ANEXOS

[1] Código fuente aplicación de control de datos (C#).

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Threading;
using CyUSB;
using System.Runtime.InteropServices;

namespace Streamer
{
    public class Form1 : System.Windows.Forms.Form
    {
        bool bVista;
        private System.Diagnostics.PerformanceCounter CpuCounter;
        USBDeviceList usbDevices;
        CyUSBDevice MyDevice;
        CyUSBEndPoint EndPoint;

        DateTime t1, t2;
        TimeSpan elapsed;
        double XferBytes;
        long xferRate;
        byte DefaultBufInitValue = 0x01;
        int BufSz;
        int QueueSz;
        int PPX;
        int IsoPktBlockSize;
        int Successes;
        int Failures;
        int a = 1;
        int K = 0;

        Thread tListen;
        static bool bRunning;

        delegate void UpdateUICallback();
        UpdateUICallback updateUI;
        private Label label6;
        private ComboBox DevicesComboBox;
        private GroupBox groupBox2;
        private RadioButton radioButton2;
        private RadioButton radioButton1;
        private TextBox datobox;
        private Label label7;
        private Label label8;
        private Label label9;

        delegate void ExceptionCallback();
        ExceptionCallback handleException;

        public Form1()
        {
            bVista = (Environment.OSVersion.Version.Major < 6) ||
                ((Environment.OSVersion.Version.Major == 6) && Environment.OSVersion.Version.Minor == 0);
            InitializeComponent();
            InitializePeformanceMonitor();
            CPULoadBox.Visible = !bVista;
            updateUI = new UpdateUICallback(StatusUpdate);
            handleException = new ExceptionCallback(ThreadException);
            // Lista de dispositivos conectados
            usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
            //Manejadores de eventos para conexion y desconexion de dispositivos
            usbDevices.DeviceAttached += new EventHandler(usbDevices_DeviceAttached);
            usbDevices.DeviceRemoved += new EventHandler(usbDevices_DeviceRemoved);
            SetDevice(false);
        }

        void usbDevices_DeviceRemoved(object sender, EventArgs e)
        {
            bRunning = false;
            if (tListen != null && tListen.IsAlive == true)
            {
                tListen.Abort();
                tListen.Join();
                tListen = null;
            }
            MyDevice = null;
            EndPoint = null;
            SetDevice(false);

            if (StartBtn.Text.Equals("Start") == false)
            {
                {
                    DevicesComboBox.Enabled = true;
                    EndPointsComboBox.Enabled = true;
                    PpxBox.Enabled = true;
                    QueueBox.Enabled = true;
                    StartBtn.Text = "Start";
```

```
                bRunning = false;
                t2 = DateTime.Now;
                elapsed = t2 - t1;
                xferRate = (long)(XferBytes / elapsed.TotalMilliseconds);
                xferRate = xferRate / (int)100 * (int)100;
                StartBtn.BackColor = Color.SpringGreen;
            }
        }
    }

    void usbDevices_DeviceAttached(object sender, EventArgs e)
    {
        SetDevice(false);
    }

       //Filtra el dispositivo por el VID-PID 04b4-00F1
    private void SetDevice(bool bPreserveSelectedDevice)
    {
        int nCurSelection = 0;
        if (DevicesComboBox.Items.Count > 0)
        {
            nCurSelection = DevicesComboBox.SelectedIndex;
            DevicesComboBox.Items.Clear();
        }
        int nDeviceList = usbDevices.Count;
        for (int nCount = 0; nCount < nDeviceList; nCount++)
        {
            USBDevice fxDevice = usbDevices[nCount];
            String strmsg;
            strmsg = "(0x" + fxDevice.VendorID.ToString("X4") + " - 0x" + fxDevice.ProductID.ToString("X4") + ") " + fxDevice.FriendlyName;
            DevicesComboBox.Items.Add(strmsg);
        }

        if (DevicesComboBox.Items.Count > 0 )
            DevicesComboBox.SelectedIndex = ((bPreserveSelectedDevice == true) ? nCurSelection : 0);

        USBDevice dev = usbDevices[DevicesComboBox.SelectedIndex];

        if (dev != null)
        {
            MyDevice = (CyUSBDevice)dev;
            GetEndpointsOfNode(MyDevice.Tree);
            PpxBox.Text = "16";
            QueueBox.Text = "8";
            if (EndPointsComboBox.Items.Count > 0)
            {
                EndPointsComboBox.SelectedIndex = 0;
                StartBtn.Enabled = true;
            }
            else StartBtn.Enabled = false;
            Text = MyDevice.FriendlyName;
        }
        else
        {
            StartBtn.Enabled = false;
            EndPointsComboBox.Items.Clear();
            EndPointsComboBox.Text = "";
            Text = "USB Streamer Cendit - conecte dispositivo";
        }
    }

      // Muestra información de los endpoints en el endpointComboBox.
    private void GetEndpointsOfNode(TreeNode devTree)
    {
        EndPointsComboBox.Items.Clear();
        foreach (TreeNode node in devTree.Nodes)
        {
            if (node.Nodes.Count > 0)
                GetEndpointsOfNode(node);
            else
            {
                CyUSBEndPoint ept = node.Tag as CyUSBEndPoint;
                if (ept == null)
                {
                }
                else if (!node.Text.Contains("Control"))
                {
                    CyUSBInterface ifc = node.Parent.Tag as CyUSBInterface;
                    string s = string.Format("ALT-{0}, {1} Byte {2}", ifc.bAlternateSetting, ept.MaxPktSize, node.Text);
                    EndPointsComboBox.Items.Add(s);
                }

