AN58764

Implementing a Virtual COM Port Using FX2LP™

Author: Prajith Cheerakkoda

Associated Project: Yes

Associated Part Family: CY7C68013/14/15/16

Related Resources: Click here

More code examples? We heard you.

For a consolidated list of USB Hi-Speed Code Examples, visit the Cypress webpage.

AN58764 explains how to use a Cypress EZ-USB® FX2LP™ chip to implement the USB Communication Device Class (CDC). An FX2LP-based design can communicate with a PC as a standard COM (serial) device. This document contains example code with the required descriptors, code to handle class-specific requests, and the Windows information (INF) file required to enumerate the CDC device.

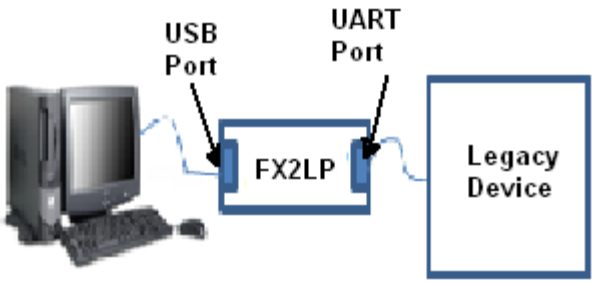
Contents

1 Introduction

The Cypress EZ-USB® FX2LP™ is a Hi-Speed USB peripheral controller. The programmability and flexibility of FX2LP allows you to implement USB device classes such as the Communication Device Class (CDC), the mass storage class (MSC), and the human interface device (HID) class. This application note discusses the implementation of a virtual COM port device using FX2LP. A PC terminal program can open a terminal, select the FX2LP COM device (such as COM5), and communicate with FX2LP as if it were UART-based. By using the default Windows serial driver (usbser.sys), you eliminate the need for an additional driver. A virtual COM port (see Figure 1) can be used with other high-speed interfaces for debug and manufacturing.

This application note assumes familiarity with chapter 14, “Timers/Counters and Serial Interface,” and section 9.2 (“Interface between EZ-USB firmware and FIFOs”) of the EZ-USB Technical Reference Manual.

Figure 1. FX2LP as a Virtual COM Port



2 Virtual COM Port

Nowadays, desktops and laptops do not have a legacy COM port with its telltale DB-9 connector. The USB CDC provides a way to replace the missing COM port (see Figure 2). Microsoft Windows natively comes with a serial driver *usbser.sys*, which does the UART to USB translations and vice versa. This driver allows legacy applications, such as HyperTerminal or Tera Term, to communicate with legacy devices (see Figure 3).

Figure 2. PC with a COM Port

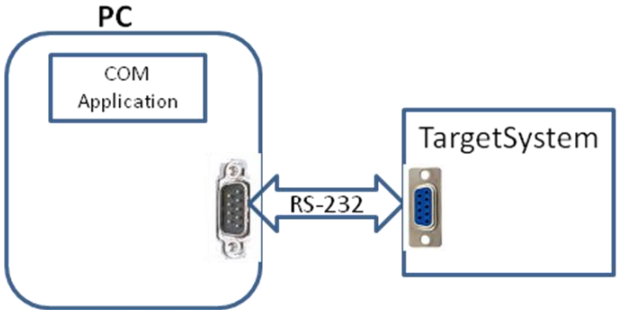
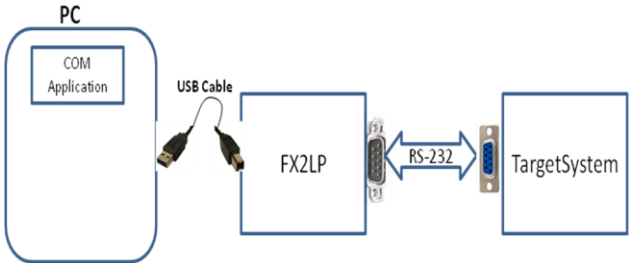


Figure 3. PC with a Virtual COM Port



3 USB Communication Device Class Specification

The USB CDC, which defines the architecture for emulating telecommunication and network devices on USB ports, is a composite USB device class. This means it supports multiple interfaces, allowing the virtual COM port to exist alongside other interfaces such as data, control, or other standard USB class interfaces. The USB CDC specifications, available through the USB Implementers Forum website, supports five basic models of communication, each having one or more subclasses.

POTS: Plain old telephone system. A system of devices that communicate through ordinary phone lines and generic COM port devices.

ISDN: Integrated services digital network. A communication system that uses phone lines with ISDN interfaces.

Networking model: Communication through Ethernet or ATM.

Wireless mobile communication: A communication model with cellphones that support voice and data communication.

Ethernet emulation model: An efficient way for devices to send and receive Ethernet frames.

In this project for serial emulation we implement the POTS model, using the Abstract Control Model (ACM) subclass.

3.1 Abstract Control Model

The Abstract Control Model can bridge the gap between legacy modem devices and USB devices. It requires two interfaces for communication devices: the communication class interface and the data class interface.

Communication Class Interface: This device management interface controls operation of the device. In addition to conveying standard USB requests, it conveys class-specific requests to the device. These latter requests are covered in the Class-Specific Requests section.

