Capítulo 6

Marco de trabajo para la programación USB

6.1. Introducción

En el capítulo anterior analizamos paso a paso el kit de desarrollo CY3682, siempre al amparo de las aplicaciones de ejemplo (Panel de Control EZ-USB y SIEMaster) proporcionadas por Cypress Semiconductor. Pero en las aplicaciones reales, el diseñador requerirá poder controlar todos los aspectos de la comunicación USB, adaptando cada parámetro (modo de transferencia, configuración de los endpoints, etc.) según sus necesidades.

Inicialmente, será primordial limitar el problema; esto es, determinaremos qué elementos software han de ser desarrollados, y qué medios disponemos para ello.

La aplicación USB final, construida con el chip CY7C68001 como elemento esclavo de un procesador externo principal, contendrá (vea la figura 6.1):

- Un driver USB de Windows o driver de clase incluido con el sistema operativo.
- Los drivers estándares de Windows.
- Firmware para la aplicación del procesador host y (opcional) un programa de aplicación para Windows USB específico.

Trataremos a continuación los principales elementos de este sistema, dejando para el apartado 6.6 un ejemplo de implementación de una sencilla comunicación USB 2.0 tipo bulk con el chip EZ-USB SX2.

6.2. Programación del driver

Teniendo en cuenta que el kit incluye el driver de dispositivo de propósito general EZ-USB (*GPD, General Purpose Driver*), **no será necesario la programación de**

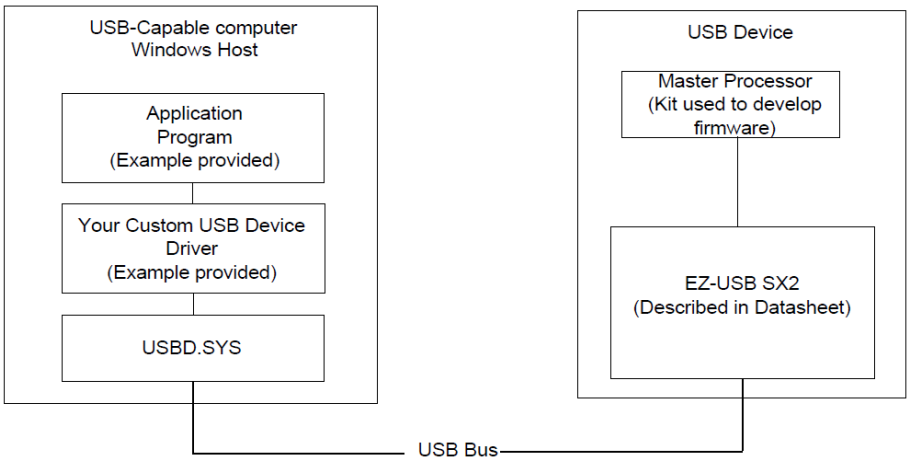


Figura 6.1: Diagrama de bloques del sistema EZ-USB final.

ningún otro driver, a menos que se desee implementar un driver personalizado o un mini-driver, en cuyo caso el GPD servirá como punto de partida.

La programación de drivers excede las pretensiones de este Proyecto, si bien el autor tuvo la oportunidad de estudiar las posibilidades para Windows:

- WDM (*Windows Driver Model*) y
- WDF (*Windows Driver Foundation*).

El primer modelo (WDM), y más antiguo, trabaja exclusivamente en modo núcleo, con lo cual cualquier error de programación puede dar lugar a un "cuelgue" del sistema. El modelo más actual (WDF), por su parte, permite trabajar en modo núcleo (utilizando el KMDF, *Kernel Mode Device Framework*) o en modo usuario (utilizando el UMDF, *User Mode Driver Framework*), siendo éste último la forma más segura y sencilla (relativamente) de generar drivers.

El estudio del modelo WDF puede ser realmente arduo, máxime si se tiene en cuenta que actualmente sólo se dispone documentación a través de la web de Microsoft, donde es fácil comenzar en un punto de partida, y terminar en otro absolutamente distinto. Por ello, para el lector interesado, se ha estructurado jerárquicamente las principales páginas de interés de la web de Microsoft WDK (*Windows Driver Kit*):

1. Driver Fundamentals: Getting Started,
   `http://www.microsoft.com/whdc/driver/foundation/default.mspx`

   a) Windows Roadmap for Driver,
   `http://www.microsoft.com/whdc/driver/foundation/DrvRoadmap.mspx#`

   b) WHDC (Windows Hardware Developers Central) Technical References for Driver Development,
   `http://www.microsoft.com/whdc/resources/respec/TechRef.mspx#`

c) Russinovich, Mark E., and Solomon, David A. Microsoft Windows Internals, Fourth Edition, http://www.microsoft.com/MSPress/books/6710.asp

d) Architecture of Microsoft Windows Driver Foundation, http://www.microsoft.com/whdc/driver/wdf/wdf-arch.mspx#

2. About the Windows Driver Kit (WDK)

a) Windows Driver Foundation (WDF), http://www.microsoft.com/whdc/driver/wdf/default.mspx

1) Introduction to the Windows Driver Foundation, http://www.microsoft.com/whdc/driver/wdf/wdf-intro.mspx
2) Windows Driver Foundation Facts, http://www.microsoft.com/whdc/driver/wdf/WDF_facts.mspx
3) FAQ: Questions from Driver Developers about Windows Driver Foundation, http://www.microsoft.com/whdc/driver/wdf/WDF_FAQ.mspx

b) Kernel-Mode Driver Framework (KMDF), http://www.microsoft.com/whdc/driver/wdf/KMDF.mspx