            }
        }

    }

       //Limpia recursos utilizados.
    protected override void Dispose(bool disposing)
    {
        if (disposing)
        {
            if (components != null)
            {
                components.Dispose();
            }
        }
        base.Dispose(disposing);
    }
```

```
        //Método para determinar el uso del CPU
        private void InitializePeformanceMonitor()
        {
            // No se permite en vista el monitor de recursos
            if (bVista) return;

            CpuCounter = new System.Diagnostics.PerformanceCounter();
            ((System.ComponentModel.ISupportInitialize)(CpuCounter)).BeginInit();
            CpuCounter.CategoryName = "Processor";
            CpuCounter.CounterName = "% Processor Time";
            CpuCounter.InstanceName = "_Total";
            ((System.ComponentModel.ISupportInitialize)(CpuCounter)).EndInit();
        }

        // Punto de entrada principal de la aplicación.
        [STAThread]
        static void Main()
        {
            try
            {
                Application.Run(new Form1());
            }
            catch (Exception e)
            {
                MessageBox.Show(e.StackTrace, "Exception '" + e.Message + "' thrown by " + e.Source);
            }
        }

        private void AboutItem_Click(object sender, System.EventArgs e)
        {
            string assemblyList = Util.Assemblies;
            MessageBox.Show(assemblyList, Text);
        }

        private void ExitItem_Click(object sender, System.EventArgs e)
        {
            Close();
        }

        private void Form1_Load(object sender, System.EventArgs e)
        {
            if (EndPointsComboBox.Items.Count > 0)
                EndPointsComboBox.SelectedIndex = 0;
        }
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            bRunning = false;
            if (tListen != null && tListen.IsAlive == true)
            {
                tListen.Abort();
                tListen.Join();
                tListen = null;
            }

            if (usbDevices != null)
                usbDevices.Dispose();
        }

        //Manejador de eventos del sistema
        private void PpxBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (EndPoint == null) return;
            int ppx = Convert.ToUInt16(PpxBox.Text);
            int len = EndPoint.MaxPktSize * ppx;
            int maxLen = 0x10000; // 64K

            if (len > maxLen)
            {
                ppx = maxLen / (EndPoint.MaxPktSize) / 8 * 8;
                MessageBox.Show("Maximum of 64kB per transfer.  Packets reduced.", "Invalid Packets per Xfer.");
                int iIndex = PpxBox.SelectedIndex; // Get the packet index
                PpxBox.Items.Remove(PpxBox.Text); // Remove the Existing  Packet index
                PpxBox.Items.Insert(iIndex, ppx.ToString()); // insert the ppx
                PpxBox.SelectedIndex = iIndex; // update the selected item index
            }

            if ((MyDevice.bSuperSpeed || MyDevice.bHighSpeed) && (EndPoint.Attributes == 1) && (ppx < 8))
            {
                PpxBox.Text = "8";
                MessageBox.Show("Minimum of 8 Packets per Xfer required for HS/SS Isoc.", "Invalid Packets per Xfer.");
            }
            if ((MyDevice.bHighSpeed) && (EndPoint.Attributes == 1))
            {
                if (ppx > 128)
                {
                    PpxBox.Text = "128";
                    MessageBox.Show("Maximum 128 packets per transfer for High Speed Isoc", "Invalid Packets per Xfer.");
                }
            }
        }

        private void DeviceComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            MyDevice = null;
            EndPoint = null;
            SetDevice(true);
        }
    }
```

```
290         private void EndPointsComboBox_SelectedIndexChanged(object sender, EventArgs e)
291         {
292             // Obtiene alternate settings
293             string sAlt = EndPointsComboBox.Text.Substring(4, 1);
294             byte a = Convert.ToByte(sAlt);
295             MyDevice.AltIntfc = a;
296 
297             // Obtiene endpoints
298             int aX = EndPointsComboBox.Text.LastIndexOf("0x");
299             string sAddr = EndPointsComboBox.Text.Substring(aX, 4);
300             byte addr = (byte)Util.HexToInt(sAddr);
301 
302             EndPoint = MyDevice.EndPointOf(addr);
303 
304             // Comprueba paquetes por transferencia
305             PpxBox_SelectedIndexChanged(sender, null);
306         }
307 
308         //Se ejecuta al hacer click sobre el botón de inicio
309         private void StartBtn_Click(object sender, System.EventArgs e)
310         {
311             if (MyDevice == null)
312                 return;
313 
314             if (QueueBox.Text == "")
315             {
316                 MessageBox.Show("Selecciones transferencias en cola", "Entrada inválida");
317                 return;
318             }
319 
320             if (StartBtn.Text.Equals("Start"))
321             {
322                 DevicesComboBox.Enabled = false;
323                 EndPointsComboBox.Enabled = false;
324                 StartBtn.Text = "Stop";
325                 StartBtn.BackColor = Color.Crimson;
326                 PpxBox.Enabled = false;
327                 QueueBox.Enabled = false;
328 
329                 if (radioButton2.Checked)
330                 DefaultBufInitValue = Convert.ToByte(datobox.Text);
331                 BufSz = EndPoint.MaxPktSize * Convert.ToUInt16(PpxBox.Text);
332                 QueueSz = Convert.ToUInt16(QueueBox.Text);
333                 PPX = Convert.ToUInt16(PpxBox.Text);
334                 EndPoint.XferSize = BufSz;
335 
336                 if (EndPoint is CyIsocEndPoint)
337                     IsoPktBlockSize = (EndPoint as CyIsocEndPoint).GetPktBlockSize(BufSz);
338                 else
339                     IsoPktBlockSize = 0;
340 
341                 bRunning = true;
342                 tListen = new Thread(new ThreadStart(XferThread));
343                 tListen.IsBackground = true;
344                 tListen.Priority = ThreadPriority.Highest;
345                 tListen.Start();
346             }
347             else
348             {
349                 if (tListen.IsAlive)
350                 {
351                     DevicesComboBox.Enabled = true;
352                     EndPointsComboBox.Enabled = true;
353                     PpxBox.Enabled = true;
354                     QueueBox.Enabled = true;
355                     StartBtn.Text = "Start";
356                     bRunning = false;
357                     t2 = DateTime.Now;
358                     elapsed = t2 - t1;
359                     xferRate = (long)(XferBytes / elapsed.TotalMilliseconds);
360                     tListen.Abort();
361                     tListen.Join();
362                     tListen = null;
363                     StartBtn.BackColor = Color.SpringGreen;
364                 }
365             }
366         }
367 
368         // Data Xfer Thread. Inicia al hacer click sobre el botón start
369         public unsafe void XferThread()
370         {
371             // Declaración de buffers
372             byte[][] cmdBufs = new byte[QueueSz][];
373             byte[][] xferBufs = new byte[QueueSz][];
374             byte[][] ovLaps = new byte[QueueSz][];
375             ISO_PKT_INFO[][] pktsInfo = new ISO_PKT_INFO[QueueSz][];
376             int xStart = 0;
377 
378             try
379             {
380                 LockNLoad(ref xStart, cmdBufs, xferBufs, ovLaps, pktsInfo);
381             }
382             catch (NullReferenceException e)
383             {
```

