UNIVERSIDAD MODELO

ESCUELA DE INGENIERÍA

MAESTRÍA EN MECATRÓNICA

Transferencia y procesamiento de datos a alta velocidad, mediante el uso de MATLAB, el puerto USB 2.0 y PIC18F2455 de Microchip™

PRESENTA:

ING. GABRIEL JESÚS POOL BALAM

PARA LA MATERIA:

DISEÑO MECATRÓNICO

PROFESOR:

DR. ALEJO MOSSO VAZQUEZ

MÉRIDA, YUCATÁN A 4 DE JULIO DE 2009

Agradecimientos:

Al Centro de Investigación científica de Yucatán A. C. (www.cicy.mx) sobre todo al departamento de Instrumentación representado por Ing. Leonardo Gus Peltinovich por su apoyo para la realización de éste trabajo.

JUNIO 2009

RESUMEN

MATLAB ("*matrix laboratory*") es un software de computación y desarrollo de aplicaciones, diseñado para llevar a cabo proyectos que impliquen cálculos matemáticos de nivel elevado y su representación gráfica. MATLAB integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno tal que los problemas y sus soluciones son expresados del mismo modo en que se escribirían, sin necesidad de hacer uso de la programación tradicional. MATLAB es capaz de manejar librerías y dll's de diferentes dispositivos instalados en la PC.

Microchip™ proporciona en su página *web* los *drivers* y archivos necesarios para establecer la comunicación por puerto USB con la familia del PIC18F2455. A través de ellos se puede efectuar transacciones de hasta 64 *bytes* por paquete cada milisegundo, por cada túnel abierto. El PIC es programado mediante "*PCWH Compiler de CCS*", versión 3.246. Los descriptores utilizados para la comunicación USB están basados en los archivos que contiene el propio compilador.

En este trabajo se muestra que, con estas herramientas y manipulando el archivo mpusbapi.dll, se puede transferir datos a alta velocidad del PIC18F2455 a la PC de manera bidireccional, con la finalidad de procesarlos matemáticamente y/o graficarlos. Un ejemplo de esta aplicación se muestra en este trabajo, y consiste en la obtención de curvas de arranque de motores de CD sin carga, a efectos de su caracterización como paso previo al diseño o selección de su controlador.

INTRODUCCIÓN

Hoy en día, la miniaturización de componentes electrónicos, la disminución de su consumo de corriente y el tamaño de los semiconductores, permite la construcción de circuitos más complejos, inteligentes y eficientes. No ha sido la excepción con el microcontrolador, ya que éste se ha desarrollado notablemente, al punto de hacer más grande sus capacidades de comunicación, memoria, circuitos de apoyo adicionales (ADC, oscilador interno, puertos, etc.). Dichos microcontroladores, requieren de muy poco componentes de soporte externo para implementarse y sus tecnologías se han perfeccionado, de tal manera, que casi no fallan. Algunos fabricantes tal como Microchip, se preocupan en desarrollar las herramientas para su programación, tales como lenguajes de alto nivel para compilar el programa y programadores económicos para descargar el programa en el chip. Debido a éstas herramientas, éste chip se ha vuelto muy popular al grado que se proponen como materias a cursar en las escuelas.

Por otro lado, ya no basta con 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 como por ejemplo: El arranque de un motor (velocidad con respecto al tiempo) que tiene una duración aproximada de 1 décima de segundo. Esto se puede lograr con el desarrollo de un software como Visual

Basic, Visual C, C++, ó cualquier otro paquete de preferencia gráfico para hacer más fácil la interfaz de usuario. Desafortunadamente, cuando se requieren procesar matemáticamente éstos datos (graficarlos, aplicar operaciones de estadística, procesar con filtros digitales, etc.), se vuelve literalmente imposible.

Con MATLAB se resuelve ese problema, pero existe un detalle, el hecho de importar los datos en tiempo real con alguna tarjeta de adquisición de datos requiere compatibilidad con éste programa sin contar adicionalmente con los recursos económicos para la compra de ésta tarjeta. Afortunadamente MATLAB ha evolucionado al grado de que se puede conectar casi con cualquier dispositivo, y su manejo es relativamente fácil e incluye una ayuda en línea.

La finalidad de éste trabajo es desarrollar una comunicación entre un micro controlador y éste software para el tratamiento de datos de manera más económica y sin algún software ejecutable que sirva de intermediario.

