2004 Microchip Technology Inc. DS00956B **AN956**

Migrando aplicaciones de UART-RS232 a USB con mínimo impacto en el software de la PC

Author: Rawin Rojvanit - Microchip Technology Inc.

Traducción: A. Sherar

INTRODUCCIÓN

La interfaz serial RS-232 ya no es un puerto común a todas las computadoras personales (PC). Este es un problema ya que muchas aplicaciones embebidas usan la interfaz RS-232 para comunicarse con un sistema externo tal como una PC. Una solución es migrar la aplicación a una interfaz USB (Universal Serial Bus).

Hay muchas formas diferentes de convertir una interfaz RS-232 a USB, cada una de ellas requiere diferente nivel de habilidad. La más simple es emular la interfaz RS-232 sobre el puerto USB. Una ventaja de este método es que la aplicación de la PC verá la conección USB como un puerto COM RS-232 y por lo tanto no se requerirán cambios en el software de la PC. Otra ventaja es que este método aprovecha controladores de Microsoft® Windows® que ya están incluidos desde W' 98 SE y versiones siguientes, haciendo innecesario el desarrollo de un driver.

El objetivo de esta nota de aplicación es explicar algunos temas básicos requeridos para una mejor comprensión del método de emulación sobre el puerto USB y describir cómo migrar una aplicación existente hacia USB. Un dispositivo que usa la implementación discutida en este documento será referido como un "dispositivo emulador USB a RS-232". El autor asume que el lector tiene algunos conocimientos básicos sobre el estándar USB. Toda referencia a las especificaciones USB en este documento se refiere a la revisión 2.0.

Aspectos de la version 1.0 del "firmware" de la Emulación RS-232:

- Código relativamente pequeño "footprint" de 3 Kbytes para la biblioteca del "firmware"
- Uso de memoria de datos de aproximadamente 50 bytes (excluido el buffer de datos)
- Maxima velocidad de "throughput" de alrededor de 80 Kbytes

• Control de flujo de datos manejado enteramente por el protocolo USB (El método XON/XOFF del puerto RS-232 y el control de flujo por hardware, serán omitidos en esta versión)

• No se requiren controladores adicionales; todos los archivos necesarios, incluido el archivo .inf para Microsoft® Windows® XP y Windows® 2000, ya están incluidos.

RESUMEN

Una aplicación Windows "ve" una conexión física RS-232 como un puerto COM y se comunica con el dispositivo externo usando las funciones CreateFile, ReadFile, y WriteFile. El módulo UART sobre el PICmicro® proporciona un dispositivo embebido con una interfaz hardware a esta conección RS-232. Cuando se cambia a USB, la aplicación Windows puede ver la conexión USB como un puerto COM virtual mediante el servicio provisto por los dos controladores de Windows, usbser.sys y ccport.sys. Los detalles de los controladores de Windows quedan fuera del alcance de este documento. Un puerto COM virtual provee a las aplicaciones de Windows con la misma interfaz de programación; por lo tanto, la modificación de la aplicación existente del software de PC es innecesaria.

Las áreas que requiren cambios son el hardware y el "firmware" embebidos. Como hardware, se requiere un microcontrollador equipado con un periférico USB de máxima velocidad a bordo para implementar esta solución. La familia de microcontroladores PIC18F2455/2550/4455/4550 será usada aquí como ejemplo. Las referencias a las hojas técnicas de los microcontroladores se encuentran en “PIC18F2455/2550/4455/4550 Data Sheet” (DS39632). Las modificaciones al código del "firmware" de la aplicación existente son mínimos; sólo se requieren pequeños cambios para llamar a las nuevas funciones USB UART que se disponen como parte del marco "firmware" USB, que están escritas en C. La Figura 1 muestra un esquema del método de migración.

Migrar al puerto USB usando el método de emulación serial RS-232 tiene las siguientes ventajas:

