 85 85 85 85 85 85 86 85 85 85 85 86 85 85 85 86
85 85 86 85 86 85 85 85 85 86 85 85 85 86 86 85 85 85 86 85 85
85 86 85 85 85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 85 85
85 85 85 86 85 85 85 85 86 85 85 85 85 85 85 85 85 86 85 85 85
86 85 85 85 85 85 85 85 85 85 85 85 85 85 86 85 85 85 85 86 85
85 85 85 85 85 86 85 85 85 85 86 86 85 85 85 85 85 86 85 85 85
85 85 85 85 86 85 85 85 85 85 86 85 85 85 85 85 85 85 86 85 85
85 85 85 86 85 85 85 85 85 85 85 86 85 85 85 85 86 85 85 85 85
85 86 85 85 85 85 85 85 86 85 85 85 85 85 86 85 85 85 85 86 85
85 85 85 85 86 85 85 85 85 86 86 85 85 85 86 85 85 85 85 85 86
85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85
85 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85
85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85
85 85 85 85 86 85 85 85 85 85 85 85 85 86 85 85 85 85 86 85 85
85 85 86 85 85 85 86 85 85 85 85 86 85 85 85 85 86 85 85 85 86
85];

p=85*ones(1,156);

Vo6=pi*[Vo61,p];

Vi6=25*ones(1,1500);

%Kp=10.711

%Tp1=0.028929

%Tp2=0.028937

%V(tf)=267.7893

%V(real)=267.0354

Procedimiento de Identificación del Tool Box MATLAB para 5V

MATLAB tiene una herramienta importante para lograr obtener la F.T.

Iniciamos la identificación, colocando "ident" en el comando windows de MATLAB, abriéndose una ventana donde se procede a configurar los parámetros para obtener la función de transferencia.

Estos pasos pueden apreciarse en los gráficos desde la Figura B.1 hasta la Figura B.10