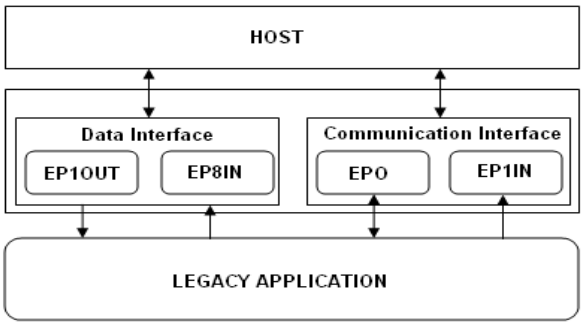
This interface also has an optional notification component. It requires one interrupt IN endpoint to notify the host about the status of the control signals. Even though the notification element is an optional component of the communication interface, it is required in a standard PC.

Data Class Interface: This interface handles data transfer between the host and the device. It requires two bulk endpoints: the IN receives data and the OUT sends it.

3.2 Data Transfer

In the Abstract Control Model subclass, the interfaces are implemented as in Figure 4.

Figure 4. CDC Interface in the Abstract Control Model



In the attached project, the host sends data from the legacy application to a legacy device through the FX2LP EP1OUT bulk endpoint. Similarly, data received from the legacy device arrives via the FX2LP EP8IN bulk endpoint. The host reads the IN endpoint to receive data.

3.3 Device Management

The host manages the communication interface using default control endpoint EP0. The USB CDC specifications defines specific requests for CDC device management.

3.4 Class-Specific Requests

To support certain types of legacy applications, two problems need to be addressed: supporting specific legacy control signals and supporting state variables. Class-specific requests are defined to support these requirements.

Set_line_coding: Sets asynchronous serial parameters: bit rate, number of stop bits, parity, and number of data bits. Refer to Table 1.

- **Bit rate:** This parameter sets the baud rate of communication (i.e., the number of bits transmitted per second).
- **Parity bit:** This parameter is used for error handling. The project attached to this application note does not use parity. Therefore, set it as “None” in the host application.
- **Number of stop bits:** Bits following actual data bits. Normally set to 1.
- **Number of data bits:** Number of bits transferred for each byte of data. Normally set to 8.

Get_line_coding: Requests asynchronous serial parameters: bit rate, number of stop bits, parity, and number of data bits. Host requests the present device configuration. Refer to Table 2.

Set_control_line_state: Sets RS-232 signals, RTS, and DTR. This setting is optional but is required if the signals are implemented. Because these signals are not present in FX2LP, the attached project doesn't use them.

The device returns notifications through the interrupt IN endpoint. The host issues periodic IN tokens to the endpoint. The endpoint returns the notification if available and acknowledges if it does not have any notification. In the firmware, SERIAL STATE notification sends the state of CD, DSR, Break, and RI. Sending this notification is optional. Refer to Table 3, Table 4, Table 5.

Table 1. SET_LINE_CODING Request Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_LINE_CODING (0X20h)	Zero	Interface	Size of structure	Line coding structure

Table 2. GET_LINE_CODING Request Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_LINE_CODING (0x21h)	Zero	Interface	Size of structure	Line coding structure

Table 3. Line Coding Structure

Offset	Field	Size	Value	Description
0	dwDTERate	4	Number	Data terminal rate (in bits per second)
4	bCharFormat	1	Number	Stop bits 0 - 1 Stop bit 1 - 1.5 Stop bits 2 - 2 Stop bits
5	bParityType	1	Number	Parity 0 - None 1 - Odd 2 - Even 3 - Mark 4 - Space
6	bDataBits	1	Number	Data bits 5, 6, 7, 8, or 16

Table 4. SET_CONTROL_LINE_STATE Request Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_CONTROL_LINE_STATE (0x22h)	Control signal bitmap	Interface	Zero	None

Table 5. Control Signal Bitmap Values for SetControlLineState

Bit Position	Description
D15….D2	RESERVED(Reset to zero)
D1	Carrier control for half duplex modems. This signal corresponds to V.24 signal 105 and RS-232 signal RTS. 0 - Deactivate carrier 1 - Activate carrier The device ignores the value of this bit when operating in full duplex mode.
D0	Indicates to DCE if DTE is present or not. This signal corresponds to V.24 signal 108/2 and RS-232 signal DTR. 0 - Not Present 1 – Present

4 Firmware

The virtual COM example code contains three files:

***Fw.c*:** Firmware Framework from Cypress. It provides low-level USB functionality with firmware hooks into the user program.

***Virtual.c*:** Contains definitions of the functions and interrupt service routines (ISRs) used for implementing the USB peripheral functionality as a virtual COM device.

***DSCR.A51*:** Assembly source code that contains USB descriptor data for the communication class and device class interfaces.

Because CDC is a composite USB device class, it has two interfaces: a data interface and a communication interface. The communication interface includes CDC class-specific descriptors called functional descriptors. The data class interface doesn't require any class-specific descriptors.

4.1 Functional Descriptors

Functional descriptors hold CDC class-specific information in a CDC descriptor. Following are the functional descriptors.