1) Architecture of the Kernel-Mode Driver Framework, http://www.microsoft.com/whdc/driver/wdf/kmdf-arch.mspx
2) Sample Drivers for the Kernel-Mode Driver Framework, http://www.microsoft.com/whdc/driver/wdf/KMDF-samp.mspx
3) How to Build, Install, Test, and Debug a KMDF Driver, http://www.microsoft.com/whdc/driver/wdf/KMDF-build.mspx
4) Introduction to Plug and Play and Power Management in the Windows Driver Foundation, http://www.microsoft.com/whdc/driver/wdf/WDF_pnpPower.mspx
5) DMA Support in Windows Drivers
6) I/O Request Flow in WDF Kernel Mode Drivers
7) How to Enable the Frameworks Verifier
8) How to Use the KMDF Log
9) Is That Handle Still Valid?
10) Troubleshooting KMDF Driver Installation
11) When does EvtCleanupCallback run?

c) User-Mode Driver Framework (UMDF), http://www.microsoft.com/whdc/driver/wdf/UMDF.mspx

1) Introduction to the WDF User-Mode Driver Framework, http://www.microsoft.com/whdc/driver/wdf/UMDF_intro.mspx

2) Architecture of the User-Mode Driver Framework,
http://www.microsoft.com/whdc/driver/wdf/UMDF-arch.mspx

3) FAQ: User-Mode Driver Framework,
http://www.microsoft.com/whdc/driver/wdf/UMDF_FAQ.mspx

4) Sample Drivers for the User-Mode Driver Framework,
http://www.microsoft.com/whdc/driver/wdf/UMDF-samp.mspx

d) Header file refactoring,
http://www.microsoft.com/whdc/driver/WDK/headers.mspx

e) PREfast,
http://www.microsoft.com/whdc/DevTools/tools/PREfast.mspx

1) PREfast Step-by-Step,
http://www.microsoft.com/whdc/DevTools/tools/PREfast_steps.mspx

f) Static Driver Verifier,
http://www.microsoft.com/whdc/devtools/tools/SDV.mspx

g) Debugging Tools for Windows,
http://www.microsoft.com/whdc/devtools/debugging/default.mspx

h) Windows Logo Testing,
http://www.microsoft.com/whdc/GetStart/testing.mspx

i) Driver Install Frameworks Tools 2.01,
http://www.microsoft.com/whdc/driver/install/DIFxtls.mspx

j) Static Driver Verifier Facts,
http://www.microsoft.com/whdc/devtools/tools/sdv_facts.mspx

No obstante, es altamente recomendable leer previamente el libro *Microsoft Windows Internal*, de Mark E. Russinovich y David A. Solomon, en su cuarta edición, con el cual el diseñador podrá entender todos los conceptos que se mencionan tanto en las páginas web anteriores como en los documentos del WDK, descargados desde dichas páginas y almacenados en el CD-ROM.

Es posible, por otro lado, descargarse numerosas herramientas que pueden servir de ayuda (gran parte de ellas puede encontrarlas en el CD-ROM adjunto).

6.3. Programación del firmware para el procesador principal

Como se puntualizaba en el apartado de "Posibles ampliaciones", la programación del software de control del procesador específico (microprocesadores estándar, DSP, ASIC, FPGA, etc.) no pertenece al alcance de este Proyecto. Será el desarrollador quien, en función del tipo de procesador, escoja un lenguaje de programación,

que se pueda compilar en un formato comprensible para el microprocesador, y un entorno de programación apropiado.

6.4. Alternativas para la codificación de la comunicación USB entre la aplicación host y el dispositivo USB 2.0

Realmente este es el aspecto más importante de los tratados en el Proyecto que nos ocupa, pues permite demostrar el funcionamiento del CI CY7C68001 (la placa EZ-USB SX2), comunicándose con una aplicación host. Se entiende por *aplicación host* el software situado en un PC que requiere la transferencia de información, a través del USB, hacia o desde el sistema electrónico diseñado.

De hecho, lo que mostraremos en los próximos párrafos será el marco de trabajo de la programación entre la aplicación host y el dispositivo USB. En cualquier caso, no se exige ningún lenguaje de programación determinado, si bien, los ejemplos que se aportarán estarán codificados en C++.

Grosso modo, podemos encontrar dos posibilidades para comunicarnos con el dispositivo USB, ambas a través del driver correspondiente:

- Mediante funciones Win32 (API de Windows) de bajo nivel, o
- Mediante la CyAPI, hallada en el *USB Developer's uStudio* (CY4604).

6.4.1. Comunicación USB a través del API de Windows

Comencemos por la primera opción. Como ya se comentó anteriormente, el kit de desarrollo CY3682 incluye un driver de propósito general EZ-USB (GPD, *General Purpose Driver*), que presenta una interfaz en modo usuario a la que puede acceder a través de las funciones de Win32 `CreateFile()` y `DeviceIoControl()`. Las distintas peticiones al dispositivo USB definidas en el capítulo 9 de la Especificación USB son manejadas mediante códigos de control de entrada/salida, también denominados, IOCTLs. Serán necesarias además distintas estructuras que servirán de parámetros de la función `DeviceIoControl()`.

El kit de desarrollo incluye numerosas aplicaciones de ejemplo (como la que puede encontrar en `C:\Cypress\USB\Examples\EzUsb\bulktest\host`) que pueden servir de punto de partida para un diseño que se comunique directamente con el GPD a través de la API de Windows.