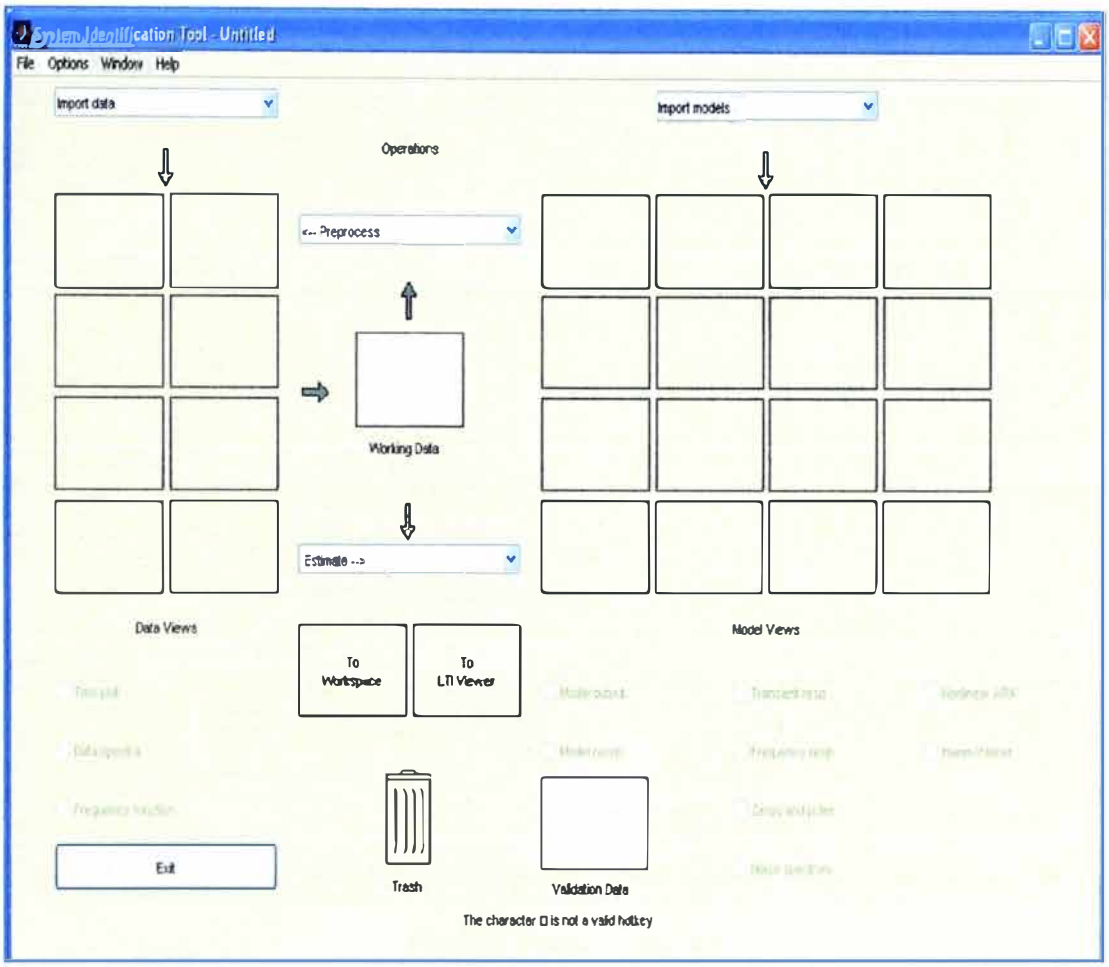


Fig.B.1 System Identification Tool

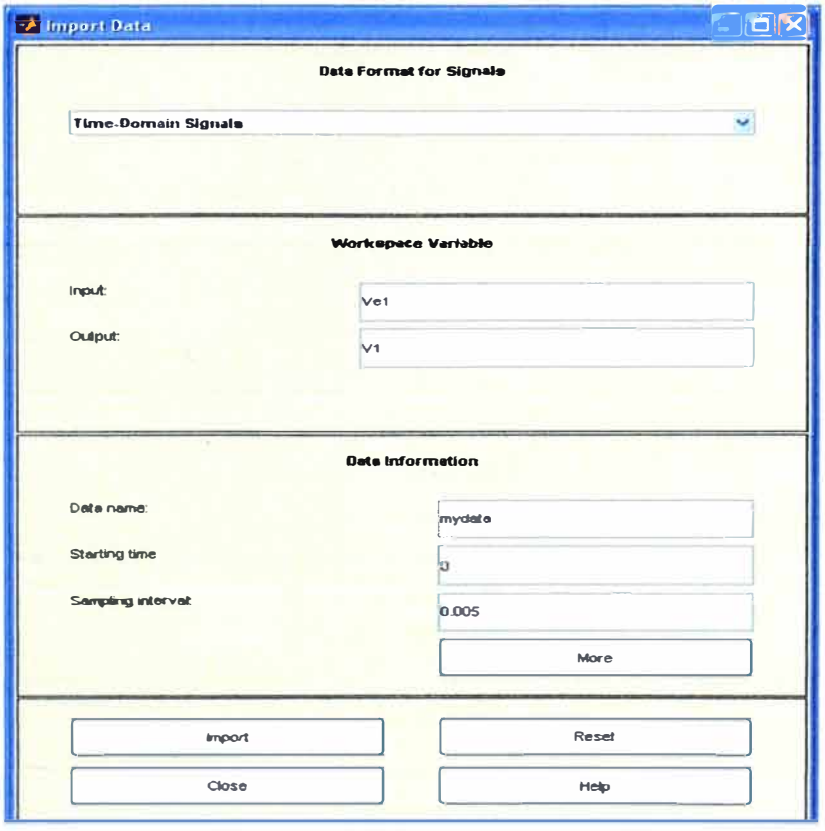


