CY3689 EZ-USB FX2LP Discovery Kit Guide

About this document

Scope and purpose

The EZ-USB® FX2LP™ series (abbreviated as FX2LP) is a highly integrated controller family that serves as the basis for any USB high-speed peripheral device. The CY3689 EZ-USB FX2LP Discovery Kit (DVK) is designed to help beginners and experienced users to implement various applications using FX2LP microcontrollers.

Table of contents

Table of contents

1 Safety Information

1.1 Regulatory Compliance

The CY3689 EZ-USB FX2LP Discovery Kit is intended for use as a development platform for hardware or software in a laboratory environment. The board is an open-system design, which does not include a shielded enclosure. Due to this reason, the board may cause interference to other electrical or electronic devices in close proximity. In a domestic environment, this product may cause radio interference. In such cases, take adequate preventive measures. Also, do not use this board near any medical equipment or RF devices.

Attaching additional wiring to this product or modifying the product operation from the factory default may affect its performance and cause interference with other apparatus in the immediate vicinity. If such interference is detected, suitable mitigating measures must be taken.

	The CY3689 EZ-USB FX2LP Discovery Kit contains ESD-sensitive devices. Electrostatic charges readily accumulate on the human body and any equipment, and can discharge without detection. Permanent damage may occur on devices subjected to high-energy discharges. Proper ESD precautions are recommended to avoid performance degradation or loss of functionality. Store unused CY3689 EZ-USB FX2LP Discovery Kit in the protective shipping package.

	**End-of-Life/Product Recycling** This kit has an end-of-life cycle of five years from the year of manufacturing mentioned on the back of the box. Contact your nearest recycler for discarding the kit.

1.2 General Safety Instructions

1.2.1 ESD Protection

ESD can damage boards and associated components. Cypress recommends that you perform procedures only at an ESD workstation. If an ESD workstation is not available, use appropriate ESD protection by wearing an antistatic wrist strap attached to the chassis ground (any unpainted metal surface) on the board when handling parts.

1.2.2 Handling Boards

CY3689 EZ-USB FX2LP Discovery Kit boards are sensitive to ESD. Hold the board only by its edges. After removing the board from its box, place it on a grounded, static-free surface. Use a conductive foam pad if available. Do not slide the board over any surface.

Any physical action on the CY3689 EZ-USB FX2LP Discovery Kit such as changing wires, jumper settings, or measuring voltages can cause stress on the CY3689 printed circuit board assembly (PCBA) and Micro USB connectors. You must ensure that the PCBA has proper support on the bottom side to avoid stress on the PCBA when the DVK is in operation.

2 Introduction

The EZ-USB® FX2LP™ series (abbreviated as FX2LP) is a highly integrated controller family that serves as the basis for any USB high-speed peripheral device. To take full advantage of the USB 2.0 480-Mbps signaling rate, FX2LP contains specialized hardware to buffer USB data and to connect seamlessly to a variety of high-bandwidth external devices such as MCUs, ASICs, and FPGAs. The CY3689 EZ-USB FX2LP Discovery Kit (DVK) is designed to help beginners and experienced users to implement applications such as USB2 Mobile add-on digital cameras, thermal cameras, logic analyzers, and multiple industrial applications using FX2LP microcontrollers. FX2LP DVK consists of the FX2LP device with EEPROM module, power subsection, micro-USB connector, and interface connectors for add-on modules.

This user guide describes the CY3689 EZ-USB FX2LP Discovery Kit (DVK), giving detailed instructions that enable you to do the following:

- Understand the DVK board and its features
- Test simple firmware projects to familiarize with the kit
- Understand the structure of any FX2LP application, including the firmware framework
- Load and debug code

2.1 Kit Contents

The CY3689 EZ-USB FX2LP Discovery Kit (DVK) contains the following items:

- CY3689 EZ-USB FX2LP Discovery Kit (DVK)
- Quick Start Guide

2.1.1 Hardware Not Included with the Kit

- Micro-USB cable

3 Getting Started

This chapter describes how to get started with the CY3689 EZ-USB FX2LP Discovery Kit. Section **3.1** explains the procedure to download and install the FX3 SDK and FX2LP DVK package required for the kit. FX3 SDK includes the latest version of drivers and host applications for the Discovery Kit. The Keil uVision IDE and GPIF Designer software for FX2LP application development are included in the FX2LP DVK package. Section **3.2** explains the steps to download firmware examples related to the Discovery Kit. Section **3.3** briefly explains the steps for setting up the board and installing the required drivers on the host PC.