Podrá encontrar todos los detalles de la programación mediante el GPD en el documento "EZ-USB General Purpose Driver Specification", impreso en el anexo de Manuales.

Los principales inconvenientes de este tipo de programación son:

1. Se trata de un funcionamiento a bajo nivel, es decir, se requiere un control absoluto de los parámetros proporcionados (estructuras, tipos de datos, ...), y deben manejarse con soltura los tipos de datos de Windows, junto con su notación (húngara).

2. Se precisa una programación mediante hilos (*threads*) para que la transferencia (transmisión/recepción) de información a través del USB no detenga la ejecución de la aplicación host.

En conclusión, para aplicaciones host sencillas, la programación directa a través del GPD puede no suponer ningún problema, pero en otros casos puede resultar realmente engorrosa.

6.4.2. Comunicación USB a través del API de Cypress

Precisamente para solventar los inconvenientes mencionados, Cypress desarrolló en 2003 el *USB Developer's uStudio* (CY4604) que incluía los siguientes elementos:

- Un driver USB genérico, desarrollado siguiendo el WDM (Windows Driver Model) y compatible con Windows 2000 y Windows XP. Incluye además soporte para Plug and Play (PnP), despertado remoto (*remote wake-up*), identificador único global (GUID) personalizable, y gestión de potencia de nivel S4. El driver puede ser usado para aplicaciones de propósito general que usen transferencias de control, *interrupt*, *bulk* o isócronas.

- Una versión mejorada del Panel de Control EZ-USB: CyConsole, que incluye características mejoradas para permitir que los desarrolladores puedan emular mejor la aplicación host USB, respuestas, y test para ajustar el firmware y la interfaz con el driver del dispositivo.

- Una librería de clase compatible con Visual C++ y Borland C++Builder: CyAPI, que proporciona una interfaz de programación de aplicación (API) con el driver genérico de dispositivo USB de Cypress.

Este kit de referencia, nos permite una programación más sencilla de la comunicación USB entre la aplicación host y el dispositivo USB, a través de la librería CyAPI. Nuestro objetivo será, pues, desarrollar una aplicación práctica donde se expongan las instrucciones necesarias con las que realizar una comunicación *bulk.*

Antes de proceder, sólo nos queda un comentario más: aunque este kit incluye un driver USB genérico, con todos los códigos de control IOCTL claramente documentados en el "Cypress CyUsb.sys Programmer's Reference", se pretende trabajar a un nivel superior, dejando a un lado el contacto directo con el API de Windows.

Comenzaremos instalando el estudio de desarrollo.

6.5. Instalando el estudio de desarrollo USB CY4604

La instalación no presenta ninguna dificultad. Ejecutaremos el archivo `USBDevStudio_1511.exe` y seguiremos los pasos hasta completar el proceso. Podremos cerciorarnos del éxito de la instalación comprobando que se han añadido los siguientes elementos a Inicio\Cypress\USB:

- CyConsole,
- Programmer's Reference - CyUSB.sys, y
- Programmer's Reference - CyAPI.lib.

Se recomienda la creación del acceso directo de CyConsole en el escritorio.

6.6. Usando la API de Cypress—CyAPI

Consideremos el siguiente supuesto:

> Disponemos de un dispositivo controlado por un procesador externo principal que requiere la comunicación con una aplicación host a través del bus serie universal. Se ha conectado adecuadamente el procesador externo con la placa EZ-USB SX2 (el pinout aparece en el manual de características técnicas), que actuará de interfaz con la aplicación host.

Se requiere el cumplimiento de los siguientes puntos:

- El dispositivo USB debe ser reconocido por el sistema operativo (Windows XP, en nuestro caso), como un "Dispositivo USB de prueba del PFC".
- El fabricante del dispositivo será "Cypress Semiconductor", y el suministrador, el "Departamento de Ingeniería Electrónica".
- Deberá cargarse el driver `CyUSB.sys`, ya que posiblemente Windows utilice el `ezusb.sys` (instalado y configurado con el kit de desarrollo CY3682).
- Deberemos ser capaces de transmitir información en modo *bulk* y verificar la correcta recepción de los datos transmitidos.

Resolveremos este problema secuencialmente a lo largo de múltiples ejercicios. Se considera que, previamente, hemos instalado el estudio de desarrollo USB CY4604, y que, para facilitar el proceso, partimos del estado final del Ejercicio 3:

- La placa EZ-USB SX2 conectada a la FX,
- Un único cable USB conectado entre la SX2 y el PC,

- El firmware `xmaster.hex` cargado en la RAM de la FX, y
- El identificador de dispositivo "Cypress EZ-USB Sample Device" correspondiente a unos valores VID\PID de 0x04B4\0x1002 en el Administrador de Dispositivos de Windows.

6.6.1. Ejercicio 12—Adición del identificador del dispositivo al driver

Para que el driver detecte un dispositivo específico, los identificadores VID y PID deberán ser añadidos al fichero `C:\Archivos de programa\Cypress\USB DevStudio\Driver\CyUSB.inf`.

Siga los siguientes pasos:

1. Localice la sección `[Cypress]` y duplique la línea

   ```
   ;%VID_VVVV&PID_PPPP.DeviceDesc%=CyUsb, USB\VID_VVVV&PID_PPPP
   ```

   eliminando el punto y coma de la línea duplicada.