Fig.B.2 Import Data

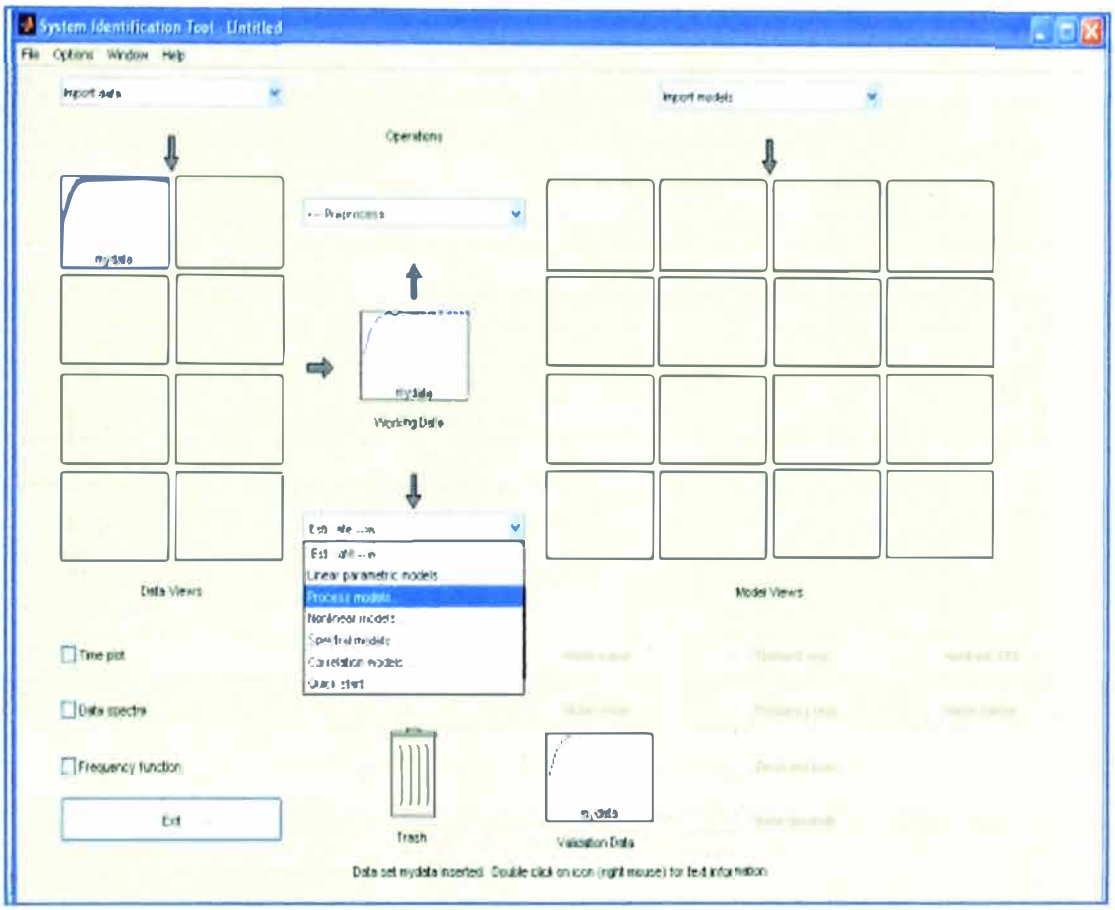


Fig.B.3 Process Model

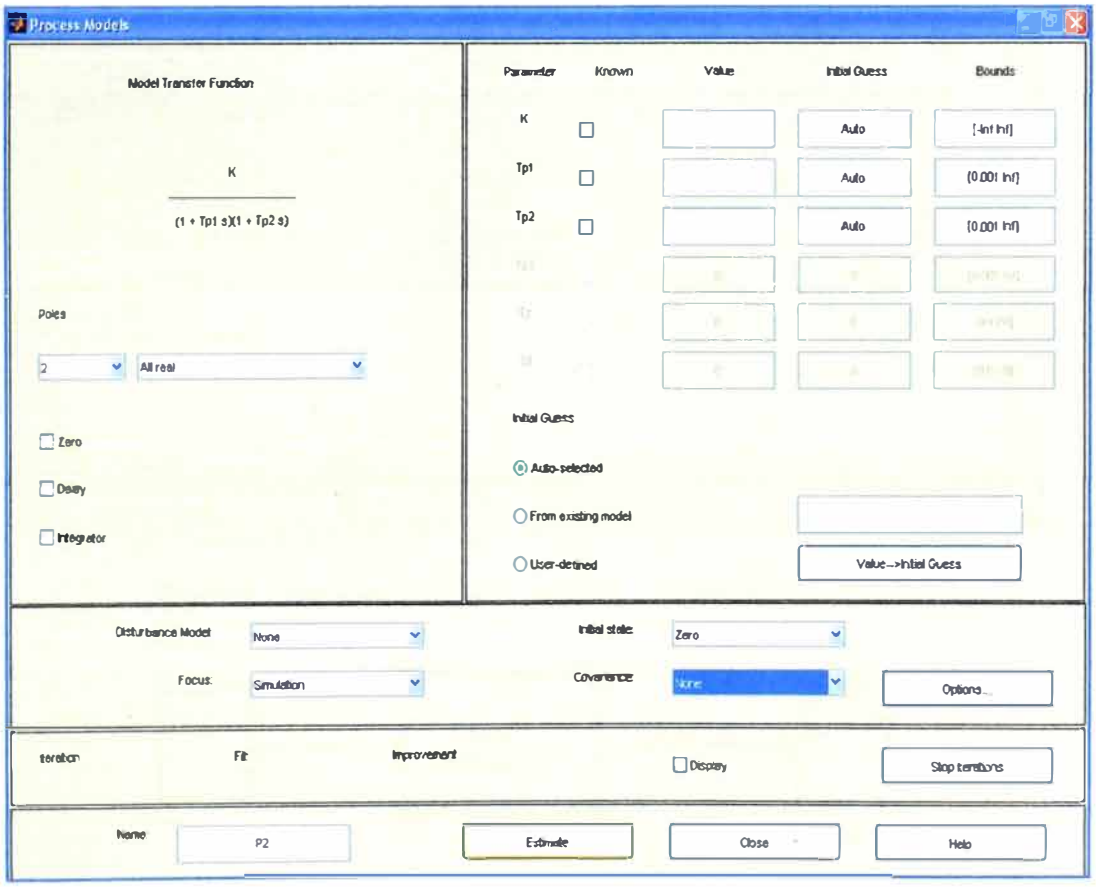