• Tiene poco o nulo impacto sobre la aplicación software de PC

• Requiere cambios mínimos al "firmware" de la aplicación existente

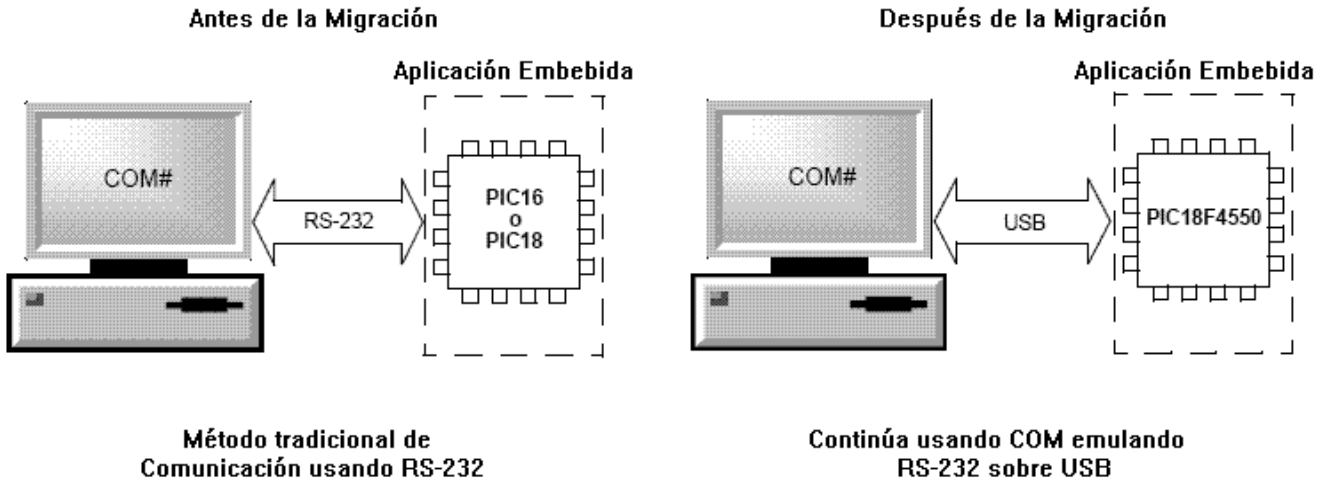
• Acorta el ciclo de tiempo de desarrollo

• Elimina la necesidad de soporte y mantenimiento del controlador de Windows lo que es muy demandante de trabajo.

• Finalmente, el método descrito aquí utiliza un camino de migración claro de muchos dispositivos PIC existentes a la familia de microcontroladores PIC18F2455/2550/4455/4550 haciendo de la actualización a USB un trabajo fácil y directo.

Como el protocolo USB ya maneja los detalles de la comunicación de bajo nivel, el concepto de "baud rate", "bit de paridad" y "control de flujo" del estándard RS-232 se convierten en algo abstracto.

FIGURA 1: COMUNICACIONES SERIALES FUNCIONALMENTE EQUIVALENTES



Especificación CDC USB

Las especificaciones para Clase de Dispositivo de Comunicaciones (CDC) definen varios modelos decomunicación, incluyendo la emulación serial. Todas las referencias a las especificaciones CDC en este documento corresponden a la version 1.1. El controlador de Microsoft Windows, usbser.sys, se ajusta a esta especificación. Por lo tanto, el dispositivo embebido tambien debe ajustarse a esta especificación a fin de utilizar el controlador de Windows ya existente. La especificación CDC describe un modelo de control abstracto para emulación serial sobre USB en la sección 3.6.2.1. En el resumen, se requieren dos interfaces USB. La primera es la interfaz Clase Comunicación, usando un "punto final de interrupción IN". Esta interfaz se usa para notificar al servidor USB del estado de conección actual RS-232 proveniente del dispositivo emulador RS-232 USB. El segundo es la interfaz Clase Datos, usando un "punto final masivo OUT" y un "punto final masivo IN". Esta interfaz se usa para transferir bytes datos tal como podrían ser transferidos normalmente sobre la interfaz real RS-232.

El diseñador no debe preocuparse en crear descriptores o manejadores de funciones para escribir requerimientos específicos de Clase. Los descriptores para un dispositivo emulador RS-232 USB y todos los manejadores requeridos para requerimientos específicos de Clase, listados en la tabla 4 de la especificación CDC, están contenidos en el firmware USB CDC de Microchip. Un diagrama de las estructuras de los descriptores, para un dispositivo emulador RS-232 USB, puede encontrarse en el Apéndice A. Un ejemplo de los descriptores CDC pueden encontrarse también en la Section 5.3 de las especificaciones CDC.