2. Cambie `VVVV` por "04B4" (correspondiente a 0x04B4), que es el valor hexadecimal del identificador de vendedor (VID) de la SX2 con el descriptor de prueba cargado por el firmware xmaster, y `PPPP` por "1002" (correspondiente a 0x1002), valor hexadecimal del identificador de producto (PID) de la SX2. El resultado será el siguiente:

   ```
   %VID_04B4&PID_1002.DeviceDesc%=CyUsb, USB\VID_04B4&PID_1002
   ```

3. Localice la sección `[Strings]` al final del archivo `CyUSB.inf`, y duplique la línea

   ```
   VID_VVVV&PID_PPPP.DeviceDesc="Cypress Generic USB Device"
   ```

4. Cambie `VVVV` por "04B4", `PPPP` por "1002" y "Cypress Generic USB Device" por "Dispositivo USB de prueba del PFC", con lo que tendría que quedar:

   ```
   VID_04B4&PID_1002.DeviceDesc="Dispositivo USB de prueba del PFC"
   ```

5. Guarde el archivo y ciérrelo.

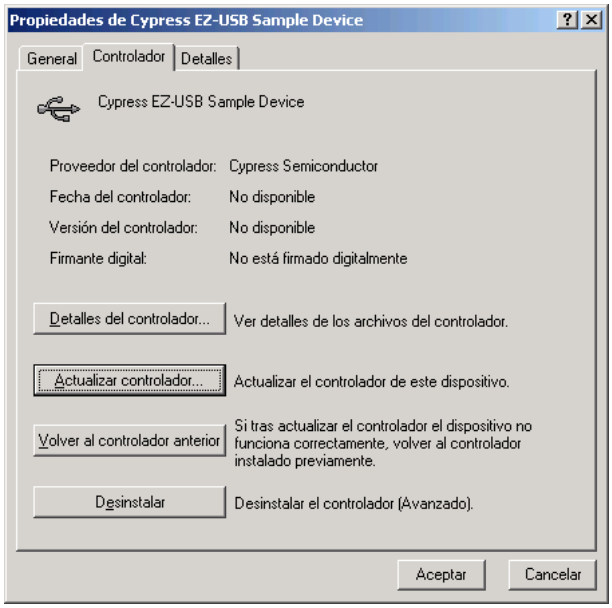


Figura 6.2: Cambiando el controlador de la EZ-USB SX2.

6.6.2. Ejercicio 13—Sustitución de las cadenas de texto de fabricante y suministrador

1. Abra de nuevo el archivo `C:\Archivos de programa\Cypress\USB DevStudio\Driver\CyUSB.inf`.

2. Sustituya `MfgName="Cypress"` por `MfgName="Cypress Semiconductor"`.

3. Sustituya `provider=%CYPRESS%` por `provider=”Escuela Técnica Superior de Ingenieros”`.

4. Sustituya `CyUsb.SvcDesc="Cypress Generic USB Driver"` por

```
CyUsb.SvcDesc="Dispositivo USB de prueba del PFC
```

6.6.3. Ejercicio 14—Forzando el uso del driver CyUSB.sys

Durante el capítulo 4 configuramos el kit de desarrollo CY3682, que instalaba y configuraba el driver de propósito general `ezusb.sys`, y sobre el que se sustentaba toda la funcionalidad que tuvimos oportunidad de comprobar. Lo que haremos en este ejercicio será forzar que Windows XP cargue el driver `CyUSB.sys`, pues la librería CyAPI se comunica únicamente con dicho driver.

1. Abra el Administrador de Dispositivos de Windows, localice el dispositivo “Cypress EZ-USB Sample Device” dentro del nodo “Controladoras de bus serie universal (USB)” y haga doble clic en su icono.

2. Seleccione la pestaña controlador (vea la figura 6.2) y pulse en “Actualizar controlador”.

3. Seleccione “No por el momento” y pulse en “Siguiente”.
4. Seleccione “Instalar desde una lista o ubicación específica (avanzado)” y pulse en “Siguiente”.
5. Seleccione “No buscar. Seleccionaré el controlador que se va a instalar” y pulse en “Siguiente”.
6. Pulse en “Utilizar disco”, luego en “Examinar”, seleccione el archivo `C:\Archivos de programa\Cypress\USB DevStudio\Driver\CyUSB.inf`, y pulse en “Abrir”. Pulse “Aceptar”
7. Pulse en “Siguiente” y autorice la instalación.
8. Pulse en “Finalizar” y luego en “Cerrar”.

Podrá comprobar que, si se siguieron los pasos como se ha indicado, ahora disponemos de un “Dispositivo USB de prueba del PFC” (vea la figura 6.3).

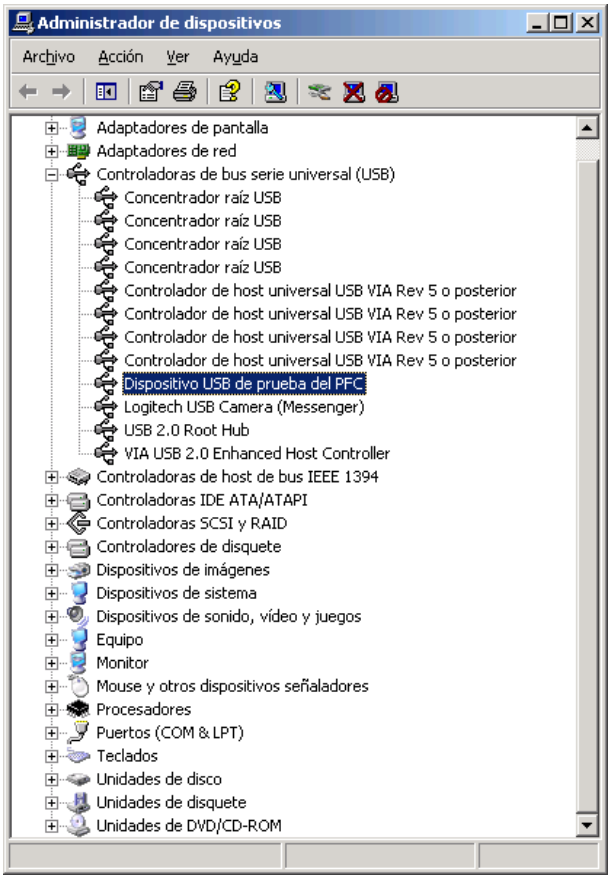


Figura 6.3: Verificando la correcta instalación del driver `CyUSB.sys`.