Fig.B.4 Configuración de la F.T

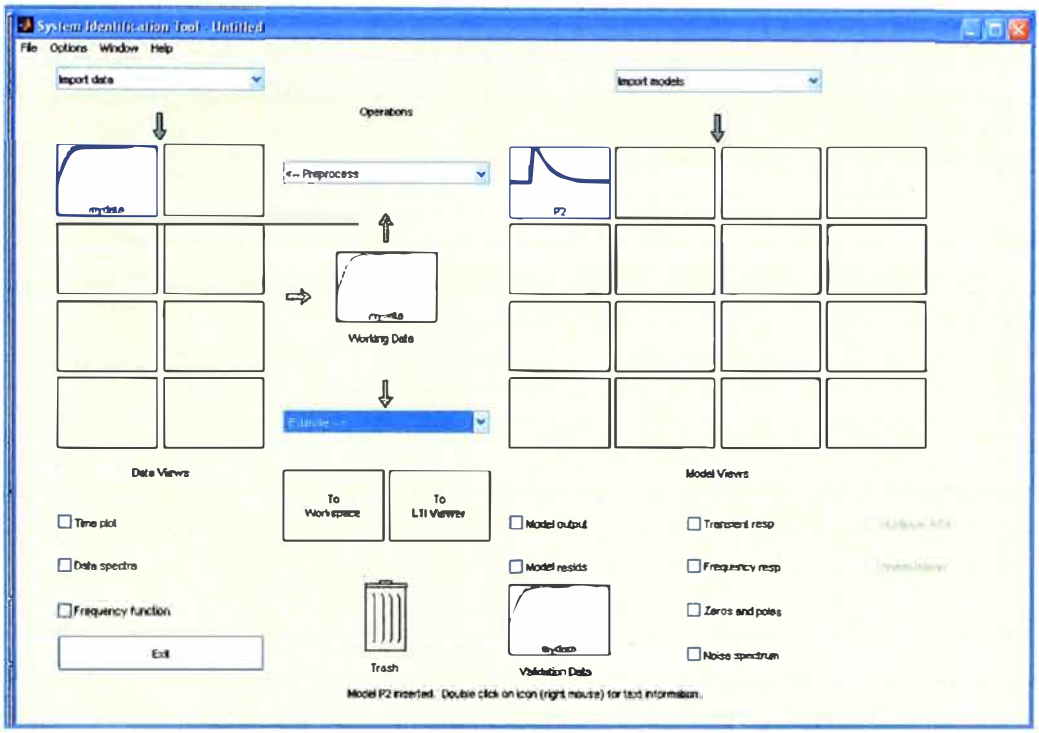


Fig.B.5. Configuración de la F.T