- Header functional descriptor
- Abstract Control Management (ACM) functional descriptor
- Union functional descriptor
- Call management functional descriptor

The descriptors are declared in the assembly source `DSCR.a51`. This assembly code is part of the attached project in this application note.

```
;; Header Functional Descriptor
       db    05H                                ;; Descriptor Size in Bytes (5)
       db    24H                                ;; CS_Interface
       db    00H                                ;; Header Functional Descriptor
       dw    1001H                              ;; bcdCDC

   ;; ACM Functional Descriptor
      db    04H                                 ;; Descriptor Size in Bytes (5)
      db    24H                                 ;; CS_Interface
      db    02H                                 ;; Abstarct Control Management Functional
Desc
      db    02H                                 ;; bmCapabilities

   ;; Union Functional Descriptor
      db    05H                                 ;; Descriptor Size in Bytes (5)
      db    24H                                 ;; CS_Interface
      db    06H                                 ;; Union Functional Descriptor
      db    00H                                 ;; bMasterInterface
      db    01H                                 ;; bSlaveInterface0

    ;; CM Functional Descriptor
      db    05H                                 ;; Descriptor Size in Bytes (5)
      db    24H                                 ;; CS_Interface
      db    01H                                 ;; CM Functional Descriptor
      db    00H                                 ;; bmCapabilities
      db    01H                                 ;; bDataInterface
```

4.1.1 Header Functional Descriptor

A header functional descriptor specifies the CDC version on which to implement the device. The version shown above is 1.1 (bcd coding, least significant byte first).

4.1.2 ACM Functional Descriptor

This descriptor specifies the commands supported by the communication class interface. Our device supports the request combination of Set_Line_Coding, Set_ Control_Line_State, and Get_Line_Coding, and the notification Serial_State. This is signified by the bmCapabilities byte equal to 0x02.

4.1.3 Union Functional Descriptor

This descriptor specifies the relationship among the interfaces that form a single functional unit. It sets one of the interfaces in the group as a master, which functions as a controlling interface for the group, and others as slaves. Here, the communication class interface acts as the master and the data class interface acts as the slave. Host and device exchange requests and notifications take place via the communication class interface. Standard and class-specific requests for the group are sent to this interface using the EP0 control endpoint. The master sends notifications using the EP1IN interrupt.

4.1.4 Call Management Functional Descriptor

This descriptor specifies how the device handles call management. Our device doesn't handle call management. The bmCapabilities field is a bitmap that shows information about device call management and the interface used for exchange of call management information.

For more information on functional descriptors, see section 5.2.3 of the USB CDC specifications.

The VID/PID can be changed according to the customer requirement in the device descriptor.

4.2 Endpoints

The communication interface uses default endpoint EP0 for device management for all standard and communication specific requests. The notification element is implemented by interrupt endpoint EP1IN.

Data interface endpoints are restricted to either bulk or isochronous and should be of the same type. In the attached project, the data interface uses EP1OUT and EP8IN for host data out and host data in respectively, both as bulk.

4.3 Data Transfer

4.3.1 DATA OUT PATH

The following FX2LP code implements data transfer from the host to the device. The OUT path comprises two ISRs:

Serial port 0 interrupt:

```
ISR_USART0(void) interrupt 4
```

EP1 OUT endpoint interrupt:

```
ISR_Ep1out(void) interrupt 0
```

For the EP1OUT endpoint, the interrupt request signifies that OUT data has been sent from the host and validated by the EZ-USB FX2LP, and is in the endpoint buffer memory. The endpoint ISR initiates FX2LP receipt of the OUT data by placing the first byte in the endpoint buffer into the serial buffer SBUF0. Remaining bytes are then transferred by the transmit() function, which is triggered by ISR_USART0.

```
void ISR_Ep1out(void) interrupt 0
// Places first byte into SBUF0
{
      EZUSB_IRQ_CLEAR();
      //Clears the USB interrupt
      EPIRQ = bmBIT3;
      //Clears EP1 OUT interrupt request
      while (TI == 1) ;

      i=0;
      bcl=EP1OUTBC;
      SBUF0=EP1OUTBUF[i++];
      }
void transmit(void)
//Transmit bytes received from EP1OUT
{
      if (!(EP1OUTCS & 0x02))
      {
       if(i<bcl)
            {     SBUF0=EP1OUTBUF[i++];
                  dum=D5ON;
                  z^=1;
                  if (z)
                  {dum=D5OFF;}
            }
            else
            {
                  EP1OUTBC = 0x04;
            // Arms endpoint
            }
      }
      }
```

The BUSY bit (EP1OUTCS.2) indicates the status of the endpoint's OUT buffer EP1OUTBUF. The FX2LP USB core sets BUSY=0 when host data is available in the OUT buffer. The statement SBUF0=EP1OUTBUF[i++]; passes each byte in the EP1OUT buffer to SBUF0. TI is the transmit flag, which FX2LP hardware sets when a byte has been completely transmitted. Successful data transfer triggers ISR_USART0 ISR.

```
void ISR_USART0(void) interrupt 4
  {
      if (RI)
            {
      if((EP2468STAT & bmEP8EMPTY))
// check if EP8 is empty
            {
            RI=0;
            EP8FIFOBUF [0] = SBUF0;
// copies received data to SBUF0
            EP8BCH = 0;
            SYNCDELAY;
            EP8BCL = 1;
            SYNCDELAY;
            dut=D2ON;
            w^=1;
            if (w)
                  {
                  dut=D2OFF;
                  }
            }
            }
       if (TI)
            {
            TI=0;
            transmit();
            }
         }
```

The ISR_USART0 checks which condition caused the interrupt or the data reception or transmission, and proceeds accordingly. An RI interrupt copies incoming data to a buffer, and a TI interrupt calls the transmit() function to check data availability in the OUT buffer and transmits data if it is available.