ANTECEDENTES

1. LA ARQUITECTURA DEL PUERTO USB

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

Sólo puede haber un Host en el bus que maneja a todos los componentes conectados como se indica en la figura 1.

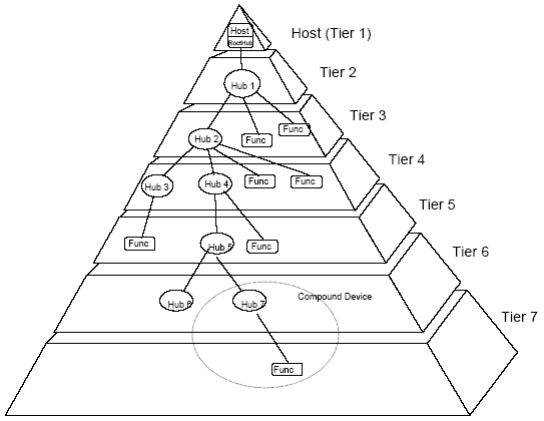


Fig. 1 Topología del BUS

El hub es un elemento plug and play en la estructura USB (Figura 2) y es un concentrador al cual, se le pueden agregar más dispositivos USB, incluyendo otro hub.

La velocidad de transferencia depende de la velocidad del HUB que se esté utilizando. Ver figura 3. Las velocidades típicas son: 480Mb /s en high speed, 12Mb/s full-speed y 1.5Mb/s en Low-speed.

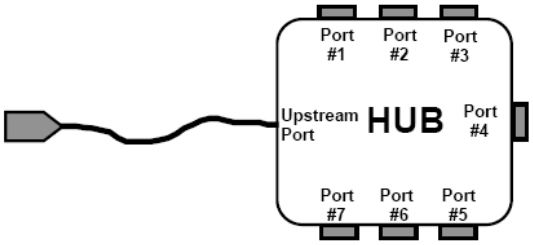


Fig. 2 HUB USB

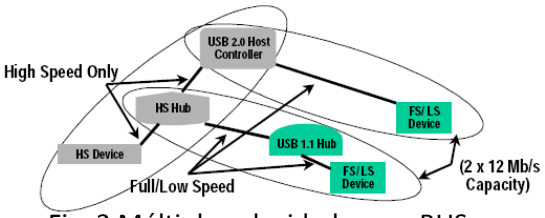


Fig. 3 Múltiple velocidad en un BUS

El cable que típicamente es usado no es largo, debido a la velocidad de transferencia y tiene la estructura mostrada en la figura 4. Sus terminales son del tipo diferencial y consta de 4 hilos.

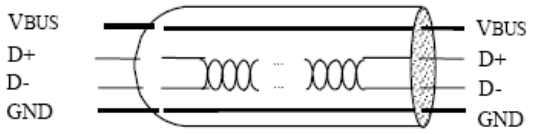


Fig. 4 Cable USB

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

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

• Bulk Data Transfers: Entrega el dato por volumen, el ancho de banda puede variar. Es usado en escáner ó cámaras. La ráfaga de datos es secuencial.

• Interrupt Data Transfers: Used for timely but reliable delivery of data, for example, characters or coordinates with human-perceptible echo or feedback response characteristics.

• Isochronous Data Transfers: Occupy a prenegotiated amount of USB bandwidth with a prenegotiated delivery latency. (Also called streaming real time transfers).

Aunque el PIC no puede funcionar como host, ya que se requiere de una gran capacidad de manejo de datos para administrar a cada componente del BUS, es suficiente que se pueda administrar como un "device", para esto se requiere "memorizarle" los protocolos necesarios para enlazarse al host.

Estos protocolos se le llaman descriptores y sirve para informarle al host todo lo necesario para que pueda administrarlo.

Recordemos que los PICs de la serie 18Fxx5x tienen tres modos de funcionamiento:

1. USB Human Interface Device (HID): Velocidad Baja, no requiere driver.

2. USB Communication Device Class (CDC): Velocidad Media, requiere driver. Crea un Puerto Serie Virtual.

3. USB Custom Driver: Velocidad Alta, requiere driver. Este es el modo que usa WinUSB (para windows vista) y el mpusbapi (windows 2000 y posterior).

Dentro de 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" ó túneles para su comunicación con el host (ver figura 5).