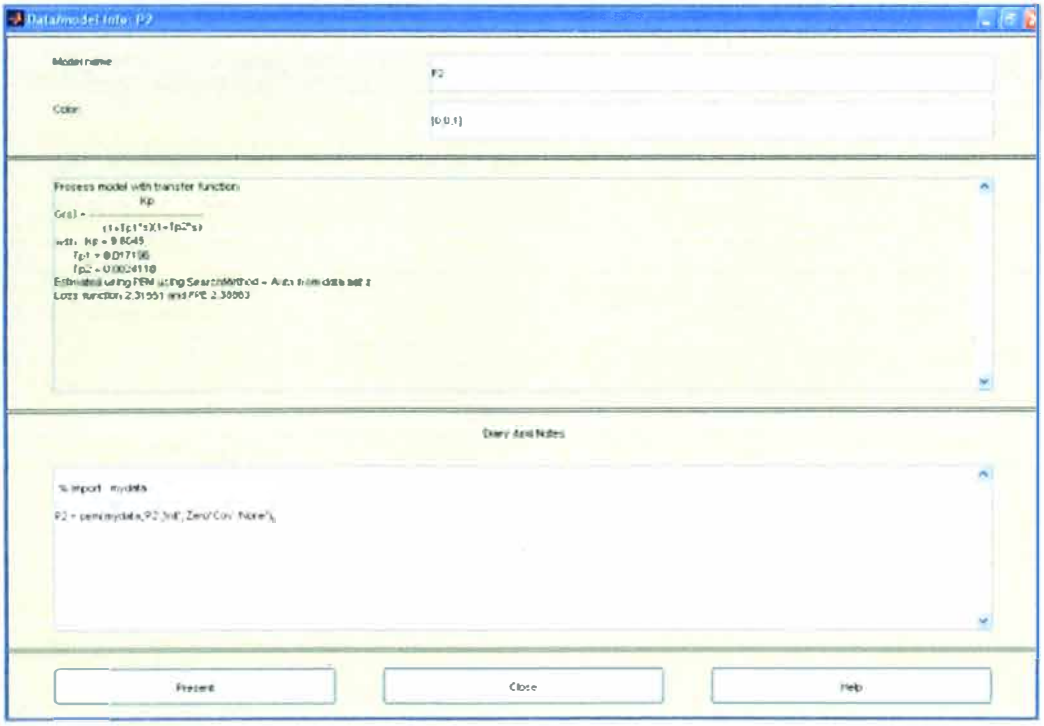


Fig.B.6 Función de Transferencia del Motor DC para 5V.