4.3.2 DATA IN PATH

The following code fragment in `ISR_USART0 ISR` accomplishes device-to-host data transfer.

```
if (RI)
      {
      if((EP2468STAT & bmEP8EMPTY))
// check if EP8 is empty
            {
            RI=0;
            EP8FIFOBUF [0] = SBUF0;
// copies received data to SBUF0
            EP8BCH = 0; //Commit EP
            SYNCDELAY;
            EP8BCL = 1;
            SYNCDELAY;
            }
```

If a serial interrupt is triggered and the receive interrupt flag is asserted, the above code puts data received in SBUF0 into the EP8 endpoint buffer and commits it for transmission.

4.4 Communication Management

CDC defines class-specific requests and a notification component for device management. SET_LINE_CODING, GET_LINE_CODING, and SET_CONTROL_STATE are the class-specific requests. Their values are defined by the following codes:

```
#define SET_LINE_CODING (0x20)
#define GET_LINE_CODING (0x21)
#define SET_CONTROL_STATE (0x22)
```

The following code identifies the request type and sets asynchronous serial parameters or sends them to the host.

The SET_LINE_CODING *case* copies the control transfer data packet to a LineCode array and passes it to Serial0Init() function. This data packet contains the asynchronous serial parameters sent by the host. Table 3 shows its byte structure.

The `Serial0Init()` function parses the contents of the LineCode array and sets the corresponding baud rate.

```
switch(SETUPDAT[1])
   {
       case SET_LINE_CODING:
           Len = 7;
           EUSB = 0 ;
           SUDPTRCTL = 0x01;
           EP0BCL = 0x00;
           SUDPTRCTL = 0x00;
           EUSB = 1;
           while (EP0BCL != Len);
           SYNCDELAY;
           for (i=0;i<Len;i++)
           LineCode[i] = EP0BUF[i];
           Serial0Init();
           break;
```

The GET_LINE_CODING *case* copies LineCode contents to the EP0BUF and commits it. As per CDC specifications, the device is required to report default line coding settings in response to GET_LINE_CODING request from the host. Therefore, this case statement copies line coding data to *`EP0BUF`* from *`LineCode`* array and commits it.

```
   case GET_LINE_CODING:

           SUDPTRCTL = 0x01;
           Len = 7;
           for (i=0;i<Len;i++)
           EP0BUF[i] = LineCode[i];
           EP0BCH = 0x00;
           SYNCDELAY;
           EP0BCL = Len;
           SYNCDELAY;
       while (EP0CS & 0x02);
       SUDPTRCTL = 0x00;
       break;
```

Because the serial port of FX2LP doesn't have DTR or DTE pins, this application doesn't include SET_CONTROL_STATE.

```
case SET_CONTROL_STATE:
break;
```

4.5 Baud Rate Selection

The code below checks the host-sent baud rate in LineCode[0] and LineCode[1] and then sets that particular baud rate for the FX2LP UART0. The baud rates supported in the associated project are 2400, 4800, 9600, 19200, 28800, 38400, 57600, and 115200 bits per second.

The `Serial0Init()` function sets the FX2LP serial port to the baud rate requested by the host.

```
void Serial0Init() // serial UART 0 with Timer 2 in mode 1 or high speed baud rate generator
      {

          SCON0  = 0x5A;  //Set Serial Mode = 1, Recieve enable bit = 1
    T2CON  = 0x34;  //Int1 is detected on falling edge, Enable Timer0,Set Timer
overflow Flag

           if ((LineCode[0] == 0x60) && (LineCode[1] == 0x09 ))      // 2400
              {

                     RCAP2H = 0xFD;      //Set TH2 value for timer2
                     RCAP2L = 0x90;      //baud rate is set to 2400 baud

              }

          else if ((LineCode[0] == 0xC0) && (LineCode[1] == 0x12 ))        // 4800
              {
                     RCAP2H = 0xFE;      //Set TH2 value for timer2
              RCAP2L = 0xC8;      //baud rate is set to 4800 baud
      }

   else if ((LineCode[0] == 0x80) && (LineCode[1] == 0x25 ))        // 9600
              {
                     RCAP2H = 0xFF;      //Set TH2 value for timer2
              RCAP2L = 0x64;      //baud rate is set to 9600 baud
      }

   else if ((LineCode[0] == 0x00) && (LineCode[1] == 0x4B ))        // 19200
              {
                     RCAP2H = 0xFF;      //Set TH2 value for timer2
              RCAP2L = 0xB2;      //baud rate is set to 19200 baud
      }
          else if ((LineCode[0] == 0x80) && (LineCode[1] == 0x70 ))         // 28800
              {
                     RCAP2H = 0xFF;      //Set TH2 value for timer2
              RCAP2L = 0xCC;      //baud rate is set to 28800 baud
      }

   else if ((LineCode[0] == 0x00) && (LineCode[1] == 0x96 ))             // 38400
      {
              RCAP2H = 0xFF;      //Set TH2 value for timer2
              RCAP2L = 0xD9;      //baud rate is set to 38400 baud
      }
```

```
else if ((LineCode[0] == 0x00) && (LineCode[1] == 0xE1 ))          // 57600
              {
                     RCAP2H = 0xFF;       //Set TH2 value for timer2
              RCAP2L = 0xE6;       //baud rate is set to 57600 baud
       }

       else
              {
                  RCAP2L = 0xF3;
                  RCAP2H = 0xFF;
              }

       TH2 = RCAP2H; //Upper 8 bit of 16 bit counter to FF
       TL2 = RCAP2L; //value of the lower 8 bits of timer set to baud rate

        }
```