```
            // Excepción para manejar la desconexión en caliente
            e.GetBaseException();
            this.Invoke(handleException);
        }
    }

     // Rutina recursiva que llena los buffers y luego llama a la función XferData para transmitir los datos
    public unsafe void LockNLoad(ref int j, byte[][] cBufs, byte[][] xBufs, byte[][] oLaps, ISO_PKT_INFO[][] pktsInfo)
    {
        cBufs[j] = new byte[CyConst.SINGLE_XFER_LEN + IsoPktBlockSize + ((EndPoint.XferMode == XMODE.BUFFERED) ? BufSz : 0)];
        xBufs[j] = new byte[BufSz];
        for (int iIndex = 0; iIndex < BufSz; iIndex++)
        {
            xBufs[j][iIndex] = DefaultBufInitValue;
            if (radioButton1.Checked == true)
            {
                if (DefaultBufInitValue == 0xff)
                    K = 1;
                if (DefaultBufInitValue == 0x01)
                {
                    K = 0;
                }
                if (K == 1)
                {
                    DefaultBufInitValue--;
                }
                if (K == 0)
                {
                    DefaultBufInitValue++;
                }
            }
         }
        oLaps[j] = new byte[20];
        pktsInfo[j] = new ISO_PKT_INFO[PPX];
        fixed (byte* tL0 = oLaps[j], tc0 = cBufs[j], tb0 = xBufs[j])  // Pin the buffers in memory
        {
            OVERLAPPED* ovLapStatus = (OVERLAPPED*)tL0;
            ovLapStatus->hEvent = (IntPtr)PInvoke.CreateEvent(0, 0, 0, 0);

            // Precarga de la cola
            int len = BufSz;
            if (EndPoint.BeginDataXfer(ref cBufs[j], ref xBufs[j], ref len, ref oLaps[j]) == false)
                Failures++;

            j++;

            if (j < QueueSz)
                LockNLoad(ref j, cBufs, xBufs, oLaps, pktsInfo);  // Llamada recursiva para llenar buffers
            else
                XferData(cBufs, xBufs, oLaps, pktsInfo);
        }
    }

     // Llamada al final del método recursivo LockNLoad().
      XferData() implementa el ciclo infinito de transferencia
    public unsafe void XferData(byte[][] cBufs, byte[][] xBufs, byte[][] oLaps, ISO_PKT_INFO[][] pktsInfo)
    {
        int k = 0;
        int len = 0;

        Successes = 0;
        Failures = 0;

        XferBytes = 0;
        t1 = DateTime.Now;
        long nIteration = 0;

        for (; bRunning; )
        {
            nIteration++;
            // WaitForXfer
            fixed (byte* tmpOvlap = oLaps[k])
            {
                OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmpOvlap;
                if (!EndPoint.WaitForXfer(ovLapStatus->hEvent, 500))
                {
                    EndPoint.Abort();
                    PInvoke.WaitForSingleObject(ovLapStatus->hEvent, 500);
                }
            }

            if (EndPoint.Attributes == 1)
            {
                CyIsocEndPoint isoc = EndPoint as CyIsocEndPoint;
                // FinishDataXfer
                if (isoc.FinishDataXfer(ref cBufs[k], ref xBufs[k], ref len, ref oLaps[k], ref pktsInfo[k]))
                {
                    ISO_PKT_INFO[] pkts = pktsInfo[k];

                    for (int j = 0; j < PPX; j++)
                    {
                        if (pkts[j].Status == 0)
                        {
                            XferBytes += pkts[j].Length;
                            Successes++;
                        }
```