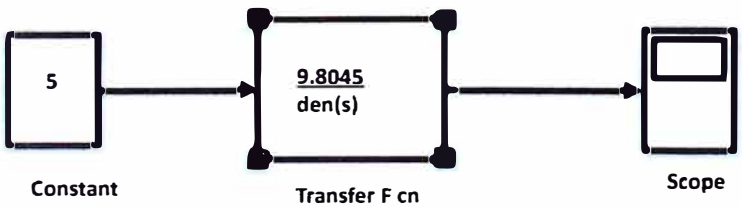


Fig.B.7. MATLAB - Simulink

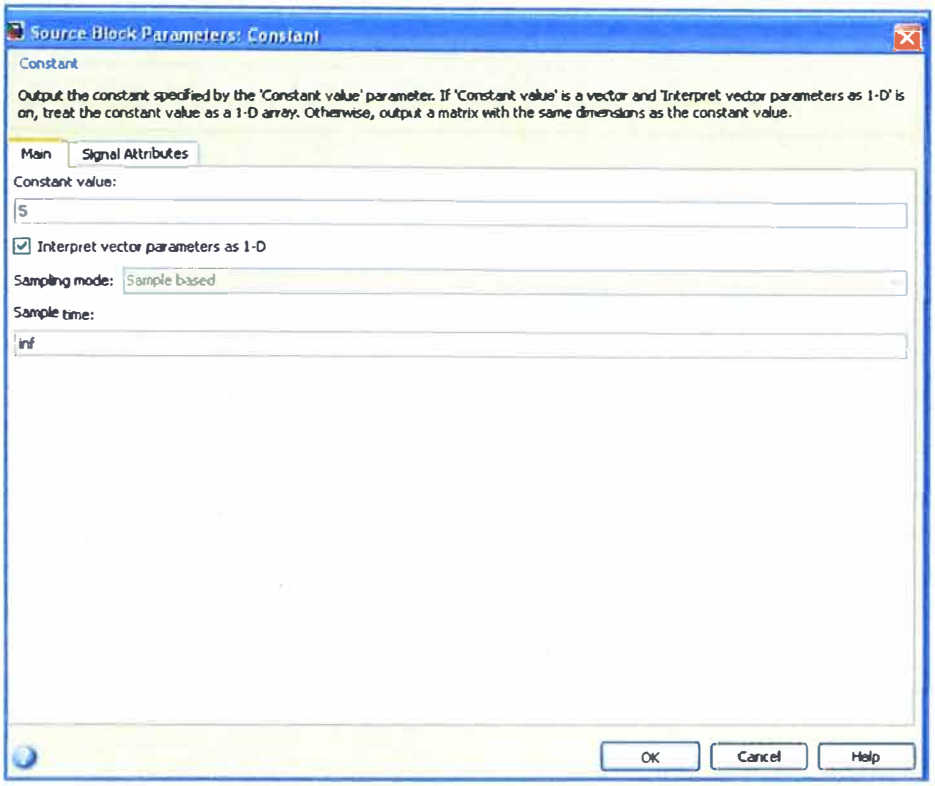


Fig.B.8 Parámetro Constante

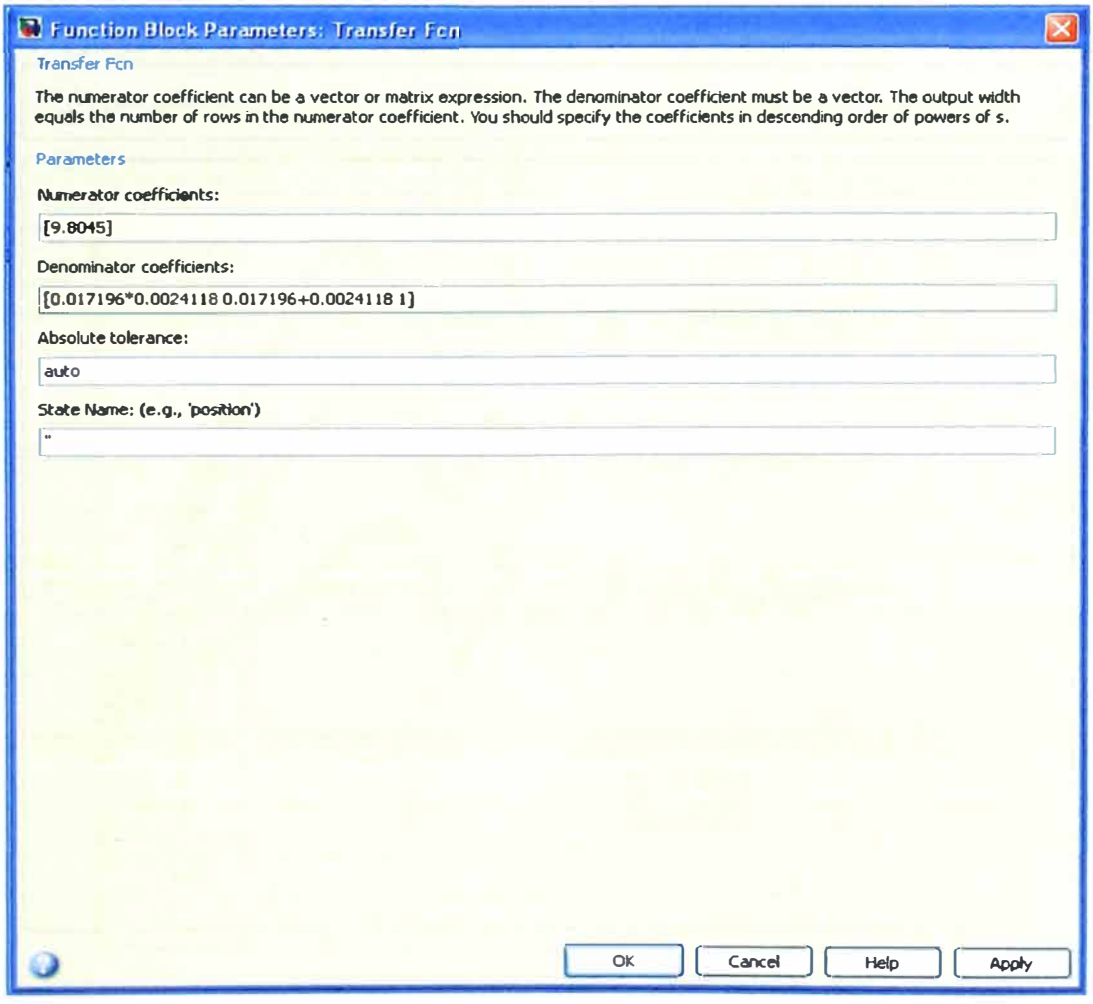


Fig.B.9 Valores de la Función de Transferencia

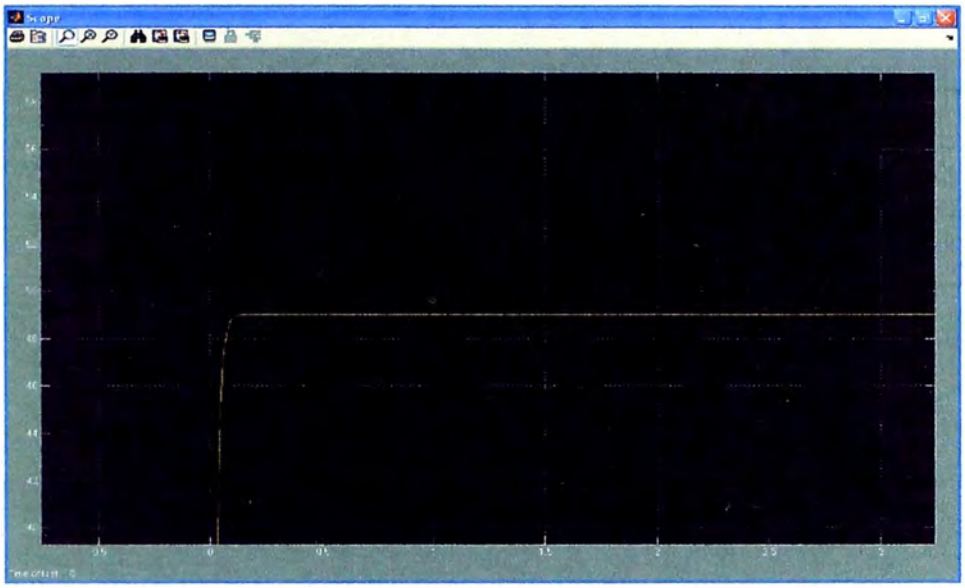


Fig.B.10 Resultados de Simulink (Teórico)

En la Figura B11,se puede observar que para las diferentes Tensiones DC se obtienen diferentes cantidades de muestras, esto trae una conclusión que a mayor tensión (Escalón) se requiere mayor cantidad de muestras.

La mayor cantidad de muestras conducen a obtener un modelo de la función de transferencia más exacta para el motor DC.