Notificaciones y Requerimientos Específicos de Clase

EL "firmware" USB de Microchip implementa los siguientes manejadores de Requerimientos Específicos de Clase como se lista en la Tabla 4 de la especificación CDC:

- SEND_ENCAPSULATED_COMMAND
- GET_ENCAPSULATED_COMMAND
- SET_LINE_CODING
- GET_LINE_CODING
- SET_CONTROL_LINE_STATE

En las próximas versiones del "firmware" UART USB se proveerá soporte adicional. Actualmente, el "firmware" no devuelve ninguna notificación "RESPONSE_AVAILABLE" pero devuelve en cambio "NAK" para decirle al servidor que no hay ninguna respuesta disponible (No AKnowledge). Para mayores detalles referirse a las secciones 3.6.2.1, 6.2 y 6.3 de las especificaciones CDC.

FIRMWARE USB DE MICROCHIP

FUNCIONES UART USB

Todas las referencias a las funciones UART USB en este documento corresponden a la version 1.0 del controlador de "firmware" del CDC. Este controlador se provee como parte del "firmware" USB de Microchip. Las funciones provistas en esta biblioteca son muy similares en funcionalidad a las de la biblioteca USART MPLAB® C18. Las funciones específicas se listan en la Tabla 1. La elección de qué función emplear depende de varios factores:

• La cadena de datos termina en cero ?

• La longitud de la cadena de datos es conocida ?

• La fuente de los datos está ubicada en la memoria del programa o en RAM ?

TABLA 1: RESUMEN DE LAS FUNCIONES DEL UART USB

Función	**Descripción**
putrsUSBUSART	Escribe una cadena terminada en cero de la memoria del programa al USB.
putsUSBUSART	Escribe una cadena terminada en cero de la memoria de datos al USB.
mUSBUSARTTxRom	Escribe una cadena de longitud conocida de la memoria del programa al USB.
mUSBUSARTTxRam	Escribe una cadena de longitud conocida de la memoria de datos al USB.
mUSBUSARTIsTxTrfReady	Indica si está listo el controlador para aceptar una nueva cadena a escribir en el USB
getsUSBUSART	Lee una cadena del USB.
mCDCGetRxLength	Lee la longitud de la última cadena leida del USB.

---

putrsUSBUSART

putrsUSBUSART escribe una cadena de datos en el USB incluyendo el caracter nulo. Usar esta versión para transferir cadenas de datos constantes o datos ubicados en la memoria del programa, hacia el "*host*".

Nota: El mecanismo de transferencia **put** (*memoria de programa -> host*) es más flexible que el **get** *(host -> memoria de programa*). Puede manejar una cadena de datos más larga que el tamaño máximo del

paquete "bulk-IN-endpoint". Para transferir una cadena larga de datos sobre múltiples transacciones USB se usa una máquina de estados. Este servicio de segundo plano lo realiza *CDCTxService()*.

Sintaxis:

*void putrsUSBUSART(const rom char *data)*

Condición previa:

mUSBUSARTIsTxTrfReady() debe devolver '1' antes que esta función sea invocada. La cadena de caracteres apuntada por *data* debe ser igual o menor que 255 bytes, incluido el caracter cero final.

Entrada:

data

Puntero a una cadena de datos terminada con el caracter cero. Si no se encuentra un caracter cero, solo los primeros 255 bytes de datos serán transferidos hacia el "host".

Salida:

No tiene

Efectos colaterales:

No tiene

Ejemplo:

```
void example_1(void)

{

if(mUSBUSARTIsTxTrfReady())

putrsUSBUSART("Hola Mundo.");

} //end example_1


rom char example_string[] = {"Microchip"};


void example_2(void)
```

```
{
if(mUSBUSARTIsTxTrfReady())
putrsUSBUSART(example_string);
} //end example_2
```

---

putsUSBUSART

putsUSBUSART escribe una cadena de datos al USB incluyendo el caracter nulo finbal. Usar esta versión para transferir datos ubicados en memoria de datos hacia el *host*.

Nota: El mecanismo de transferencia **put** (*memoria de programa -> host*) es más flexible que el **get** *(host -> memoria de programa*). Puede manejar una cadena de datos más larga que el tamaño máximo del paquete "bulk-IN-endpoint". Para transferir una cadena larga de datos sobre múltiples transacciones USB se usa una máquina de estados. Este servicio de segundo plano lo realiza *CDCTxService()*.

Condición previa:

mUSBUSARTIsTxTrfReady() debe devolver '1' antes que esta función pueda ser invocada. La cadena de caracteres apuntada por *data* debe ser igual o menor que 255 bytes, incluido el caracter cero final.

Entrada:

data

Puntero a una cadena de datos terminada con el caracter cero. Si no se encuentra un caracter cero, solo los primeros 255 bytes de datos serán transferidos hacia el *host*.