```
                        else
                            Failures++;
                        pkts[j].Length = 0;
                    }
                }
                else
                    Failures++;
            }
            else
            {
                // FinishDataXfer
                if (EndPoint.FinishDataXfer(ref cBufs[k], ref xBufs[k], ref len, ref oLaps[k]))
                {
                    XferBytes += len;
                    Successes++;
                }
                else
                    Failures++;
            }

            len = BufSz;
            if (EndPoint.BeginDataXfer(ref cBufs[k], ref xBufs[k], ref len, ref oLaps[k]) == false)
                Failures++;

            k++;
            if (k == QueueSz)
            {
                k = 0;
                t2 = DateTime.Now;
                elapsed = t2 - t1;
                xferRate = (long)(XferBytes / elapsed.TotalMilliseconds);
                if (bRunning == true)
                {
                    this.Invoke(updateUI);
                }
                Thread.Sleep(1);
            }
        }
        EndPoint.Abort();
    }

    public void StatusUpdate()
    {
        if (bRunning == false) return;
        if (xferRate > ProgressBar.Maximum)
        {
            ProgressBar.Maximum = (int)(xferRate*1.1);
        }
        ProgressBar.Value = (int)xferRate;
        ThroughputLabel.Text = ProgressBar.Value.ToString();

        SuccessBox.Text = Successes.ToString();
        FailuresBox.Text = Failures.ToString();
        label8.Text = Convert.ToString(elapsed);
    }

    public void ThreadException()
    {
        StartBtn.Text = "Start";
        bRunning = false;
        t2 = DateTime.Now;
        elapsed = t2 - t1;
        xferRate = (long)(XferBytes / elapsed.TotalMilliseconds);
        xferRate = xferRate / (int)100 * (int)100;
        tListen = null;
        StartBtn.BackColor = Color.SpringGreen;
    }

    private void PerfTimer_Tick(object sender, EventArgs e)
    {
        if (bVista) return;

        float cpu = CpuCounter.NextValue();
        CpuBar.Value = (int)cpu;
        CpuLabel.Text = string.Format("{0} %", (int)cpu);
    }

    private void radioButton2_CheckedChanged(object sender, EventArgs e)
    {
        if (a % 2 == 0)
        {

        }
        else {
            datobox.Enabled = true;
            this.datobox.Focus();
            MessageBox.Show("Introduzca el Byte a enviar en el cuadro. (0-255)","¡Importante!");
        }
        a++;
    }
    private void radioButton1_CheckedChanged(object sender, EventArgs e)
    {
        datobox.Enabled = false;
    }
}
}
```

[2] Descriptores FX2LP (Assembler).

```
DSCR_DEVICE   equ   1   ;; Descriptor type: Device
DSCR_CONFIG   equ   2   ;; Descriptor type: Configuration
DSCR_STRING   equ   3   ;; Descriptor type: String
DSCR_INTRFC   equ   4   ;; Descriptor type: Interface
DSCR_ENDPNT   equ   5   ;; Descriptor type: Endpoint
DSCR_DEVQUAL  equ   6   ;; Descriptor type: Device Qualifier

DSCR_DEVICE_LEN   equ   18
DSCR_CONFIG_LEN   equ    9
DSCR_INTRFC_LEN   equ    9
DSCR_ENDPNT_LEN   equ    7
DSCR_DEVQUAL_LEN  equ   10

ET_CONTROL   equ   0   ;; Endpoint type: Control
ET_ISO       equ   1   ;; Endpoint type: Isochronous
ET_BULK      equ   2   ;; Endpoint type: Bulk
ET_INT       equ   3   ;; Endpoint type: Interrupt

public      DeviceDscr, DeviceQualDscr, HighSpeedConfigDscr, FullSpeedConfigDscr, StringDscr, UserDscr

;DSCR   SEGMENT   CODE

;;-------------------------------------------------------------------------------
;; Global Variables
;;-------------------------------------------------------------------------------
;      rseg DSCR     ;; locate the descriptor table in on-part memory.

CSEG   AT 100H

DeviceDscr:
      db   DSCR_DEVICE_LEN      ;; Descriptor length
      db   DSCR_DEVICE   ;; Decriptor type
      dw   0002H      ;; Specification Version (BCD)
      db   00H        ;; Device class
      db   00H         ;; Device sub-class
      db   00H         ;; Device sub-sub-class
      db   64         ;; Maximum packet size
      dw   0B404H      ;; Vendor ID
      dw   0310H      ;; Product ID (Sample Device)
      dw   0000H      ;; Product version ID
      db   1         ;; Manufacturer string index
      db   2         ;; Product string index
      db   0         ;; Serial number string index
      db   1         ;; Number of configurations

org (($ / 2) +1) * 2
DeviceQualDscr:
      db   DSCR_DEVQUAL_LEN   ;; Descriptor length
      db   DSCR_DEVQUAL   ;; Decriptor type
      dw   0002H      ;; Specification Version (BCD)
      db   00H        ;; Device class
      db   00H         ;; Device sub-class
      db   00H         ;; Device sub-sub-class
      db   64         ;; Maximum packet size
      db   1         ;; Number of configurations
      db   0         ;; Reserved

org (($ / 2) +1) * 2

HighSpeedConfigDscr:
      db   DSCR_CONFIG_LEN              ;; Descriptor length
      db   DSCR_CONFIG                 ;; Descriptor type
      db   (HighSpeedConfigDscrEnd-HighSpeedConfigDscr) mod 256 ;; Total Length (LSB)
      db   (HighSpeedConfigDscrEnd-HighSpeedConfigDscr)  /  256 ;; Total Length (MSB)
      db   1      ;; Number of interfaces
      db   1      ;; Configuration number
      db   0      ;; Configuration string
      db   10100000b   ;; Attributes (b7 - buspwr, b6 - selfpwr, b5 - rwu)
      db   50      ;; Power requirement (div 2 ma)
```