4.6 INF File

On a PC, virtual COM port enumeration requires an INF file. The INF file associates the VID/PID of the device to the default Windows COM port driver, usbser.sys. The INF file is included along with this application note.

5 Test Procedure

5.1 Software Requirements

- Cypress USB Control Center, which is part of the Cypress USB Suite. Use USB Control Center to download firmware into the FX2LP board.
- HyperTerminal or equivalent software (e.g., Tera Term) for the PC. This example uses Tera Term.

5.2 Hardware Requirements

The FX2LP code allows a PC running a terminal program to communicate with the FX2LP chip as if it were a COM port. To test the application, we need a way to send and receive text from the FX2LP chip. For this purpose, the serial port on the FX2LP Development Board makes a second connection to the PC. Open one Tera Term window to talk to the FX2LP COM port and a second Tera Term window to talk to the FX2LP Development Board serial port. When you have set up the connections in this way, you can send text back and forth between the two Tera Term windows.

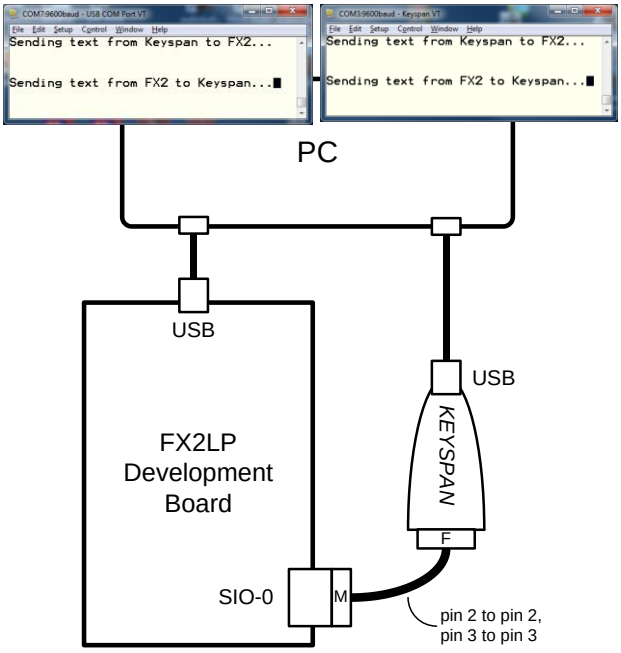
You also need the following hardware:

- An EZ-USB FX2LP Development Kit (CY3684).
- A PC with USB and UART ports. RS-232 ports are now difficult to find on a PC, but you can use a USB-to-serial converter. A good choice is the Keyspan 19HS, available through Amazon.
- A USB cable.
- A serial cable. For the Keyspan adapter, use a DB-9 male to DB-9 female, non-crossover cable (pin 2 to pin 2, pin 3 to pin 3).

5.3 Hardware Setup

Figure 5 provides a block diagram of the hardware test setup.

Figure 5. Hardware Test Setup



5.4 Firmware Download

1. On the FX2LP board, move the EEPROM Enable Switch (SW2) to the down position (No EEPROM). This enables the FX2LP to enumerate as a code loader. Then connect the FX2LP USB port to the PC.

   If you are using the FX2LP board for the first time, Windows will need to install a driver for the Cypress USB loader. Navigate to the Device Manager by clicking the Windows Start button, right-clicking Computer in the right-hand column, then selecting Properties. This opens the System screen. Select Device Manager at the top left column. Right-click “Cypress FX2 – No EEPROM (3.01.0000.2)” under “USB Serial Bus Controllers.” Then select “Update Driver Software”. Select “Browse my computer for driver software” and browse to…

   C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0

   \Drivers\cyusbfx1_fx2lp

   Select the folder corresponding to your operating system.

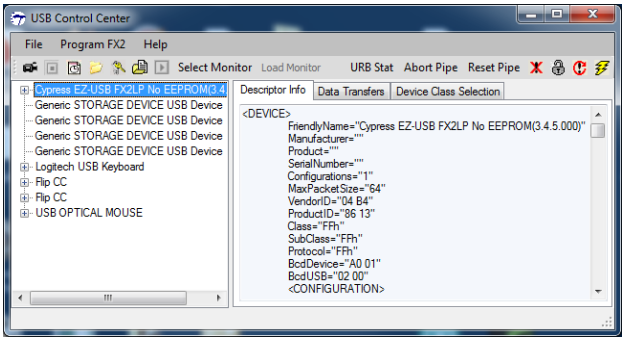
   The driver should install after you click Next. To confirm, check that the yellow exclamation point has disappeared from the Cypress USB entry.