3.1 SDK Installation

FX2LP shares the same Software Development Kit (SDK) as FX3, which comes with tools, drivers, and application examples. These help to accelerate application development using FX2LP and FX3 devices. The SDK provides application development and debug support on Windows, Linux, and MacOS platforms. SDK installation is mandatory for first time users, since it provides all the required utilities, tools, and system files.

1. Download the latest version of FX3 SDK Setup from **EZ-USB FX3 Software Development Kit**. The SDK setup includes host applications and drivers required for CY3689.
2. Run the downloaded setup file and InstallShield Wizard. The default installation location will be displayed in the tool; if required, change the installation folder. Click **Next**.

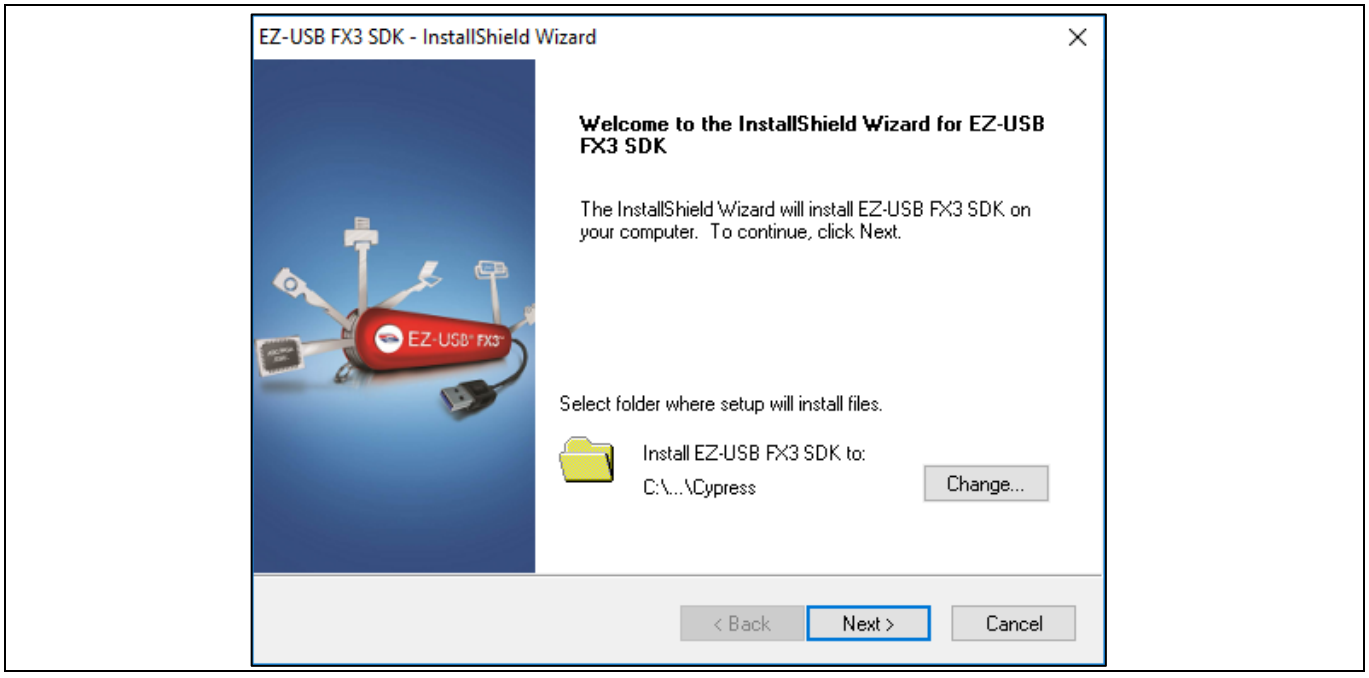


Figure 1 FX3 SDK Setup Installation - Installation Folder Selection

3. Choose Installation type as **Typical** and click **Next**.

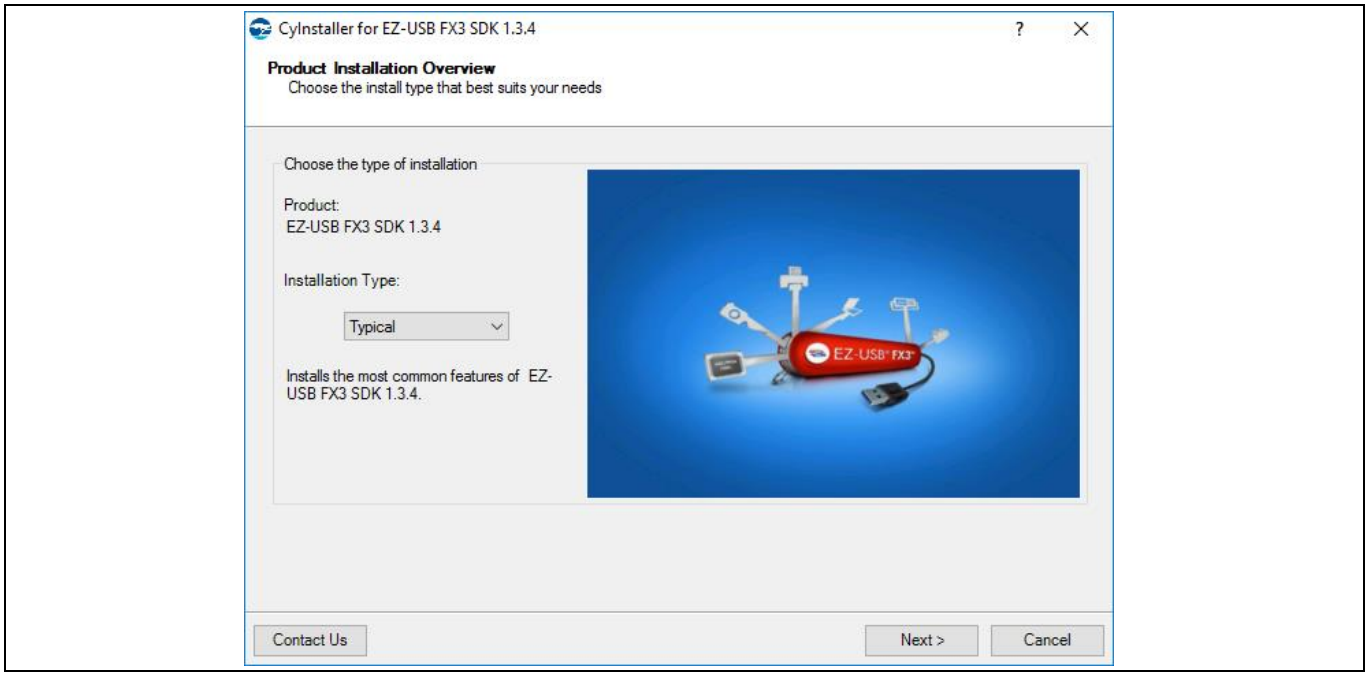


Figure 2 FX3 SDK Setup Installation- Installation Type Selection

4. Accept the **Cypress End User License Agreement** and click **Next** to continue installation

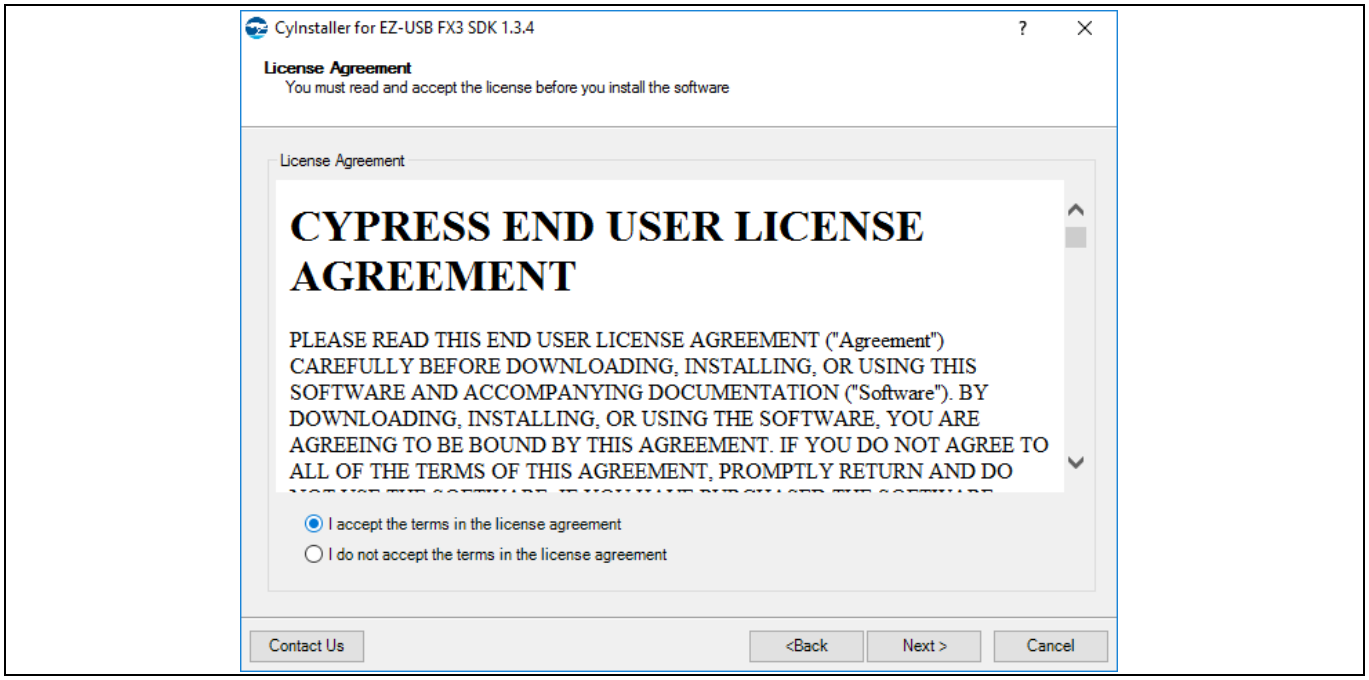


Figure 3 FX3 SDK Setup Installation-Cypress End User License Agreement

5. Accept the **Eclipse Public License Agreement**, which is required for installing Eclipse tools and IDEs. Click **Next** to continue installation.