Salida:

No tiene

Efectos colaterales:

No tiene

Ejemplo:

```
char example_string[4];

void example_1(void)

{

example_string[0]='U';

example_string[1]='S';

example_string[2]='B';

example_string[3]=0x00;

if(mUSBUSARTIsTxTrfReady())

putsUSBUSART(example_string);

}//end example_1
```

---

mUSBUSARTTxRom

Usar esta macro para transferir datos ubicados en la memoria de programa. La longitud de los datos a ser transferidos debe ser conocida y pasarse en un parámetro. La respuesta de esta función estaráindefinida si se la invoca cuando mUSBUSARTIsTxTrfReady() devuelve '0'.

Nota: Esta macro solo maneja la preparación de la transferencia. La transferencia real es manejada por *CDCTxService().*

Sintaxis:

*void mUSBUSARTTxRom(rom byte *pData, byte len)*

Condición previa:

mUSBUSARTIsTxTrfReady() debe devolver '1' antes que esta función pueda ser invocada. El valor de *len* debe ser igual o menor que 255 bytes.

Entrada:

pDdata

Puntero al primer byte de los datos.

len

Número de bytes a ser transferidos.

Salida:

No tiene

Efectos colaterales:

No tiene

Ejemplo:

```
rom char example_string[] = {0x31,0x32,0x33};

void example_1(void)

{

if(mUSBUSARTIsTxTrfReady())

mUSBUSARTTxRom((rom byte*)example_string,3);

}//end example_1
```

---

mUSBUSARTTxRam

Usar esta macro para transferir datos ubicados en la memoria de datos. La longitud de los datos a ser transferidos debe ser conocida y pasada como un parámetro. La respuesta de esta función será indefinida si es invocada cuando *mUSBUSARTIsTxTrfReady()* devuelve '0'.

Nota: Esta macro solo maneja la preparación de la transferencia. La transferencia real es manejada por *CDCTxService().*

Sintaxis:

*void mUSBUSARTTxRam(byte *pData, byte len)*

Condición previa:

mUSBUSARTIsTxTrfReady() debe devolver '1' antes que esta función pueda ser invocada. El valor de *len* debe ser igual o menor que 255 bytes.

Entrada:

pDdata

Puntero al primer byte de los datos.

len

Número de bytes a ser transferidos.

Salida:

No tiene

Efectos colaterales:

No tiene

Ejemplo:

```
char example_string[3];
void example_1(void)
{
example_string[0] = 'U';
example_string[1] = 'S';
example_string[2] = 'B';
if(mUSBUSARTIsTxTrfReady())
mUSBUSARTTxRam((byte*)example_string,3);
}//end example_1
```

mUSBUSARTIsTxTrfReady

Esta macro es usada para averiguar si la clase CDC está lista para enviar más datos.

Uso típico: if*(mUSBUSARTIsTxTrfReady())*

Note: No invocar esta función como una función de bloqueo (i.e., while(! *mUSBUSARTIsTxTrfReady()*);).

Sintaxis:

BOOL mUSBUSARTIsTxTrfReady(void)

Condición previa:

No tiene

Entrada:

No tiene

Salida:

BOOL

Si el controlador de "firmware" está listo para recibir una nueva cadena de datos a escribir en el USB, devolverá ‘1’, de lo contrario devolverá ‘0’.

Efectos colaterales:

No tiene

Ejemplo:

```
void example_1(void)

{

if(mUSBUSARTIsTxTrfReady())

putrsUSBUSART(“Microchip”);

}//end example_1
```

---

getsUSBUSART

getsUSBUSART copia una cadena de bytes recibidos por el CDC USB "bulk-OUT-endpoint" hacia una dirección especificada por el usuario.

Es una función de no-bloqueo. No espera datos si ni hay datos disponibles. En su lugar, devuelve '0' para notificar a la función de llamada que no hay datos disponibles.

Nota: Si el número real de bytes recibidos es mayor que el número de bytes esperados (*len*), sólo el número de bytes esperados especificado será copiado al "buffer". Si el número real de bytes recibidos es menor que el número de bytes esperados (*len*), sólo el número real de bytes recibidos será copiado al "buffer".

Sintaxis:

*byte getsUSBUSART(char *buffer, byte len)*

Condición previa:

Valor del argumento de entrada, len, debe ser igual o menor que el máximo "tamaño de endpoint" responsable por recibir el conjunto "bulk" de datos del *host* USB por la clase CDC. Este valor máximo de "tamaño de endpoint" es definido como CDC_BULK_OUT_EP_SIZE y se encuentra en el archivo usbcfg.h. CDC_BULK_OUT_EP_SIZE puede ser igual a 8, 16, 32 o 64 bytes. El argumento de entrada debe apuntar a un área de "buffer" de tamaño mayor o igual que el especificado por *len*.