```
;; Alt Interface 0 Descriptor - NO ENDPOINTS!!!!!!!!
      db    DSCR_INTRFC_LEN   ;; Descriptor length
      db    DSCR_INTRFC       ;; Descriptor type
      db    0                 ;; Zero-based index of this interface
      db    0                 ;; Alternate setting
      db    0                 ;; Number of end points
      db    0ffH              ;; Interface class
      db    00H               ;; Interface sub class
      db    00H               ;; Interface sub sub class
      db    0                 ;; Interface descriptor string index


;; Alt. Interface 1 Descriptor - Isoc OUT 3x1024 byte packets/uFrame
      db    DSCR_INTRFC_LEN   ;; Descriptor length
      db    DSCR_INTRFC       ;; Descriptor type
      db    0                 ;; Zero-based index of this interface
      db    1                 ;; Alternate setting
      db    1                 ;; Number of end points
      db    0ffH              ;; Interface class
      db    00H               ;; Interface sub class
      db    00H               ;; Interface sub sub class
      db    0                 ;; Interface descriptor string index

;; Isoc OUT Endpoint Descriptor
      db    DSCR_ENDPNT_LEN   ;; Descriptor length
      db    DSCR_ENDPNT       ;; Descriptor type
      db    02H               ;; Endpoint 2 and direction OUT
      db    ET_ISO            ;; Endpoint type
      db    00H               ;; Maximun packet size (LSB)
      db    14H               ;; Max packect size (MSB) 10100b 3x1024 byte packets/uFrame
      db    01H               ;; Polling interval


HighSpeedConfigDscrEnd:

org (($ / 2) +1) * 2

FullSpeedConfigDscr:
      db    DSCR_CONFIG_LEN              ;; Descriptor length
      db    DSCR_CONFIG                 ;; Descriptor type
      db    (FullSpeedConfigDscrEnd-FullSpeedConfigDscr) mod 256 ;; Total Length (LSB)
      db    (FullSpeedConfigDscrEnd-FullSpeedConfigDscr)  /  256 ;; Total Length (MSB)
      db    1       ;; Number of interfaces
      db    1       ;; Configuration number
      db    0       ;; Configuration string
      db    10100000b   ;; Attributes (b7 - buspwr, b6 - selfpwr, b5 - rwu)
      db    50      ;; Power requirement (div 2 ma)

;; Interface Descriptor
      db    DSCR_INTRFC_LEN      ;; Descriptor length
      db    DSCR_INTRFC         ;; Descriptor type
      db    0                ;; Zero-based index of this interface
      db    0                ;; Alternate setting
      db    1                ;; Number of end points
      db    0ffH             ;; Interface class
      db    00H                ;; Interface sub class
      db    00H                ;; Interface sub sub class
      db    0                ;; Interface descriptor string index

;; Endpoint Descriptor
      db    DSCR_ENDPNT_LEN   ;; Descriptor length
      db    DSCR_ENDPNT       ;; Descriptor type
      db    02H               ;; Endpoint number, and direction
      db    ET_ISO            ;; Endpoint type
      db    0ffH              ;; Maximun packet size (LSB)
      db    03H               ;; Max packect size (MSB)
      db    01H               ;; Polling interval

FullSpeedConfigDscrEnd:

org (($ / 2) +1) * 2

UserDscr:
      dw    0000H
      end
```

[3] Firmware FX2LP (fw.c).

```c
#include "fx2.h"
#include "fx2regs.h"
//-----------------------------------------------------------------------------
// Constantes
//-----------------------------------------------------------------------------
#define DELAY_COUNT   0x9248*8L  // Delay for 8 sec at 24Mhz, 4 sec at 48
#define _IFREQ  48000            // IFCLK constant for Synchronization Delay
#define _CFREQ  48000            // CLKOUT constant for Synchronization Delay
//-----------------------------------------------------------------------------
// Macros
//-----------------------------------------------------------------------------
#define   min(a,b) (((a)<(b))?(a):(b))
#define   max(a,b) (((a)>(b))?(a):(b))
#include "fx2sdly.h"             // Define _IFREQ and _CFREQ above this #include
//-----------------------------------------------------------------------------
// Variables Globales
//-----------------------------------------------------------------------------
volatile BOOL   GotSUD;
BOOL      Rwuen;
BOOL      Selfpwr;
volatile BOOL   Sleep;                  // Sleep mode enable flag
WORD   pDeviceDscr;   // Pointer to Device Descriptor; Descriptors may be moved
WORD   pDeviceQualDscr;
WORD   pHighSpeedConfigDscr;
WORD   pFullSpeedConfigDscr;
WORD   pConfigDscr;
WORD   pOtherConfigDscr;
WORD   pStringDscr;
//-----------------------------------------------------------------------------
// Prototypes
//-----------------------------------------------------------------------------
void SetupCommand(void);
void TD_Init(void);
void TD_Poll(void);
BOOL TD_Suspend(void);
BOOL TD_Resume(void);
BOOL DR_GetDescriptor(void);
BOOL DR_SetConfiguration(void);
BOOL DR_GetConfiguration(void);
BOOL DR_SetInterface(void);
BOOL DR_GetInterface(void);
BOOL DR_GetStatus(void);
BOOL DR_ClearFeature(void);
BOOL DR_SetFeature(void);
BOOL DR_VendorCmnd(void);
// control de endpoints y registros(EPnCS)
#define epcs(EP) (EPCS_Offset_Lookup_Table[(EP & 0x7E) | (EP > 128)] + 0xE6A1)