2. The Windows firmware loader app, called CyControl.exe, is available at:

   C:\Cypress\Cypress Suite USB 3.4.7\
   CyUSB.NET\examples\Control Center\bin\Release

   Note that the Control Center folder includes source code for your inspection and modification.

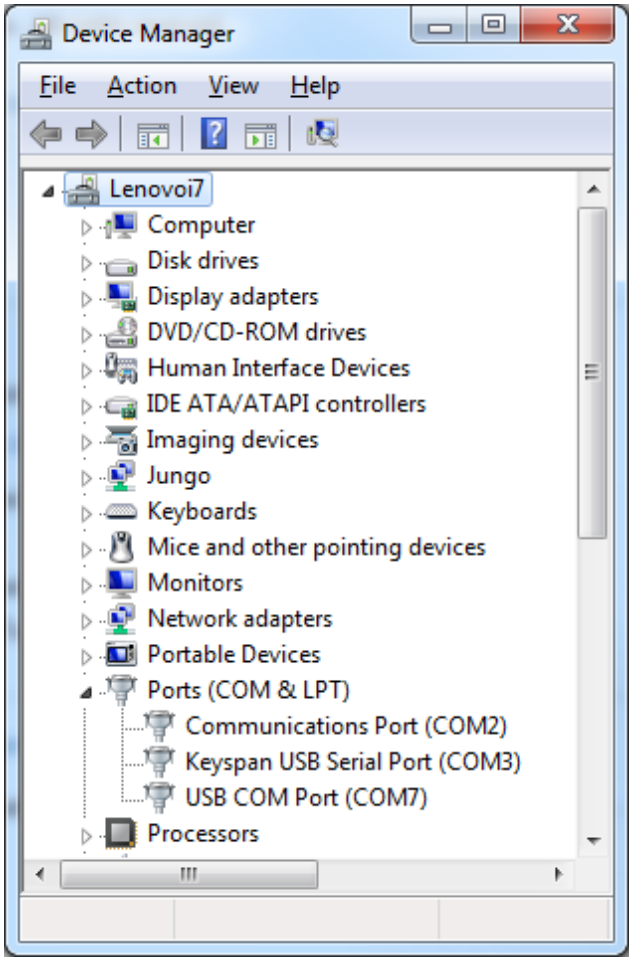
3. Start CyControl.exe, and you should see the following:



The left panel shows all USB devices connected to your PC. To simplify this panel so it shows only the Cypress FX2LP board, select the “Device Class Selection” tab and uncheck everything except “Devices served by the CyUSB.sys driver (or a derivative).” Now the left panel should show only the Cypress device.

4. Highlight the Cypress EZ-USB entry. Then select Program FX2, RAM. Navigate to the folder in which you installed the unzipped application note code. Then navigate to “CDT_VCP/Virtual COM Example Code/Output”; double-click the file VirtualCom.hex. This hex file is the result of a compile by the Keil tools used to create FX2LP application code. Refer to the application note AN42499 (Using and Troubleshooting the KEIL™ Debugger Environment) for more understanding on the Keil tools. The Keil project files are in the same directory as the .hex file. After this code loads, the FX2LP board disconnects from USB and reconnects as the virtual COM port.
5. **Note:** If you want to permanently load the hex file into the FX2LP board, select “Program FX2->64KB EEPROM” in step 4. After programming is complete, disconnect the FX2LP board from the USB, move both slide switches to the UP position (Enable Large EEPROM), and plug the FX2LP board back into the USB. This boot loads the hex file from EEPROM into the FX2LP chip every time power is applied.
6. The FX2LP board has changed identities—from a USB loader to the virtual COM device, as programmed by the .hex file you just loaded into it. Therefore, Windows needs a driver that is compatible with the virtual COM device. Follow the same driver installation procedure that you used in step 1, but browse instead to the folder in which this application note code is located, then to \CDT_VCP\INF file. Click the Next button to install the FX2LP virtual COM driver.
7. Locate the new FX2LP COM device by going back to the Device Manager and expanding the “Ports (COM & LPT)” item. A new serial device should appear as “USB COM Port (COMx).” Windows assigns the numeric digit ‘x.’
8. Connect the serial cable between the FX2LP board and the Keyspan adapter; then connect the Keyspan adapter to a USB plug on the PC. If the adapter does not enumerate, download the Keyspan driver and install it using the same method described above.

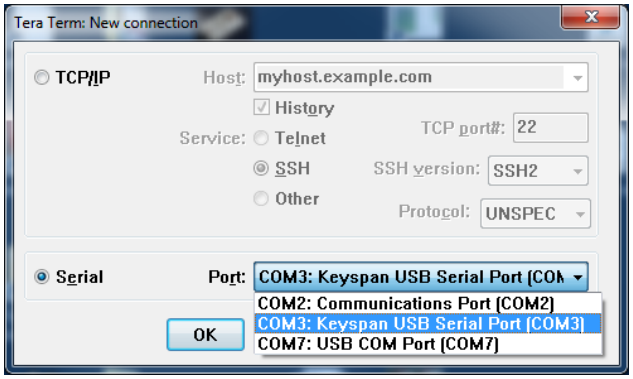
9. The “Ports…” entry should now show an additional serial device called “Keyspan USB Serial Port (COM3).”



5.5 Two Terminal Windows

Use TeraTerm to open two independent terminal windows, one for the Keyspan device and the other for the FX2LP COM device.