Entrada:

buffer

Puntero dónde serán almacenados los bytes recibidos.

len

El número de bytes esperados.

Salida:

byte

El número de bytes copiados al *buffer.*

Efectos colaterales:

Públicamente accesible, la variable *cdc_rx_len* es actualizada con el número de bytes copiados al *buffer*. Una vez que *getsUSBUSART es* invocada, puede recuperarse *cdc_rx_len* llamando a la macro *mCDCGetRxLength()*.

Ejemplo:

```
char input_buffer[64];
void example_1(void)
{
byte index, count, total_sum;
if(getsUSBUSART(input_buffer, 8)
{
count = mCDCGetRxLength();
total_sum = 0;
for(index = 0; index < count; index++)
        total_sum += input_buffer[index];
}//end if
}//end example_1

void example_2(void)
{
if(getsUSBUSART(input_buffer, CDC_BULK_OUT_EP_SIZE)
{
// Do something…
}//end if
}//end example_2
```

---

mCDCGetRxLength

mCDCGetRxLength se usa para averiguar el número de bytes copiados al "*buffer*" del usuario por la última llamada a la función *getsUSBUSART*.

Macro:

byte mCDCGetRxLength(void)

Condición previa:

No tiene

Entrada:

No tiene

Salida:

byte

mCDCGetRxLength devuelve el número de bytes copiados al "*buffer*" del usuario desde la última llamada a la función *getsUSBUSART.*

Efectos colaterales:

No tiene

Ejemplo:

```
char input_buffer[64];

void example_1(void)

{

if(getsUSBUSART(input_buffer, 2)

        {

        // Do something with input_buffer[0]

        if(mCDCGetRxLength() == 2)

        // Do something with input_buffer[1]

        }//end if

}//end example_1
```

Cosas importantes a saber cuando se usan las funciones UART USB

Aunque las funciones provistas UART USB simplifican bastante el uso del puerto USB en una aplicación, hay aún algunas consideraciones para tener en mente cuando se desarrolla o se modifica el código de una aplicación.

DISEÑO DEL CÓDIGO

1. El "firmware" USB de Microchip es un entorno multitarea cooperativo. No debe haber ninguna función de bloqueo en el código del usuario. Como las tareas del USB son consultadas y resueltas en el lazo principal del programa, cualquier función de bloqueo que dependa del estado del puerto USB puede hacer que la aplicación se "cuelgue". Se debe usar una máquina de estados en lugar de una función de bloqueo.

2. *mUSBUSARTTxRom* y *mUSBUSARTTxRam* esperan un puntero de datos de tipo rom *byte** y *byte**, respectivamente. Puede ser necesaria la conversión de tipos de datos.

3. while(!*mUSBUSARTIsTxTrfReady()*); es una función de bloqueo. No la use.

4. *putrsUSBUSART*, *putsUSBUSART*, *mUSBUSARTTxRom* y *mUSBUSARTTxRam* no son funciones de bloqueo. No mandan datos al puerto USB host inmediatamente, ni esperan a que la transmisión se haya completado. Todo lo que hacen es preparar los registros especiales de las funciones necesarias y de las máquinas de estado para la operación de transferencia. La rutina que realmente efectúa la transferencia de datos hacia el host es *CDCTxService()*. Esta le sigue la ruta a las máquinas de estados y descompone las cadenas largas de datos en múltiples paquetes de datos para el puerto USB. Es llamada una vez por cada lazo del programa principal en la rutina de servicio *USBTasks()*. (La máquina de estados para *CDCTxService()* se muestra en el Apéndice B) Debido a esto, las llamadas de funciones "back-to back" no funcionarán; cada nueva llamada sobreescribirá la transacción pendiente. La forma correcta de enviar cadenas consecutivas es usando una máquina de estados, como se muestra en el Ejemplo 1. Una alternativa al uso de máquina de estados es usar un buffer intermediario global, como se muestra en el Ejemplo 2.

5. Debe usarse un conjunto correct de descriptores para la clase CDC. Para ver un ejemplo referirse al proyecto de diseño de referencia.