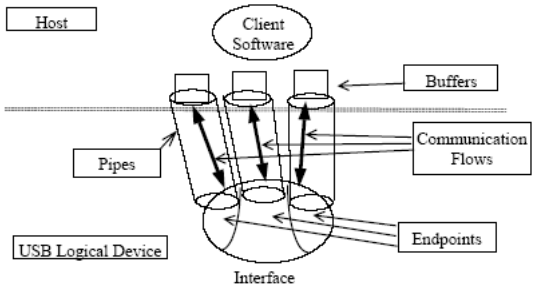


Fig 5. Flujo de comunicación USB

2. LA MPUSBAPI.DLL DE MICROCHIP

Para una mayor facilidad de desarrollo de aplicaciones basadas en el bus USB, Microchip ha creado un archivo 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.

Este driver sirve tanto para Visual Basic como para Visual C, entre otros.

Revisando ejemplos, y la ayuda que trae incluida sobre el manejo de la dll se verifica que el modo de abrir la pipe es el siguiente:

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

(*MPUSBGetDeviceCount)(PCHAR pVID_PID)

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

Seguidamente con la instrucción **(*MPUSBOpen)**

(DWORD instance, // Input

PCHAR pVID_PID, // Input

PCHAR pEP, // Input

DWORD dwDir, // Input

DWORD dwReserved); // Input <Future Use>

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, ésto permite que MPUSBRead, MPUSBWrite y MPUSBReadInt tengan un valor de time-out.

Nota: el valor del 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 cuantos 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 MPUSOpen() 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.

pVID_PID: Input: String que contiene el PID&VID del dispositivo objetivo. El formato es "vid_xxxx&pid_yyyy". Donde xxxx es el valor del VID y el yyyy el del PID, los dos en hexadecimal. Ejemplo: Si un dispositivo tiene un VID=0x04d8 y un PID=0x000b, el string de entrada es: "vid_0x04d8&pid_0x000b".

pEP: Input: String con el número del Endpoint que se va a 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 específicas. Las funciones específicas que utilizan éste parámetro son: MPUSBRead, MPUSBWrite, MPUSBReadInt.

Nota: Para utilizar MPUSBReadInt(), el formato de pEP tiene que ser "\\MCHP_EPz_ASYNC". Esta opción sólo está disponible para un Endpoint interrupción IN. La pipe de datos abierta con "_ASYNC" 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 driver 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 ésta 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 ésta 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 unida tiene que crearse con el atributo de acceso MP_READ. En conclusión, "handle" es el número de pipe que nos arrojó la instrucción anterior con dwDir=1.

pData: Output: Puntero al buffer que recibe el dato leído de 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 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 con 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.

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

Por ultimo, 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)

Existen otras dos instrucciones que no se implementaron en éste desarrollo, pero es necesario conocerlos, éstas 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)

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

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

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.

Esta dll se llama de acuerdo a la convención del lenguaje C, NO funciona si es llamada con el formato de STDLIB.

OBJETIVO

El objetivo principal del presente trabajo es integrar el software de MATLAB con el PIC18F2455 de Microchip con el fin de diseñar una tarjeta de adquisición de datos ajustada a la necesidad personalizada de cada desarrollo.

OBJETIVOS PARTICULARES

1. Enlace a la PC mediante USB.
2. Enlace de MATLAB al PIC.
3. Pruebas finales y correcciones.

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 envío y recepción hacia la PC, en la PC se descarga el 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 teclado, display genéricos de dos líneas y gráficos, el puerto serie, usb, I2C, etc. Si tiene dudas sobre la descarga del driver consulte el apartado 3. Pruebas finales y correcciones en éste documento.

Configurando el Hardware

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

2. Antes de compilar el código de programa <<daq.c>> adjunto en éste archivo comprimido con *PCWH Compiler de CCS* versión 3.246 ó posterior, primero escoja el PIC que utilice PIC18F2455/2550/4455/4550 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 8 Mhz

   PLL3 … para Xtal de 12 Mhz

   PLL4 … para Xtal de 20 Mhz, etc.

4. Abra el archivo C:\Archivos 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 (aprox en la linea 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 is Microchip)

   0x0B,0x00, //product id

   0x01,0x00, //device release number

5. Compile el programa y grábelo en el PIC (Asegúrese de que antes de abrir el archivo a compilar presione el botón “inicio” (ubicado en la parte superior izquierda) ➔ “Close all” del compilador).