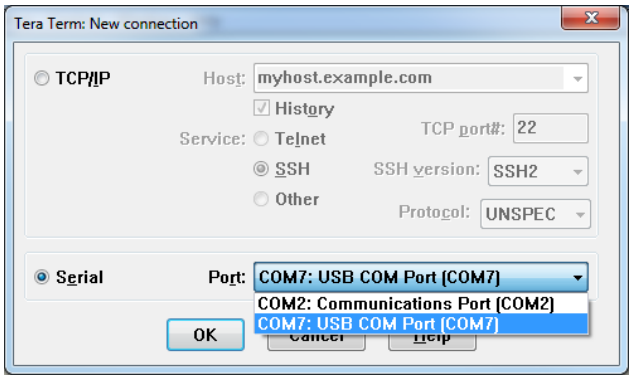
1. Start Tera Term and select the serial radio button. Then select the Keyspan COM port.



Note: Refer to Appendix A if neither of your new COM devices appears in the Port drop-down menu.

2. Select Setup, then Window, and change the title to “Keyspan.” This will help differentiate the two terminal windows.

3. Select Setup, then Terminal; select “New-line” for both Receive and Transmit to be “CR+LF.” Also check the “Local echo” box; this enables line feeds and lets you see what you are typing in the sending window.
4. Leave the Tera Term window open, and launch another instance of Tera Term. This time, select the FX2LP serial port:



5. Repeat steps 2 and 3, but label this terminal window FX2LP.
6. Now you can type in either Tera Term window and see the result in the other Tera Term window, as Figure 5 shows.

6 Other CDC Uses

This application note uses the CDC to implement a physical UART in the FX2LP chip that responds to COM traffic from the PC. You can also use the CDC class to communicate PC data to a custom application without servicing an actual UART. The Keil project is included with this application note code. By stripping out the code that manages the FX2LP UART, you can take advantage of the communication channel with the PC to design your own protocol. The advantage is on the PC side; any program that talks to a COM port can talk to your custom application.

7 Reference

Axelson, Jan. *Serial Port Complete.* 2nd ed. Madison, Wisconsin: Lakeview Research, 2007.

8 Summary

This application note explains serial port emulation on a USB port using the standard Windows driver. It provides an overview of the CDC class and shows how to implement it in FX2LP. This document includes a detailed explanation of firmware and a step-by-step procedure to test the application.

About the Authors

Names: Prajith Cheerakkoda.

Title: Applications Engineer

Appendix A. “Too Many” COM Ports

Some terminal programs don’t allow high COM port numbers—for example, COM17. The reason Windows assigns such high numbers is that many applications that create COM ports do not remove the port when the application is no longer running. You can clear out these unused COM ports as follows:

1. Start a Command Prompt window by selecting Start, All Programs, Accessories, and right-clicking Command Prompt.
2. Choose “Run As Administrator.”
3. In the DOS window, type: set devmgr_show_nonpresent_devices=1
4. In the DOS window, type: start devmgmt.msc. This brings up the Device Manager window.
5. In the View menu, select “Show hidden devices.”

Look at the Ports (COM & LPT) entry. You may be surprised to see many old COM ports. The ones not actually present are grayed out. To eliminate the ones you are no longer using, right-click each one and select Uninstall. This will give Windows a chance to pick lower COM numbers for the application.

Appendix B. Application Notes and Reference Designs

B.1 Application Notes

- AN65209- Getting started with FX2LP™

  AN65209 introduces you to the EZ-USB® FX2LP™ USB 2.0 device controller. This application note helps you build aproject for FX2LP and explore its various development tools, and then guides you to the appropriate documentation to accelerate in-depth learning about FX2LP.

- AN58009- Serial (UART) Port Debugging of EZ-USB® FX1/FX2LP™ Firmware

  AN58009 describes the code to be added to EZ-USB® FX2LP™ firmware for serial port debugging. This code enables you to print debug messages using the UART in the HyperTerminal program on a Windows computer or to capture them in a file.

- AN15456 - Guide to Successful EZ-USB® FX2LP™ and EZ-USB FX1™ Hardware Design and Debug

  This application note identifies possible USB hardware design issues, especially when operating at high-speed. It also facilitates the process of catching potential problems before building a board and assists in the debugging when getting a board up and running.

- AN50963 - EZ-USB® FX1™/FX2LP™ Boot Options

  This application note discusses the various methods to download firmware in to FX1/FX2LP.

- AN66806 - EZ-USB® FX2LP™ GPIF Design Guide

  This application note describes the steps necessary to develop GPIF waveforms using the GPIF Designer.

- AN61345 - Implementing an FX2LP™- FPGA Interface

  This application note provides a sample project to interface an FX2LP with an FPGA. The interface implements Hi-Speed USB connectivity for FPGA-based applications such as data acquisition, industrial control and monitoring, and image processing. FX2LP acts in Slave-FIFO mode and the FPGA acts as the master. This application note also gives a sample FX2LP firmware for Slave-FIFO implementation and a sample VHDL and Verilog project for FPGA implementation.

- AN57322 - Interfacing SRAM with FX2LP over GPIF

  This application note discusses how to connect the Cypress CY7C1399B SRAM to FX2LP using the General Programmable Interface (GPIF). It describes how to create read and write waveforms using GPIF Designer. This application note is also useful as a reference to connect FX2LP to other SRAMs.