6. El tamaño del "endpoint" para los canales de datos están definidos por CDC_BULK_OUT_EP_SIZE y CDC_BULK_IN_EP_SIZE, ubicados en el archivo encabezador usbcfg.h. Como estos "endpoints" son de tipo "bulk", el "tamaño máximo de endpoint" descrito debe ser 8, 16, 32 o 64 bytes.

7. El tipo "byte" es definido como un caracter sin signo (unsigned char) en el archivo encabezador typedefs.h.

8. Siempre se debe controlar cuándo el controlador de "firmware" está listo para enviar más datos desde el host USB, invocando a *mUSBUSARTIsTxTrfReady().*

EJEMPLO 1: USO CORRECTO DE LA LLAMADA A LAS FUNCIONES UART USB

Método Incorrecto (llamadas "back-to-back"):

```
if(mUSBUSARTIsTxTrfReady())
{
putrsUSBUSART("Hola Mundo");
putrsUSBUSART("Hola de nuevo");
}//end if
```

Método Correcto (máquina de estados):

```
byte state = 0;
if(state == 0)
{
if(mUSBUSARTIsTxTrfReady())
        {
        putrsUSBUSART(“Hola Mundo”);
        state++;
        }//end if
}
else if(state == 1)
{
```

```
if(mUSBUSARTIsTxTrfReady())
        {
        putrsUSBUSART("Hola de nuevo");
        state++;
        }//end if
}//end if
```

EJEMPLO 2: METODO DE "BUFFER" GLOBAL ALTERNATIVO

```
char io_buffer[64];
. . .
// Lazo de Programa Principal
while(1)
{
USBTasks();
if (mUSBUSARTIsTxTrfReady())
        putsUSBUSART(io_buffer);
// SendToBuffer attachs multiple
// strings together
// (The user will need to provide
// their own SendToBuffer function)
SendToBuffer("Hola Mundo");
SendToBuffer("Hola de nuevo");
} // end while
```

PREPARANDO EL PROYECTO DE CODIGO

1. Insertar #include "system\usb\usb.h" en cada archivo que use las funciones CDC.

2. USB_USE_CDC debe estar definido en el archivo usbcfg.h cuando se usen las funciones CDC.

3. Los archivos encabezador y fuente, cdc.c y cdc.h, deben ser agregados al archivo fuente del proyecto. Todos deben ubicarse en el directorio "system\usb\class\cdc\".

ID de vendedor USB (VID) e ID de Producto (PID)

Los VID y PID son importantes debido a que son usadas por el SO Windows para diferenciar dispositivos USB y para determinar cuál es el controlador que debe ser usado. El VID es un número de 16-bit asignado por el Foro de Implementadores USB (USB-IF). Debe ser obtenido por cada fabricante que desea ofrecer y vender productos USB. El VID puede comprarse directamente en el Foro USB-IF. Puede encontrarse más información en: http://www.usb.org/developers/vendor.

Cada VID viene con 65,536 diferentes PIDs los que también son números de 16-bit. En el entorno de "firmware" USB de Microchip, los VID y PID están ubicados en el archivo usbdsc.c. Ambos valores pueden ser modificados para coincidir con diferentes números de VID y PID.

Controladores para Microsoft Windows® 2000 y Windows® XP

Microsoft Windows no tiene un archivo .inf estándar para the controlador CDC. Los controladores son, sin embargo, parte de la instalación de Windows. La única cosa necesaria es proveer un archivo .inf cuando un dispositivo CDC es conectado inicialmente a un sistema Windows. Archivos .inf de ejemplo se proveen con el proyecto de referencia de emulación RS-232 CDC y están ubicados en el directorio del código fuente <Install>\fw\CDC\inf. Antes de usarlos, deben ser modificados para reflejar los VID y PID específicos de la aplicación. Esto es además de cualquier cambio que se haya hecho a usbdsc.c y que ya debe coincidir con esos valores. Los VID y PID están ubicados en la cadena "USB\VID_xxxx&PIDyyyy", donde "xxxx" es el VID hexadecimal y "yyyy" es el PID hexadecimal. La cadena es en general parte de una de las líneas bajo el encabezador "[DeviceList]". Si lo desean, los usuarios pueden también modificar las definiciones de las variables bajo el encabezador "[Strings]". Esto cambia el texto de la identificacion del dispostivo vista por el usuario en el manejador del dispostivo.