NOTA IMPORTANTE: De no completar éstos pasos la PC **NO** detectará al PIC

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 Aplication and markets ➔ USB ➔ MCHPFSUSB FrameworK ➔ Software/Tools ➔ << Microchip Application Libraries v2010-02-09 >> ó simplemente teclee "usb software/tools" en la ventanita de búsqueda y déle un clic en el botón "site search", generalmente el acceso a la página queda en los primeros resultados de la búsqueda, el cual, al darle click lleva directamente al driver. En el mismo paquete incluye ejemplos que incluyen el programa fuente para la compresión 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 mercado: conecte al dispositivo a la PC, en cuanto le solicite los driver, sólo proporcione la dirección donde fue descomprimido el driver (la misma dirección del paso anterior). Si todo es correcto debemos de observar en el administrador de dispositivos un nuevo hardware que se agregó tal como se muestra en la figura 6. **NOTA:** Si Ud. Olvida ó no sustituye correctamente en el descriptor las 3 líneas que se comentan en el código del programa, el compilador CCS compilará correctamente pero al conectarse el PIC en la PC, éste no reconocerá el driver.

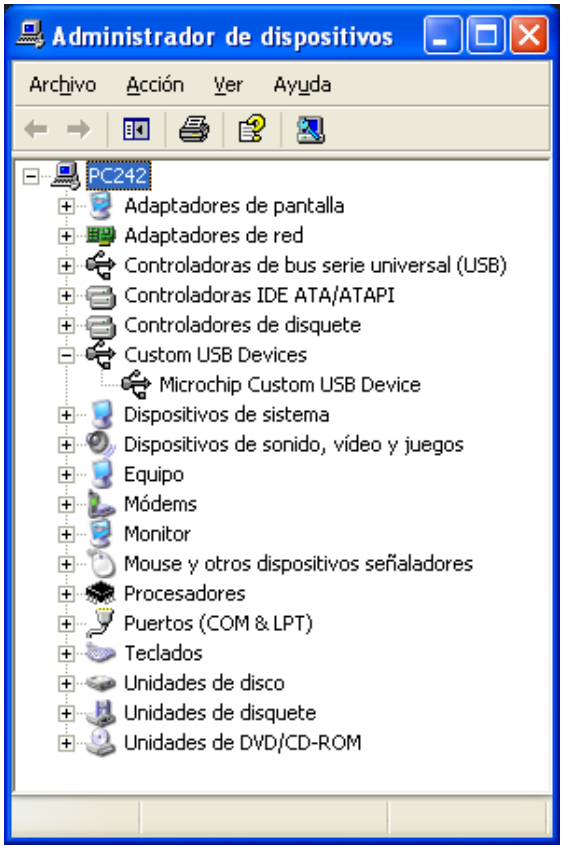


Fig. 6 Instalación del PIC en la PC

4. En propiedades del dispositivo instalado se puede observar el número PID&VID que se configuró en el PIC tal como se muestra en la figura 7.

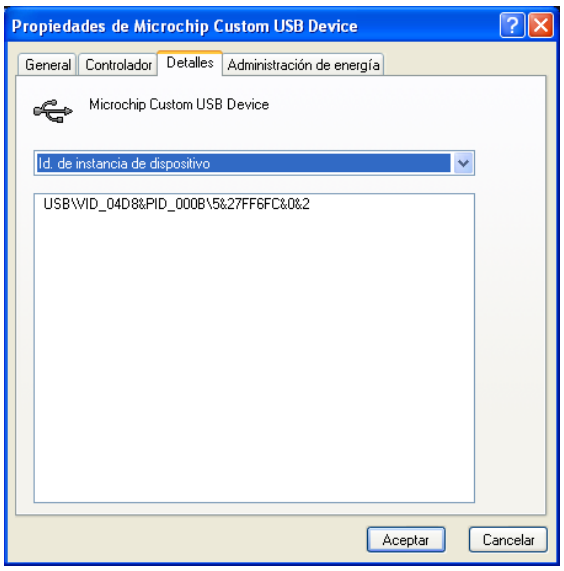


Fig. 7 En propiedades del PIC instalado en la PC se observa el número de PID&VID

Comentarios Adicionales

Todo esto se desarrolló basándose en ejemplos que el compilador trae incluidos en la carpeta de "examples". Cabe aclarar que los diseños en USB con PIC día a día se van incrementando. Sólo basta investigar un poco en la red para ver los resultados (Consulte la bibliografía de éste documento). El código del programa del PIC se encuentra al final del documento con sus respectivos comentarios.