6.6.4. Ejercicio 15—Aplicación host básica para la comunicación USB 2.0 de tipo bulk con el chip EZ-USB SX2

Considerando que el firmware `xmaster` provoca que los datos que la FX lea del endpoint tipo bulk de salida (desde el PC) serán reenviados al endpoint tipo bulk

de entrada (hacia el PC), implementaremos una comunicación USB 2.0 entre una aplicación host y el chip EZ-USB SX2. Hemos de tener en cuenta que en la tasa de transferencia media afectará el tiempo de procesado de la FX.

Crearemos una aplicación de consola *dummy* que establecerá, iniciará y finalizará la comunicación con el chip EZ-USB SX2 a través de CyAPI. Hay que recordar, que es necesario enlazar `CyAPI.lib` con el proyecto en el entorno de desarrollo empleado (Borland C++Builder 6, en nuestro caso).

Seguidamente proporcionamos el contenido de `UnidadBasica.cpp` (que, lógicamente, se incluye en el CD anexo junto con el archivo de proyecto `.bpr`). Entre paréntesis se numeran los comentarios que aparecerán posteriormente.

```
#include <windows.h>
#include "CyAPI.h"

void main() {
   const int packetSize = 512; // (1)
   LONG pckSize = packetSize;  // (2)
   char outBuffer[packetSize]; // (3)
   char inBuffer[packetSize];
   int devices, vID, pID, d = 0;

   CCyUSBDevice *USBDevice = new CCyUSBDevice(); // (4)
   devices = USBDevice->DeviceCount();

   do { // (5)
      USBDevice->Open(d);
      vID = USBDevice->VendorID;
      pID = USBDevice->ProductID;
      d++;
   } while ((d < devices) && (vID != 0x04b4) && (pID != 0x1002));

   for (int i=0; i<packetSize; i++)
        outBuffer[i] = i;

   USBDevice->BulkOutEndPt->XferData(outBuffer, pckSize); // (7)
   USBDevice->BulkInEndPt->XferData(inBuffer, pckSize);   // (8)

   USBDevice->Close(); // (9)
}
```

Comentarios:

1. El firmware `xmaster` configura los endpoint tipo bulk para admitir un tamaño máximo de paquete de 512 bytes. Se comprueba que para que el proceso de transferencia de datos sea satisfactorio, todos los envíos deben realizarse en paquetes de 512 bytes.

2. La función `XferData` requiere que el tamaño de paquete sea de tipo `LONG`.

3. Los datos se almacenarán en arrays de cadenas de 512 bytes.

4. Se hace una llamada al driver CyUSB para obtener un manejador (*handle*) a los dispositivos USB de Cypress conectados. Con el depurador de C++Builder

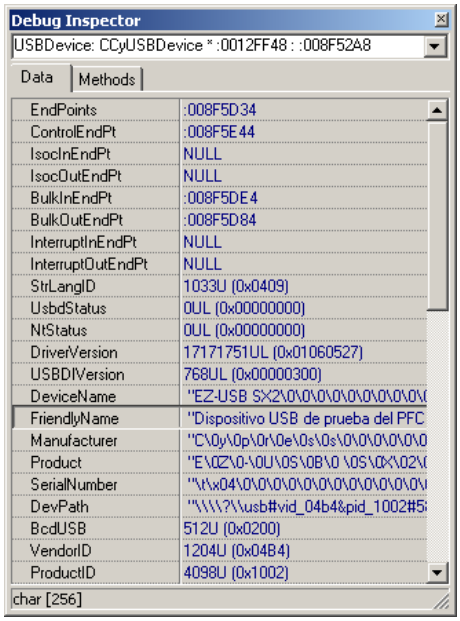


Figura 6.4: Valores de las propiedades del objeto `USBDevice` al obtener el manejador desde el driver.

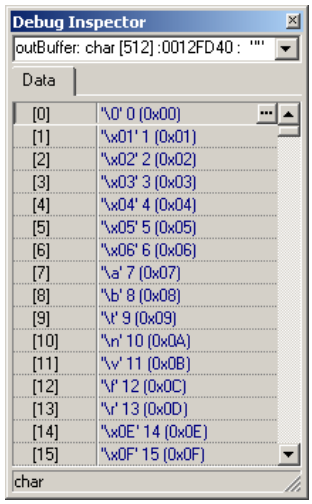


Figura 6.5: Buffer de salida `outBuffer` con una secuencia de 512 valores.