// Manejador de tareas
void main(void)
{
   DWORD   i;
   WORD   offset;
   DWORD   DevDescrLen;
   DWORD   j=0;
   WORD   IntDescrAddr;
   WORD   ExtDescrAddr;

   // Inicializar Global States
   Sleep = FALSE;               // Disable sleep mode
   Rwuen = FALSE;               // Disable remote wakeup
   Selfpwr = FALSE;            // Disable self powered
   GotSUD = FALSE;               // Clear "Got setup data" flag

   // Inicializar user device
   TD_Init();
   EZUSB_IRQ_ENABLE();            // Enable USB interrupt (INT2)
   EZUSB_ENABLE_RSMIRQ();            // Wake-up interrupt
   INTSETUP |= (bmAV2EN | bmAV4EN);     // Enable INT 2 & 4 autovector
   USBIE |= bmSUDAV | bmSUTOK | bmSUSP | bmURES | bmHSGRANT;   // Enable selected interrupts
   EA = 1;                  // Enable 8051 interrupts

#ifndef NO_RENUM
   // Renumeración
   if(!(USBCS & bmRENUM))
   {
      EZUSB_Discon(TRUE);
   }
#endif
```

```
    USBCS &=~bmDISCON;
    CKCON = (CKCON&(~bmSTRETCH)) | FW_STRETCH_VALUE; // Set stretch to 0

    Sleep = FALSE;               // clear Sleep flag.

    // Ciclo infinito
    while(TRUE)
    {
       if(GotSUD)             // Espera por SUDAV
       {
          SetupCommand();          // Implement setup command
            GotSUD = FALSE;            // Clear SUDAV flag
       }
       if (Sleep)        //manejador modo bajo consumo
       {
           if(TD_Suspend())
           {
               Sleep = FALSE;
               do
               {
                   EZUSB_Susp();         // idle mode.
               }
               while(!Rwuen && EZUSB_EXTWAKEUP());
              EZUSB_Resume();
              TD_Resume();
           }
       }
    }
}

// Manejo de solicitudes
void SetupCommand(void)
{
    void   *dscr_ptr;

    switch(SETUPDAT[1])
    {
       case SC_GET_DESCRIPTOR:                  // *** Get Descriptor
          if(DR_GetDescriptor())
             switch(SETUPDAT[3])
             {
                case GD_DEVICE:            // Device
                   SUDPTRH = MSB(pDeviceDscr);
                   SUDPTRL = LSB(pDeviceDscr);
                   break;
                case GD_DEVICE_QUALIFIER:            // Device Qualifier
                   SUDPTRH = MSB(pDeviceQualDscr);
                   SUDPTRL = LSB(pDeviceQualDscr);
                   break;
                case GD_CONFIGURATION:         // Configuration
                   SUDPTRH = MSB(pConfigDscr);
                   SUDPTRL = LSB(pConfigDscr);
                   break;
                case GD_OTHER_SPEED_CONFIGURATION:  // Other Speed Configuration
                   SUDPTRH = MSB(pOtherConfigDscr);
                   SUDPTRL = LSB(pOtherConfigDscr);
                   break;
                case GD_STRING:            // String
                   if(dscr_ptr = (void *)EZUSB_GetStringDscr(SETUPDAT[2]))
                   {
                      SUDPTRH = MSB(dscr_ptr);
                      SUDPTRL = LSB(dscr_ptr);
                   }
                   else
                      EZUSB_STALL_EP0();   // Stall End Point 0
                   break;
                default:           // Invalid request
                   EZUSB_STALL_EP0();      // Stall End Point 0
             }
          break;
       case SC_GET_INTERFACE:                   // *** Get Interface
          DR_GetInterface();
          break;
       case SC_SET_INTERFACE:                   // *** Set Interface
          DR_SetInterface();
          break;
       case SC_SET_CONFIGURATION:                // *** Set Configuration
          DR_SetConfiguration();
          break;
       case SC_GET_CONFIGURATION:                // *** Get Configuration
          DR_GetConfiguration();
          break;
       case SC_GET_STATUS:                  // *** Get Status
          if(DR_GetStatus())
             switch(SETUPDAT[0])
             {
                case GS_DEVICE:            // Device
                   EP0BUF[0] = ((BYTE)Rwuen << 1) | (BYTE)Selfpwr;
                   EP0BUF[1] = 0;
                   EP0BCH = 0;
                   EP0BCL = 2;
                   break;
```