2. ENLACE DEL 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 drivers 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, aún 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 matlab 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 de estar en la misma carpeta que el archivo usb.m

Seguidamente se explica el desarrollo de la comunicación. Primero es necesario conocer qué función de MATLAB es más fácil de implementar:

Existen varios métodos, el que se utilizó fue:

a. Manipular directamente la dll con la instrucción "Loadlibrary"

Ya que se manipula la librería de una manera directa y sin intermediarios.

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

Siguiendo los pasos descritos en la sección de antecedentes de éste documento, las instrucciones que se requieren implementar tienen la siguiente secuencia y formato en MATLAB son:

a. Primero copie 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 version de driver más reciente, reemplace éstos 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 ('libreria', 'MPUSBGetDeviceCount', vid_pid_norm)

De donde:

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

d. Seguidamente abra la pipe para leer (si no desea leer puede omitir éste paso).

Formato:

[my_out_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, out_pipe, uint8(0), uint8 (0));

De 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 (si no desea escribir puede omitir éste paso).

Formato:

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

De 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('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64), 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('libreria', 'MPUSBWrite', my_out_pipe, data_out, uint8(64), uint8(64), uint8(10));

De 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 abiertas windows genera errores y se pierde la comunicación.

Formato:

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

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

IMPORTANTE:

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

Formato:

unloadlibrary librería;

Una vez enlazado con el PIC, los datos pueden fluir las veces que sea necesario de un sentido a otro y manipularlos como se desee, ya que se tiene el completo control del 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 en 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 MATLAB y el PIC.

3. **Pruebas finales y correcciones.**

Antes de compilar el programa y grabarlo en el PIC es necesario asegurarse de que antes de abrir el archivo a compilar presione el botón "inicio" del mismo compilador (ubicado en la parte superior izquierda) ➔ "Close all" . De no hacer esto es posible que se compile el archivo anterior con el que Ud haya trabajado.

Para el enlace con la PC por 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 a la hora 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.

Por lo que respecta 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 funcionará. Para esto no olvide sustituir en el descriptor las 3 líneas que se comentan en el código del programa que se anexa en éste documento.

La DLL que proporciona Microchip se puede descargar desde su sitio web. Asegúrese de obtener la versión más actual. En el mismo paquete incluye ejemplos que incluyen el programa fuente para la compresión 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\MCHPUSB DRIVER\ MCHPUSB DRIVER RELEASE NOTES.HTM"

Ahí se proporciona información sobre la compatibilidad en sistemas operativos y ejemplos desarrollados.

BIBLIOGRAFÍA

http://www.usb.org/developers/docs/

http://slalen.iespana.es/programacion/datos/MPUSBApi.pdf

www.google.com.mx búsqueda “PIC USB”.