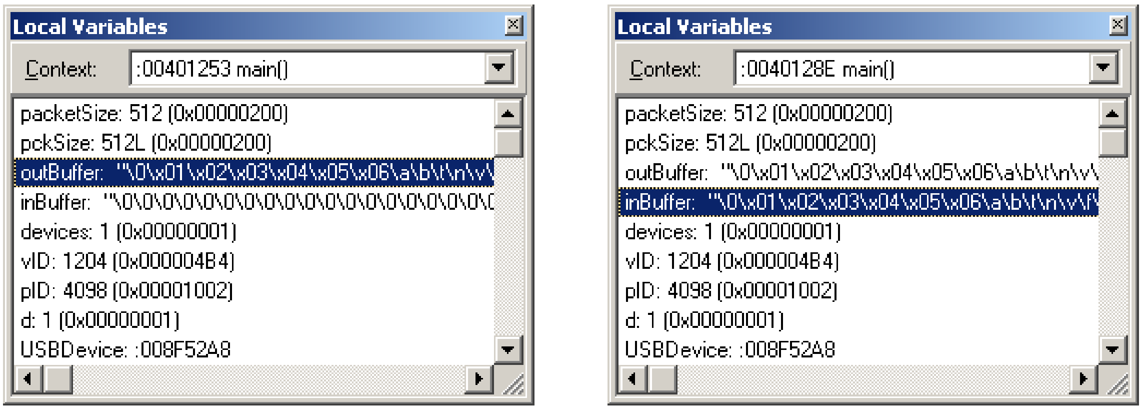
podemos ver la ventana de la figura 6.4, donde se aprecian los valores de las distintas propiedades del objeto `USBDevice`.

5. En el bucle `do-while` se busca entre los dispositivos USB de Cypress, el que tenga vID igual a 0x04b4h y pID igual a 0x1002h, es decir, busca el chip EZ-USB SX2.

6. Almacenamos la secuencia $0, 1, 2, 3, \ldots, 255, 0, 1, 2, 3, \ldots, 255$ en el buffer de salida (se trata de una variable *signed*), como aparece en la figura 6.5.

7. Se transfieren de forma síncrona los datos del buffer de salida de tipo cadena al primer endpoint de tipo bulk de salida (desde el PC) con la función `XferData`.

8. Al finalizar el envío, se solicita la recepción síncrona de información del endpoint de tipo bulk de entrada (hacia el PC) de nuevo con la función `XferData`.



(a) (b)

Figura 6.6: (*a*) Ventana de variables locales en modo depuración antes de iniciar la transferencia al endpoint de salida y (*b*) después de recibir los datos desde el endpoint de entrada.

Teniendo en cuenta que, por simplicidad, no hemos mostrado información en pantalla del proceso de comunicación, nos remitimos a la ventana de variables locales en modo depuración (figura 6.6).

9. Una vez acabado el proceso de comunicación, liberamos el manejador del dispositivo USB de Cypress.

6.6.5. Ejercicio 16—Aplicación host práctica para la comunicación USB 2.0 de tipo bulk con el chip EZ-USB SX2

Después de haber trazado en el ejercicio anterior las líneas básicas de la programación de una comunicación USB 2.0 tipo bulk con el chip EZ-USB SX2, haciendo uso de la CyAPI, trataremos a continuación de diseñar una aplicación que envíe un archivo BMP determinado a la placa, lo obtenga de vuelta, y lo muestre por pantalla.

De nuevo, el firmware `xmaster` nos servirá de medio para disponer de retorno los datos que enviamos. Sin embargo, en esta ocasión diseñaremos una aplicación VCL, consistente en un formulario con un botón para transferir la imagen de prueba, una barra de progreso y distintas etiquetas.

El algoritmo de comunicación estará implementado en el método `OnClick` del botón de transferencia. Básicamente se encarga de abrir el archivo de datos origen (una imagen BMP), hacerlo trozos de 512 bytes, enviarlo, recibirlo, almacenarlo en otro archivo de datos destino, y finalmente representar gráficamente su información (mostrar la imagen que contiene) y datos estadísticos de la transferencia.