CONCLUSIÓN

La emulación serial RS-232 proporciona un camino de migración fácil para aplicaciones pasando de UART a USB. Del lado de la PC, se requiere una mínima modificación del software. Del lado del dispositivo embebido, la familia de microcontroladores PIC18F2455/2550/4455/4550 proporciona una actualizacion simple del hardware desde las familias de dispositivos PIC16C745/765 y PIC18FXX2. También se incluyen por conveniencia bibliotecas de "firmware" con APIs amigables. El Proyecto de referencia de emulación RS-232 CDC contiene además ejercicios tutoriales.

REFERENCIAS

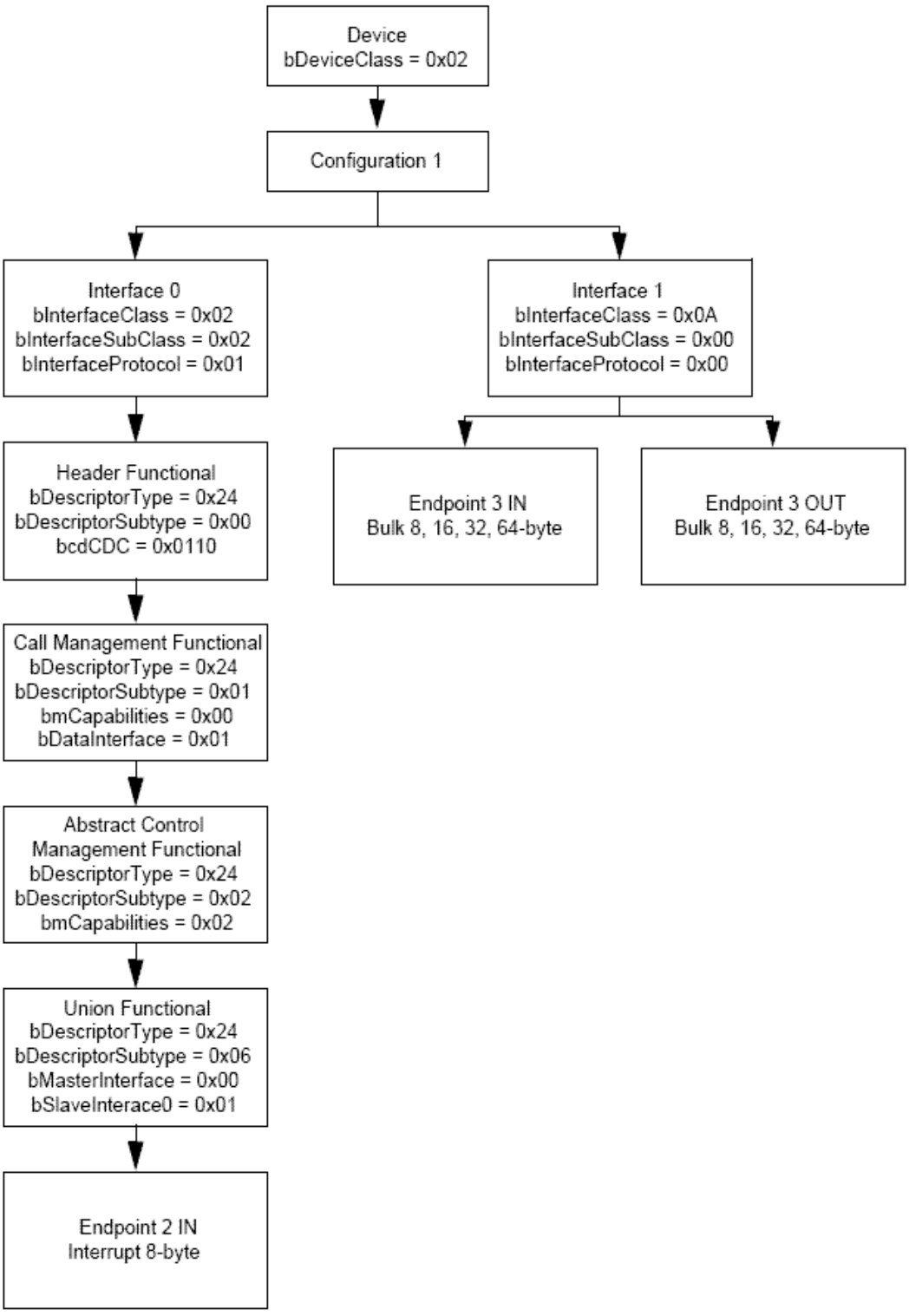
"PIC18F2455/2550/4455/4550 Data Sheet" (DS39632), Microchip Technology Inc., 2004.

"USB Specification Revision 2.0", USB Implementers Forum Inc., 2000.