http://www.hobbypic.com/

http://www.todopic.com.ar/foros/index.php?topic=13418.0;do=Bookmark

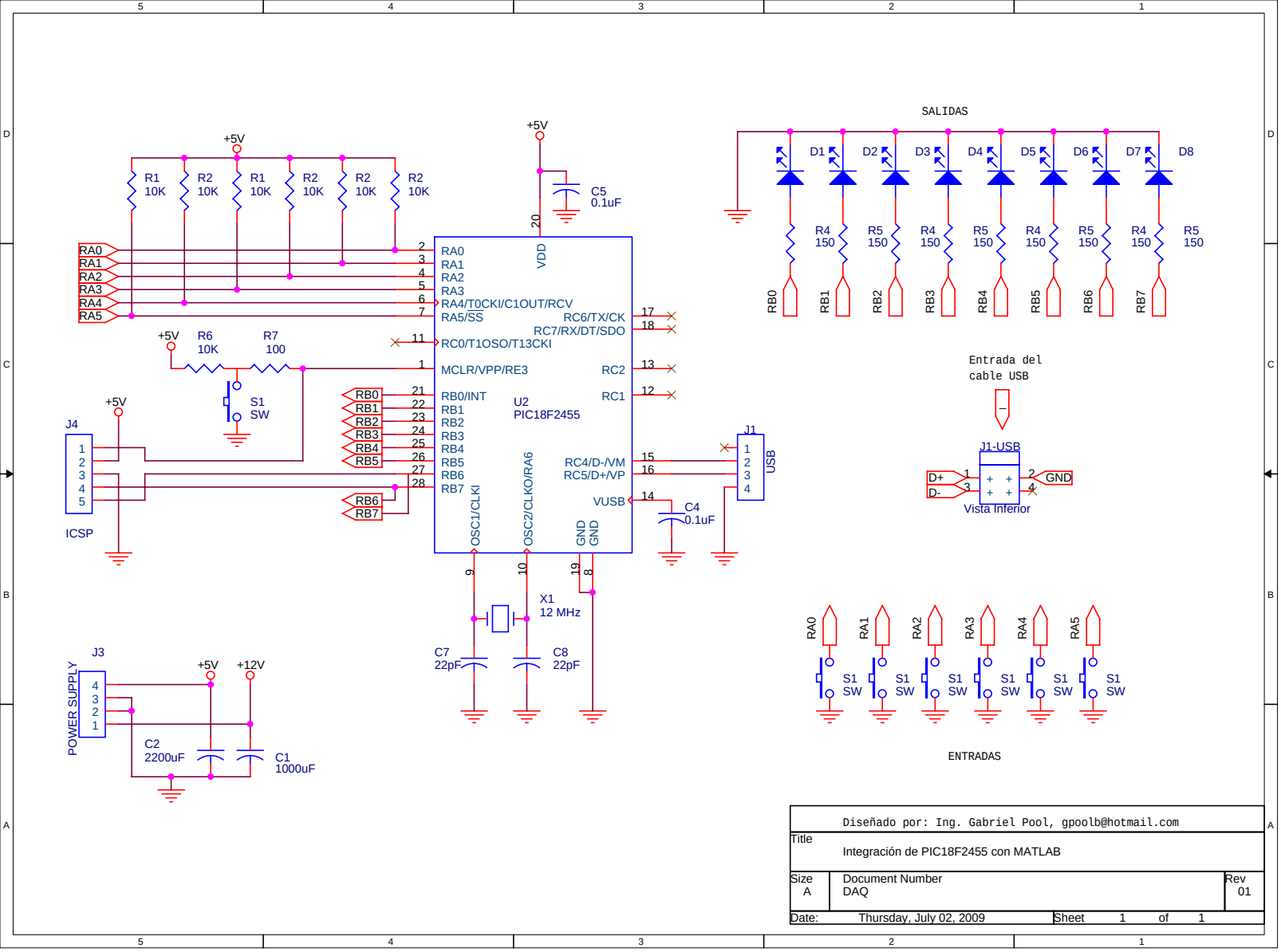
http://picmania.garcia-cuervo.com/index.php

Driver de microchip:

Microchip Application Libraries v2010-02-09

http://ww1.microchip.com/downloads/en/devicedoc/MCHP_App_%20Lib%20v2010_02_09_Installer.zip

Asegúrese que el archivo sea el más actual.


SALIDAS
+5V
R1 10K
R2 10K
C5 0.1uF
RA0
RA1
RA2
RA3
RA4
RA5
RA4/T0CKI/C1OUT/RCV
RA5/SS
RC6/TX/CK
RC7/RX/DT/SDO
RC0/T1OSO/T13CKI
MCLRVPP/RE3
RC2
RC1
RB0/INT
RB1
RB2
RB3
RB4
RB5
RB6
RB7
U2
PIC18F2455
VDD
VUSB
OSC1/CLKI
OSC2/CLKO/RA6
GND
RC4/D-/VM
RC5/D+/VP
D1
D2
D3
D4
D5
D6
D7
D8
R4 150
R5 150
RB0
RB1
RB2
RB3
RB4
RB5
RB6
RB7
R6 10K
R7 100
S1 SW
J4
ICSP
J1
USB
Entrada del cable USB
J1-USB
D+
D-
GND
Vista Inferior
X1 12 MHz
C7 22pF
C8 22pF
C4 0.1uF
J3
POWER SUPPLY
+12V
C2 2200uF
C1 1000uF
ENTRADAS
Diseñado por: Ing. Gabriel Pool, gpoolb@hotmail.com
Title
Integración de PIC18F2455 con MATLAB
Size A
Document Number DAQ
Rev 01
Date: Thursday, July 02, 2009
Sheet 1 of 1