```
173                 case GS_INTERFACE:          // Interface
174                    EP0BUF[0] = 0;
175                    EP0BUF[1] = 0;
176                    EP0BCH = 0;
177                    EP0BCL = 2;
178                    break;
179                 case GS_ENDPOINT:          // End Point
180                    EP0BUF[0] = *(BYTE xdata *) epcs(SETUPDAT[4]) & bmEPSTALL;
181                    EP0BUF[1] = 0;
182                    EP0BCH = 0;
183                    EP0BCL = 2;
184                    break;
185                 default:             // Invalid Command
186                    EZUSB_STALL_EP0();       // Stall End Point 0
187              }
188           break;
189        case SC_CLEAR_FEATURE:                   // *** Clear Feature
190           if(DR_ClearFeature())
191              switch(SETUPDAT[0])
192              {
193                 case FT_DEVICE:             // Device
194                    if(SETUPDAT[2] == 1)
195                       Rwuen = FALSE;        // Disable Remote Wakeup
196                    else
197                       EZUSB_STALL_EP0();    // Stall End Point 0
198                    break;
199                 case FT_ENDPOINT:          // End Point
200                    if(SETUPDAT[2] == 0)
201                    {
202                       *(BYTE xdata *) epcs(SETUPDAT[4]) &= ~bmEPSTALL;
203                       EZUSB_RESET_DATA_TOGGLE( SETUPDAT[4] );
204                    }
205                    else
206                       EZUSB_STALL_EP0();    // Stall End Point 0
207                    break;
208              }
209           break;
210        case SC_SET_FEATURE:                   // *** Set Feature
211           if(DR_SetFeature())
212              switch(SETUPDAT[0])
213              {
214                 case FT_DEVICE:             // Device
215                    if(SETUPDAT[2] == 1)
216                       Rwuen = TRUE;       // Enable Remote Wakeup
217                    else if(SETUPDAT[2] == 2)
218                       break;
218                       break;
219                    else
220                       EZUSB_STALL_EP0();    // Stall End Point 0
221                    break;
222                 case FT_ENDPOINT:          // End Point
223                    *(BYTE xdata *) epcs(SETUPDAT[4]) |= bmEPSTALL;
224                    break;
225              }
226           break;
227        default:                      // *** Invalid Command
228           if(DR_VendorCmnd())
229              EZUSB_STALL_EP0();            // Stall End Point 0
230     }
231     // Acknowledge handshake
232     EP0CS |= bmHSNAK;
233  }
234  // Wake-up interrupt
235  void resume_isr(void) interrupt WKUP_VECT
236  {
237     EZUSB_CLEAR_RSMIRQ();
238  }
```

[4] Firmware (CYStream.c).

```
#pragma NOIV               // Do not generate interrupt vectors
#include "fx2.h"
#include "fx2regs.h"
#include "fx2sdly.h"            // SYNCDELAY macro

extern BOOL   GotSUD;         // Received setup data flag
extern BOOL   Sleep;
extern BOOL   Rwuen;
extern BOOL   Selfpwr;

enum {              //Enumera desde cero las alternativas de ENDPOINT
    Alt0 = 0,
    Alt1_IsocOUT
};
enum {
    Full_Alt0= 0,   //Enumera desde cero las alternativas de ENDPOINT (FULLSPEED)
    Full_Alt1_Full
};
BYTE    Configuration;      // Current configuration
BYTE    AlternateSetting = Alt1_IsocOUT;   // Alternate settings INICIAL.

void TD_Init(void)             //FUNCION DE INCIALIZACIÓN
{
    #if 1
   ////////////////////////--1 Y 2. CPU CLOCK Y IFCONFIG --////////////////////////////,
   // set the CPU clock to 48MHz
   CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1) ;  //CPUCS = 0x10;
   SYNCDELAY;

   // set the slave FIFO interface to 48MHz
   IFCONFIG = 0xFB;     // asíncrono
   SYNCDELAY;

    ////////////////////////3. Set REVCTL.0 Y REVCTL.1 to '1'.////////////////////////////,
    REVCTL=0x03; //DEBE SER INICIALIZADO A '1'
    SYNCDELAY;

    ////////////////////////4. CONFIGURA EP2 Y DESHABILITA LOS DEMÁS EP´S//////////////////,
    EP2CFG = 0x98;  //0x98: EP2 IS DIR=OUT, TYPE=ISO, SIZE=1024, BUF=4X
    SYNCDELAY;
    EP1OUTCFG = (EP1OUTCFG & 0x7F);
    SYNCDELAY;
    EP1INCFG = (EP1INCFG & 0x7F);
    SYNCDELAY;
    EP4CFG = (EP4CFG & 0x7F);
    SYNCDELAY;
    EP6CFG = (EP6CFG & 0x7F);
    SYNCDELAY;
    EP8CFG = (EP8CFG & 0x7F);
    SYNCDELAY;

    ////////////////////////5. RESETEA TODOS LOS FIFOS-SEGUN EL PROCESO DEL MANUAL TÉCNICO.//////////
    FIFORESET = 0x80;
    SYNCDELAY;
    FIFORESET = 0x82; // reset, FIFO 2
    SYNCDELAY; //
    FIFORESET = 0x84; // reset, FIFO 4
    SYNCDELAY; //
    FIFORESET = 0x86; // reset, FIFO 6
    SYNCDELAY; //
    FIFORESET = 0x88; // reset, FIFO 8
    SYNCDELAY; //
    FIFORESET = 0x00; // deactivate NAK-ALL
    SYNCDELAY;