"USB Class Definitions for Communication Devices Version 1.1", USB Implementers Forum Inc., 1999.

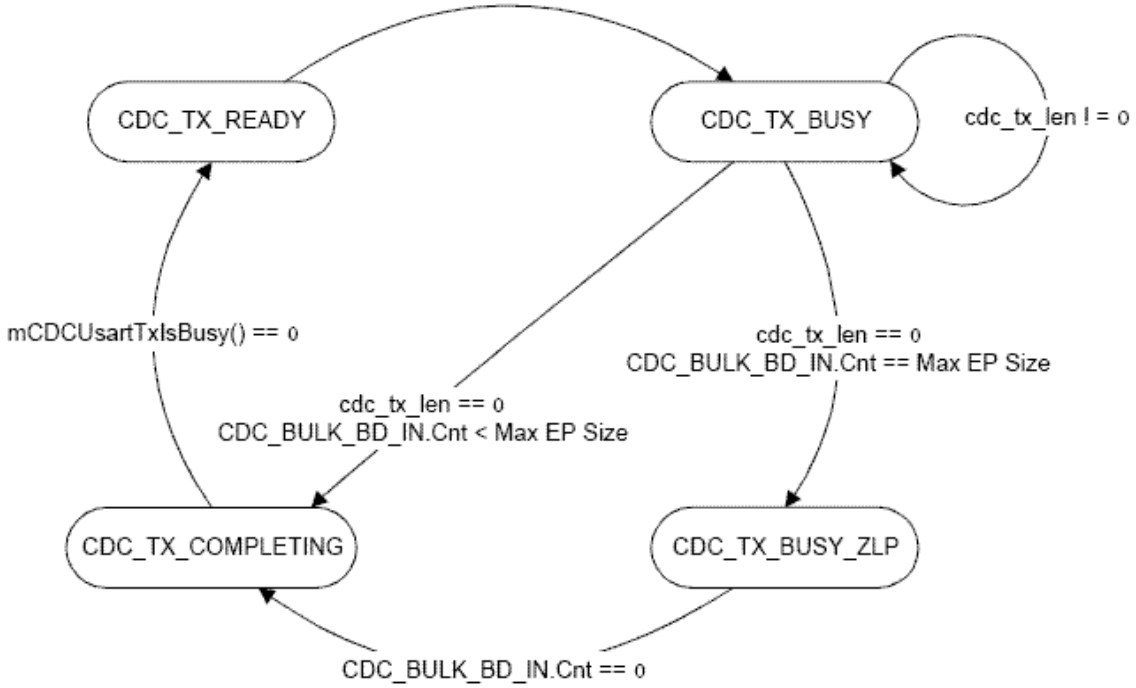
APÉNDICE A: ESQUEMA DESCRIPTOR CDC PARA EMULACIÓN SERIAL

FIGURA A-1: EJEMPLO DE UN CONJUNTO DESCRIPTOR PARA EMULACIÓN SERIAL



APÉNDICE B: MAQUINA DE ESTADOS CDC

FIGURA B-1: DIAGRAMA DE LA MAQUINA DE ESTADOS CDCTxService()



APÉNDICE C: CÓDIGO FUENTE

Debido a su tamaño, el código fuente completo para esta nota de aplicación no se incluye en el texto. Ud. puede descargarlo, incluyendo todos los archivos necesarios y el archivo .inf de Microsoft Windows, desde el web site de Microchip en www.microchip.com

APÉNDICE D: NOTA SOBRE EL TUTORIAL DE EMULACIÓN CDC RS-232

El Proyecto de Referencia de Emulation RS-232 CDC provee una demostración completa de la aplicación discutida en este documento, así como un tutorial para desarrollar aplicaciones. Los ejercicios tutoriales están escritos especialmente para la placa Demo FS USB PICDEM™. Pueden hacerse modificaciones para ejecutar este proyecto sobre una plataforma diferente. El tutorial se incluye como parte del código fuente del mismo. Para más información referirse al archivo user.c en el proyecto. Cuando se usa el programa HyperTerminal en Microsoft Windows, la reconexión física o eléctrica del dispositivo USB causa que el sistema operativo (Windows) asigne un nuevo manejador de controlador para ese dispositivo en particular, interrumpiendo así la comunicación con todas las aplicaciones que tengan su ruta hacia el viejo manejador. En HyperTerminal, colgando la conexión antes de desconectar el dispositivo se permite que el programa re-enumere el puerto COM y por lo tanto lo referencie al manejador del controlador correcto.