Mostramos seguidamente las líneas de código de `UnidadPrincipal.cpp`. (Al igual que el archivo anterior, y sus respectivos archivos de proyecto, los podrá encontrar en el CD anexo.)

```
//---------------------------------------------------------------------------
// PROYECTO FINAL DE CARRERA
// Evaluación del chip EZ-USB SX2
//
// Autor: Alejandro Raigón Muñoz
// Tutor: Dr. Jonathan Noel Tombs
//
// Mayo-2007
//---------------------------------------------------------------------------
//      DESCRIPCIÓN: Esta aplicación trata de demostrar el proceso de
//      transferencia USB 2.0 tipo bulk utilizando el chip EZ-USB SX2. Para ello
//      tomamos una imagen BMP, la enviamos a la placa EZ-USB SX2, y la
//      recibimos de la misma a través de endpoints bulk. Posteriormente
//      se muestra el resultado obtenido, así como estadísticas de la
//      transferencia.
//
//      Se supone que la EEPROM de la EZ-USB FX tiene almacenado el firmware
//      xmaster, y que la conexión de las placas FX y SX2 es la descrita
//      en el ejercicio 3 de la Memoria. Asimismo, se supone que la SX2 está
//      correctamente conectada al PC, y que ha sido detectada e identificada
//      satisfactoriamente.
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "UnidadPrincipal.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
#include "CyAPI.h"
#include <time.h>

TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        // Abrimos la imagen que se enviará por puerto USB a la EZ-USB SX2
        TFileStream *source = new TFileStream("source.bmp", fmOpenRead);
        // Creamos el fichero destino
        TFileStream *target = new TFileStream("target.bmp", fmCreate);

        // Los paquetes se enviarán en trozos de 512 bytes
        const int packetSize = 512;
        // La rutina de transferencia USB (XferData) requiere un tipo LONG
        LONG pckSize = packetSize;
        // Los buffer de salida y entrada serán de tipo "char"
        char outBuffer[packetSize];
        char inBuffer[packetSize];
        int devices, vID, pID, d = 0;
        // En el bitmap cargaremos la imagen recibida
        Graphics::TBitmap *bitmap = new Graphics::TBitmap;
        // que será pintada con unas dimensiones de 800x600
        TRect destArea = TRect((ClientWidth-800)/2,45,
                               (ClientWidth+800)/2,600+45);

        int temp;                       // Variable auxiliar
        double tmp;                     // Variable auxiliar
```

```
        AnsiString stringAux;           // Variable auxiliar

        // Limpiamos el área de dibujo inicial
        Canvas->FillRect(destArea);

        // Abrimos el dispositivo USB de Cypress
CCyUSBDevice *USBDevice = new CCyUSBDevice(Handle);
devices = USBDevice->DeviceCount();

        // Buscamos la placa EZ-USB SX2
        do {
USBDevice->Open(d);
vID = USBDevice->VendorID;
pID = USBDevice->ProductID;
d++;
} while ((d < devices) && (vID != 0x04b4) && (pID != 0x1002));

        // Calculamos los trozos de 512 bytes que serán enviados
        temp = source->Size/packetSize;
        ProgressBar1->Visible = true;
        ProgressBar1->Max = temp+1;

        // Almacenamos el valor temporal al inicio del proceso de transferencia
        clock_t t1 = clock();

        USBDevice->BulkOutEndPt->TimeOut = 500;
        source->Read(outBuffer, packetSize);

        // Enviamos los paquetes de 512 bytes
        for(int i = 0; i < temp; i++) {
                // Leemos un paquete desde el archivo fuente
                source->Read(outBuffer, packetSize);
                // Lo transferimos al primer endpoint tipo bulk de salida
                USBDevice->BulkOutEndPt->XferData(outBuffer, pckSize);
                // Lo recuperamos del primer endpoint tipo bulk de entrada
                USBDevice->BulkInEndPt->XferData(inBuffer, pckSize);
                // Y lo almacenamos en el archivo destino
                target->Write(inBuffer, packetSize);
                // Incrementando la barra de progreso consecuentemente
                ProgressBar1->Position = i;
        }

        // Si queda un paquete de menos de 512 bytes será enviado seguidamente
        temp = source->Size % packetSize;

        if (temp) {
                source->Read(outBuffer, temp);
                USBDevice->BulkOutEndPt->XferData(outBuffer, pckSize);
                USBDevice->BulkInEndPt->XferData(inBuffer, pckSize);
                target->Write(inBuffer, temp);
        }

        // NOTA: Aunque el tamaño de paquete del último trozo debería ser
        // inferior a "packetSize", si no se transfieren paquetes de 512 bytes
        // se ha observado que el firmware "xmaster" no opera correctamente.
        // No obstante, no se escriben nada más que los "temp" primeros bytes en
        // el archivo destino.

        // Almacenamos el valor temporal al finalizar el proceso de transferencia
        clock_t t2 = clock();

        delete target;  // Liberamos el handle del archivo destino,
```

```
        bitmap->LoadFromFile("target.bmp"); // lo cargamos en el bitmap
        Canvas->StretchDraw(destArea, bitmap);  // y lo mostramos en pantalla

        // A continuación mostraremos cierta información de la transferencia
        Canvas->Font->Color = clRed;
        Canvas->Font->Size = 12;
        Canvas->Font->Style = TFontStyles()<< fsBold;

        tmp = (double) 2*source->Size/1024;
        stringAux = "Kilobytes transferidos: " + FormatFloat("0.0",tmp) + " kB";
        Canvas->TextOut(120,70, stringAux);

        tmp = (t2-t1)/CLK_TCK;
        stringAux = "Tiempo empleado: " + FormatFloat("0.0",tmp) + " segundos";
        Canvas->TextOut(120,90, stringAux);

        tmp = (2*source->Size/1024)/tmp;
        stringAux = "Throughput: " + FormatFloat("0.0",tmp) + " kB/s";
        Canvas->TextOut(120,110, stringAux);

        // Téngase en cuenta que en el throughput intervienen distintos
        // factores, a saber: tasa de transferencia hacia y desde la
        // EZ-USB SX2 y tiempo de procesado de la EZ-USB FX

        ProgressBar1->Visible = false;
        USBDevice->Close();     // Cerramos el dispositivo,
        delete bitmap;          // liberamos el handle del bitmap
        delete source;          // y el del archivo fuente
}
```