    ////////////////////////6. ARM OUT BUFFERS-->WRITTING N TIMES OUTPKTEND w/skip=1//////////////////
    OUTPKTEND = 0x82;
    SYNCDELAY;
    OUTPKTEND = 0x82;
    SYNCDELAY;
    OUTPKTEND = 0x82;
    SYNCDELAY;
    OUTPKTEND = 0x82;
    SYNCDELAY;
    ////////////////////////7. . Set bit EP2FIFOCFG.4=1. ////////////////////////////////////////////
    EP2FIFOCFG = 0x10; //ESTE REGISTRO NOS PERMITE TENER DATOS EN LOS FIFOS!! AUTOOUT=1;
    SYNCDELAY;
    //no utiliza los demas edpoint y no son amplicables las condiciones que se le configuren.
    EP4FIFOCFG = 0x0C; // EP4 is AUTOOUT=0, AUTOIN=1, ZEROLEN=1, WORDWIDE=0
    SYNCDELAY;
    EP6FIFOCFG = 0x0C; // EP6 is AUTOOUT=0, AUTOIN=1, ZEROLEN=1, WORDWIDE=0
    SYNCDELAY;
    EP8FIFOCFG = 0x0C; // EP8 is AUTOOUT=0, AUTOIN=1, ZEROLEN=1, WORDWIDE=0
    SYNCDELAY;
```

```
    ////////////////////////////(FIFO)////////////////////////////////////////////////////////
    PINFLAGSAB = 0x00;      // defines FLAGA as prog-level flag, pointed to by FIFOADR[1:0]
    SYNCDELAY;              // FLAGB as full flag, as pointed to by FIFOADR[1:0]
    PINFLAGSCD = 0x00;      // FLAGC as empty flag, as pointed to by FIFOADR[1:0]
    SYNCDELAY;
    PORTACFG = 0x40; //set bit 6=1, para que pin7 sea: SCLS#
    SYNCDELAY;
    FIFOPINPOLAR = 0x3F; //0X3F SLOE,SLWR,PKTEND,SLRD,EF,FFes configurado para activar en HIGH
    #endif
    Rwuen = TRUE;                // Enable remote-wakeup
}
void TD_Poll(void)   //Vacia xq el 8051 no hace nada en la transmisión de datos
{

}
BOOL TD_Suspend(void)
{
    return(TRUE);
}
BOOL TD_Resume(void)
{
    return(TRUE);
}
BOOL DR_GetDescriptor(void)
{
    return(TRUE);            //ESTO ES PARA SABER CUANDO SE HA ENVIADO EL DESCRIPTOR AL HOST!!
}
BOOL DR_SetConfiguration(void)
{
    Configuration = SETUPDAT[2];
    return(TRUE);
}
BOOL DR_GetConfiguration(void)
{
    EP0BUF[0] = Configuration;
    EP0BCH = 0;
    EP0BCL = 1;
    return(TRUE);
}
BOOL DR_SetInterface(void) //ESTO DEBE SUCEDER AL SELECCIONAR ENDPOINT EN C# STREAMER
{
    AlternateSetting = SETUPDAT[2];
    return(TRUE);
}
BOOL DR_GetInterface(void)
{
    EP0BUF[0] = AlternateSetting;
    EP0BCH = 0;
    EP0BCL = 1;
    return(TRUE);
}
BOOL DR_GetStatus(void)
{
    return(TRUE);
}
BOOL DR_ClearFeature(void)
{
    return(TRUE);
}
BOOL DR_SetFeature(void)
{
    return(TRUE);
}
BOOL DR_VendorCmnd(void)
{
    return(TRUE);
}
void ISR_Sudav(void) interrupt 0
{
    GotSUD = TRUE;            // Set flag
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSUDAV;         // Clear SUDAV IRQ
}
void ISR_Sutok(void) interrupt 0
{
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSUTOK;         // Clear SUTOK IRQ
}
void ISR_Sof(void) interrupt 0
{
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSOF;            // Clear SOF IRQ
}
```

```
//INTERRUPCION QUE OCURRE LUEGO DE UN RESET, SE COLOCA COMO PREDEFINIDA FS
void ISR_Ures(void) interrupt 0
{
    if(EZUSB_HIGHSPEED())
    {
        pConfigDscr = pHighSpeedConfigDscr;
        pOtherConfigDscr = pFullSpeedConfigDscr;
    }
    else
    {
        pConfigDscr = pFullSpeedConfigDscr;
        pOtherConfigDscr = pHighSpeedConfigDscr;
    }
    ((CONFIGDSCR xdata*)pConfigDscr)->type = CONFIG_DSCR;
    ((CONFIGDSCR xdata*)pOtherConfigDscr)->type = OTHERSPEED_DSCR;

    EZUSB_IRQ_CLEAR();
    USBIRQ = bmURES; // Clear URES IRQ
}
void ISR_Susp(void) interrupt 0
{
   Sleep = TRUE;
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSUSP;
}
//INTERRUPCION QUE OCURRE LUEGO DE UN A NEGOCIACION HS, SE COLOCA COMO PREDEFINIDA HS, SINO FS PREDEFINIDA
//ESTA FUNCION SE ENCARGA DE ASOCIAR TIPO CONFIG A HS Y OTHERSPEED A FS O VICEVERSA
void ISR_Highspeed(void) interrupt 0
{
   if (EZUSB_HIGHSPEED())
   {
      pConfigDscr = pHighSpeedConfigDscr;
      ((CONFIGDSCR xdata *) pConfigDscr)->type = CONFIG_DSCR;
      pOtherConfigDscr = pFullSpeedConfigDscr;
      ((CONFIGDSCR xdata *) pOtherConfigDscr)->type = OTHERSPEED_DSCR;
   }
   else
   {
      pConfigDscr = pFullSpeedConfigDscr;
      pOtherConfigDscr = pHighSpeedConfigDscr;
   }
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmHSGRANT;
}
```