- AN42499 - Setting Up, Using, and Troubleshooting the Keil Debugger Environment

  This application note is a step-by-step beginner's guide to using the Keil Debugger. This guide covers the serial cable connection from PC to SIO-1/0, the monitor code download, and required project settings. Additionally, it gives guidelines to start and stop a debug session, set break points, step through code, and solve potential problems.

- AN4053 - Streaming Data through Isochronous/Bulk Endpoints on EZ-USBR FX2 and EZUSB FX2LP

  This application note provides background information for a streaming application using the EZ-USB FX2 or the EZ-USB FX2LP part. It provides information on streaming data through BULK endpoints, ISOCHRONOUS endpoints, and high bandwidth ISOCHRONOUS endpoints along with design issues to consider when using the FX2/FX2LP in high-bandwidth applications.

- AN58069 - Implementing an 8-Bit Parallel MPEG2-TS Interface Using Slave FIFO Mode in FX2LP

  This application note explains how to implement an 8-bit parallel MPEG2-TS interface using the Slave FIFO mode. The example code uses the EZ-USB FX2LP at the receiver end and a data generator as the source for the data stream. Hardware connections and example code are included.

- AN58170 - Code/Memory Banking Using EZ-USB

  The EZ-USBFX2 family of chips contains an 8051 core. The 8051 core has 16-bit address lines and is able to access 64KB of memory. However, some applications require more than 64KB. This application note describes methods of overcoming this 64KB boundary.

- AN1193 - Using Timer Interrupt in Cypress EZ-USB FX2LP Based Applications

This application note helps EZ-USBR FX2LP firmware developers to use timer interrupts in their applications.

- AN63787 - EZ-USB® FX2LP™ GPIF and Slave FIFO Configuration Examples using 8-bit Asynchronous Interface
  This application note discusses how to configure the General Programmable Interface (GPIF) and slave FIFOs in EZ-USB FX2LP in both manual mode and auto mode to implement an 8-bit asynchronous parallel interface. This application note is tested with two FX2LP development kits connected back-to-back; the first one operating in master mode and the second operating in slave mode.

- AN61244 - Firmware Optimization in EZ-USB
  This application note describes firmware optimization methods in EZ-USB. Some of these methods are common to any processor and some are specific to the 8051 core of EZ-USB FX2LP.

- AN74505 – EZ USB FX2LP - Developing USB Application on MAC OS X using LIBUSB
  This application note describes a host application built on the MAC OS platform that uses libusb. The host application (Cocoa application) communicates with the BULK IN and BULK OUT endpoints ofFX2LP, using the interfaces provided by the APIs of libusb. This host application implements the transfer with devices that pass the particular VID/PID (=0x04B4/0x1004) identification.

- AN45471 - Vendor Command Design Guide for FX2LP
  This application note demonstrates how to code USB vendor commands to perform specific product. In addition, the note explains how to use the Cypress CyConsole utility to issue vendor commands.

B.2 Reference Designs

Several reference designs of FX2LP for popular applications are available. The reference designs include demonstration source code, reference schematics, and a BOM, where appropriate, for the design.

The reference designs available on the Cypress website are:

- CY4661 - External USB Hard Disk Drives (HDD) with Fingerprint Authentication Security
  The CY4661 reference design kit from Cypress and UPEK provides customers with a turnkey solution for an external USB hard disk drive (HDD), with fingerprint authentication, and security to protect and authenticate data. The reference design uses UPEK's Touch Strip Fingerprint Authentication Solution (TCS3 swipe fingerprint sensor and TCD42 security ASIC).

- FX2LP DMB-T/H TV Dongle reference design
  This reference design kit is based on the Cypress FX2LP and Legend Silicon’s chipset. A captured and demodulated RF signal converted to an MPEG2 TS stream by the Legend Silicon chipset is sent to the PC through an FX2LP. The PC plays these streams using a media player. This is a complete design, including all required files.

Document History

Document Title: AN58764 - Implementing a Virtual COM Port Using FX2LP™

Document Number: 001-58764

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2845294	PRKU	01/18/2010	New application note.
*A	3174314	SSJO	02/15/2011	Updates per review.
*B	3242361	SSJO	04/27/2011	Added information about 2400 baud rate. Updated code.
*C	3411330	PRJI	10/17/2011	Minor text edits. Updated template.
*D	3660417	PRJI	07/05/2012	Updated Introduction. Updated Communication Device Class Specification. Updated Firmware (Added EndPoints, Data Transfer, Communication Management, Baud Rate Selection). Updated Test Procedure. Updated Summary. Updated in new template.
*E	3894981	PRJI	02/27/2013	Updated firmware to implement interrupt driven data transfer.
*F	4088759	RSKV	8/6/2013	Updated Abstract Added more figures (Figures 1, 2, 3 and 5) Updated Test procedure Added Appendix A
*G	4293608	PRJI	02/27/2014	Updated in new template. Completing Sunset Review
*H	5196553	MVTA	04/01/2016	Added link for code examples Updated Table 1, Table 2 and Table 4 Added link for AN42499 in section 5.3 (firmware download)
*I	5582404	HPPC	03/28/2017	Updated template.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709