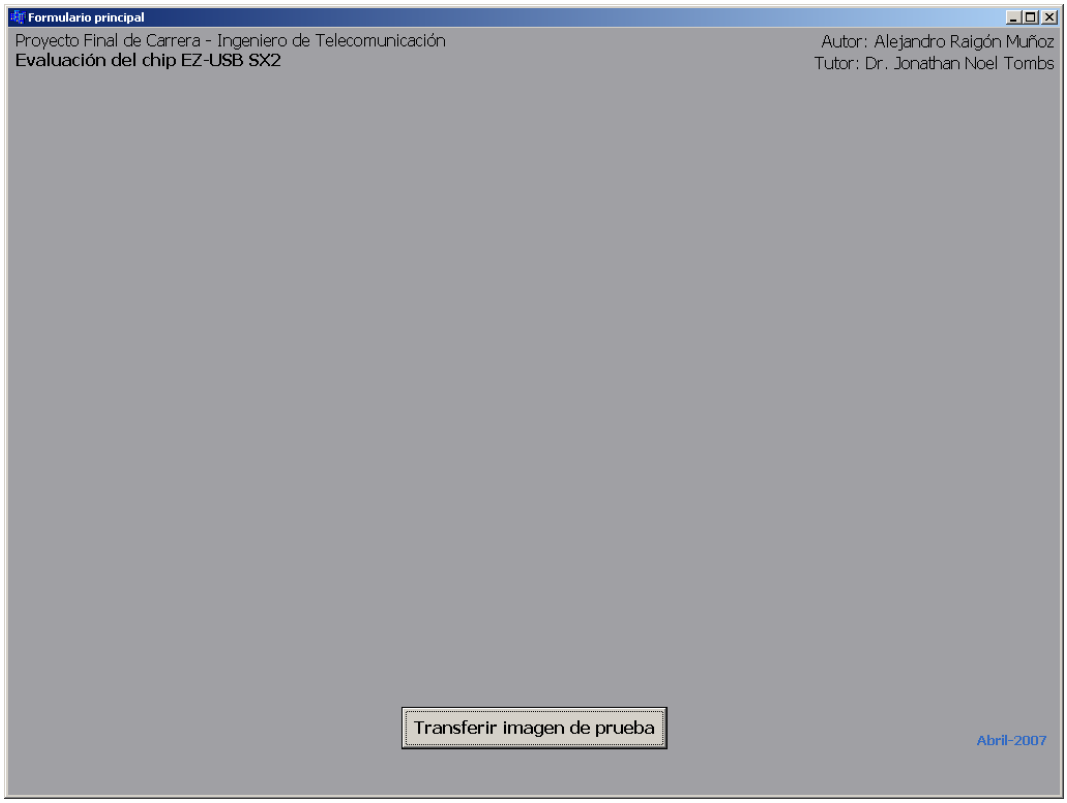


Figura 6.7: Formulario principal de la aplicación práctica de una comunicación USB 2.0 tipo bulk con el chip EZ-USB SX2.

El formulario principal inicialmente tendrá el aspecto mostrado en la figura 6.7, y, después de finalizar la transferencia de forma satisfactoria, podremos visualizar la imagen recogida en la figura 6.8.

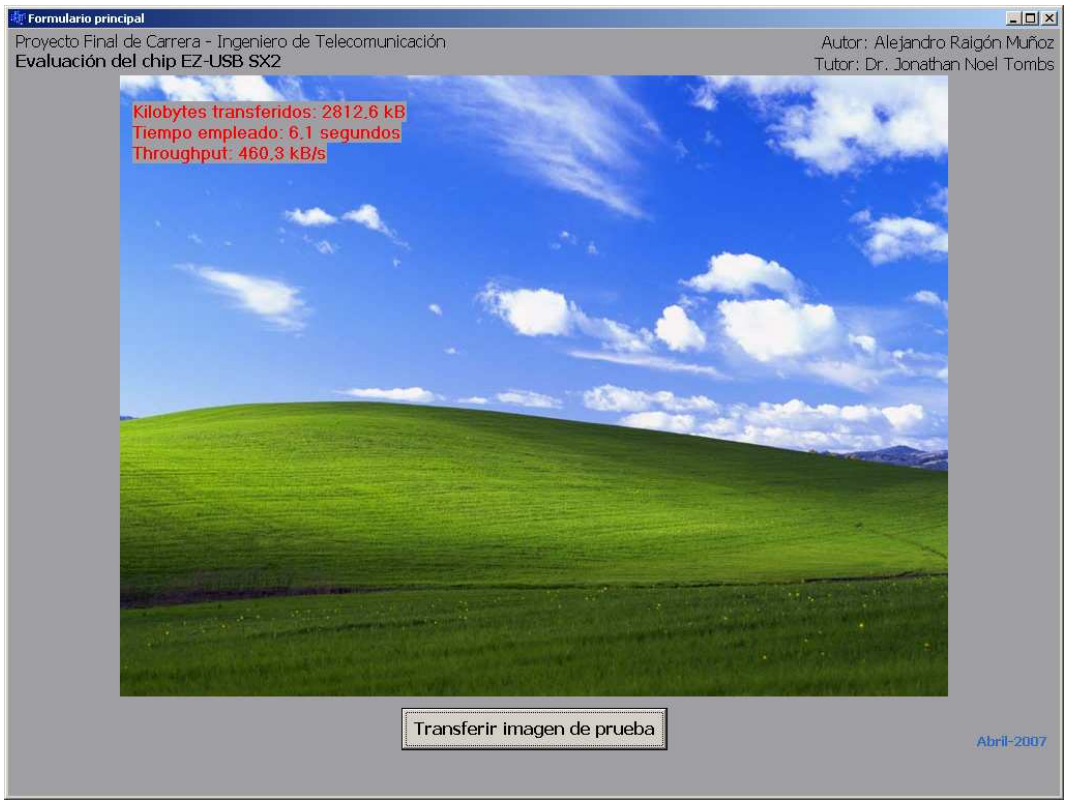


Figura 6.8: Resultado satisfactorio de la transferencia de ida y vuelta de la imagen de prueba.

6.7. Conclusiones

A pesar de los defectos, mencionados con anterioridad, acerca de la mala calidad de la documentación disponible, y del servicio de asistencia técnica mejorable, es posible implementar con éxito una transferencia USB 2.0 utilizando el chip EZ-USB SX2.

Como hemos visto, la sencillez del CyAPI permite reducir enormemente el tiempo que el desarrollador debe invertir en obtener una aplicación USB final funcional, pudiendo centrarse, en consecuencia, en el lado del procesador principal (PIC, DSP, FPGA, etc.).