También es necesario obtener la mayor cantidad de muestras para el análisis del motor DC en su 2% ó el 5% de error en estado estacionario.

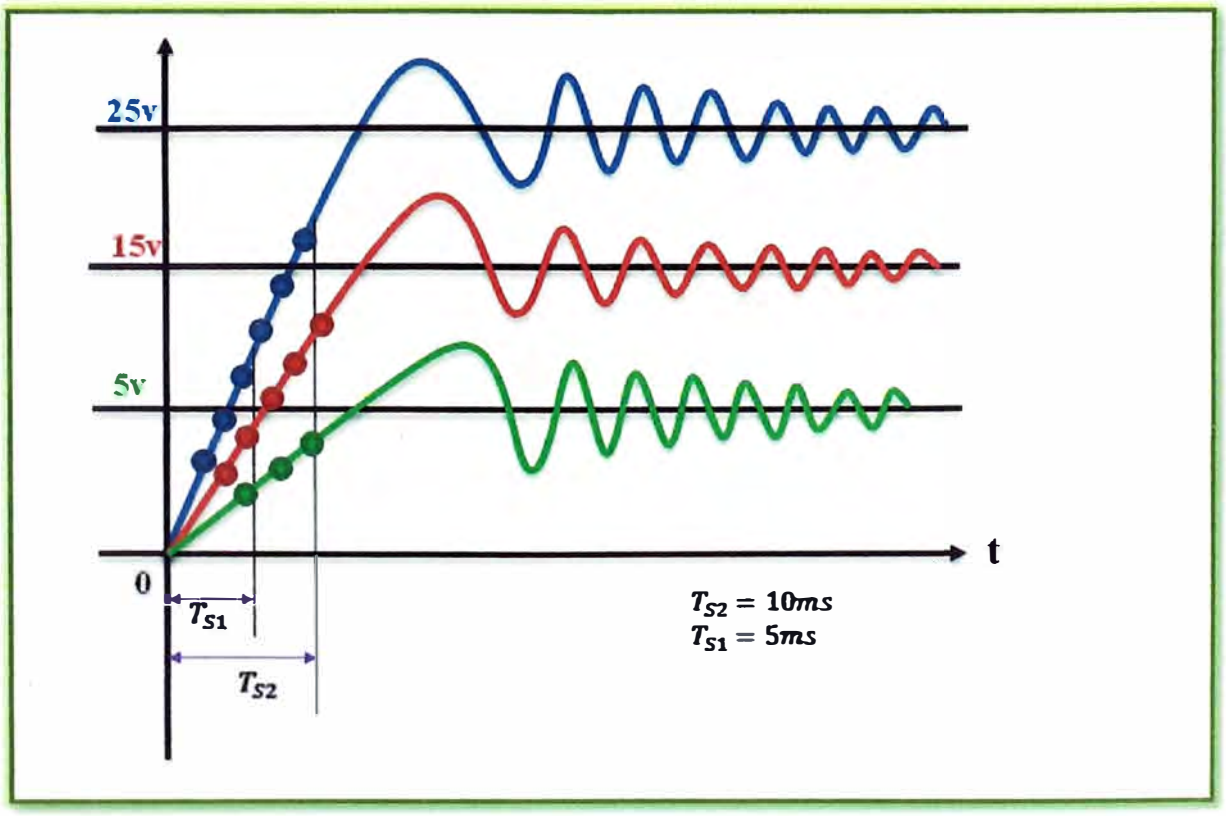


Fig.B.11 Muestras Obtenidas con las tensiones de Arranque aplicados al Motor DC

BIBLIOGRAFÍA

[1] ING. Gabriel Pool Balam ``Transferencia y procesamiento de datos de alta velocidad mediante el uso de MATLAB, el puerto USB2.0 y PIC18F2455´´,

Universidad Modelo – México, 2009.

[2] Eduardo Garcia Breijo `` Compilador C CCS y Simulador PROTEUS para Microcontroladores PIC´´, Edición Alfaomega-Marcombo – Barcelona, 2008.

[3] Foxi Reader ``Data Sheet MICROCHIP´´, Microchip Technolo°gy Inc-2007