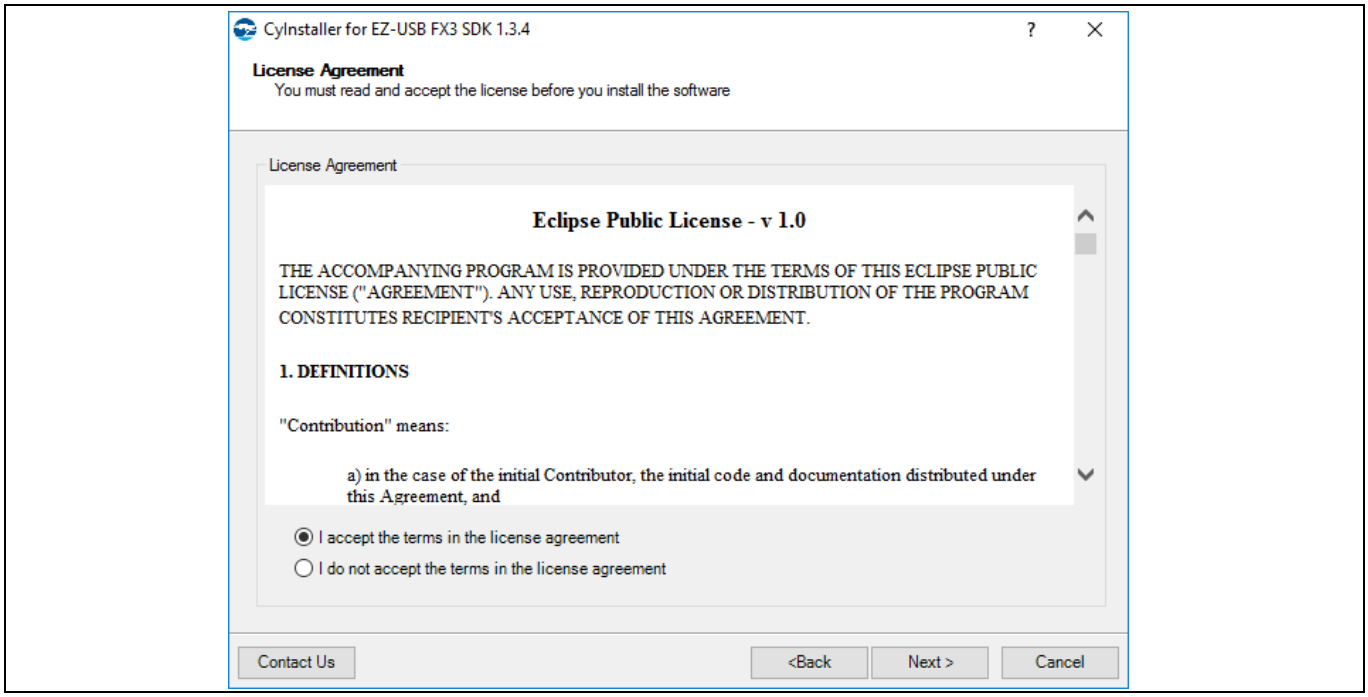


Figure 4 FX3 SDK Setup Installation- Eclipse Public License Agreement

6. Accept **Eclipse Foundation Software User Agreement** and click **Next**.

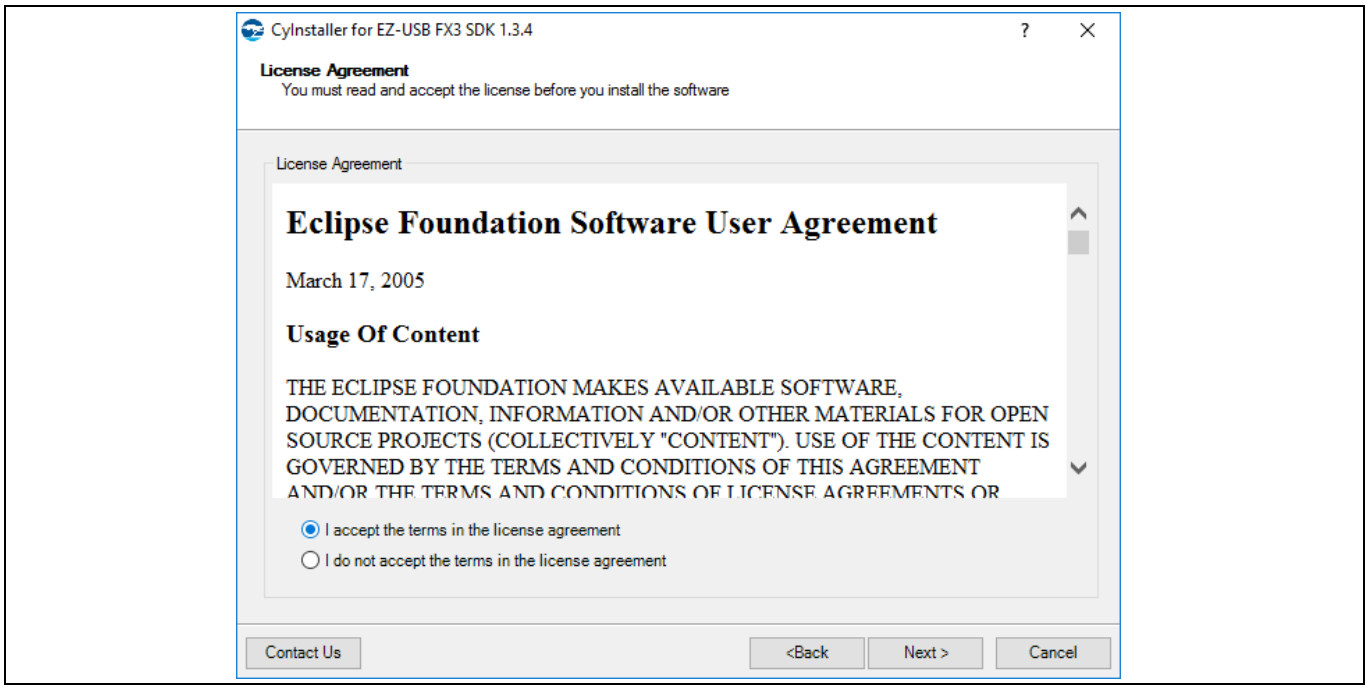


Figure 5 FX3 SDK Setup Installation- Eclipse Foundation Software User Agreement

7. The installation screen will display as in **Figure 6**. The panel on the left of the installation window shows the tools, software, and related files installed with the SDK.

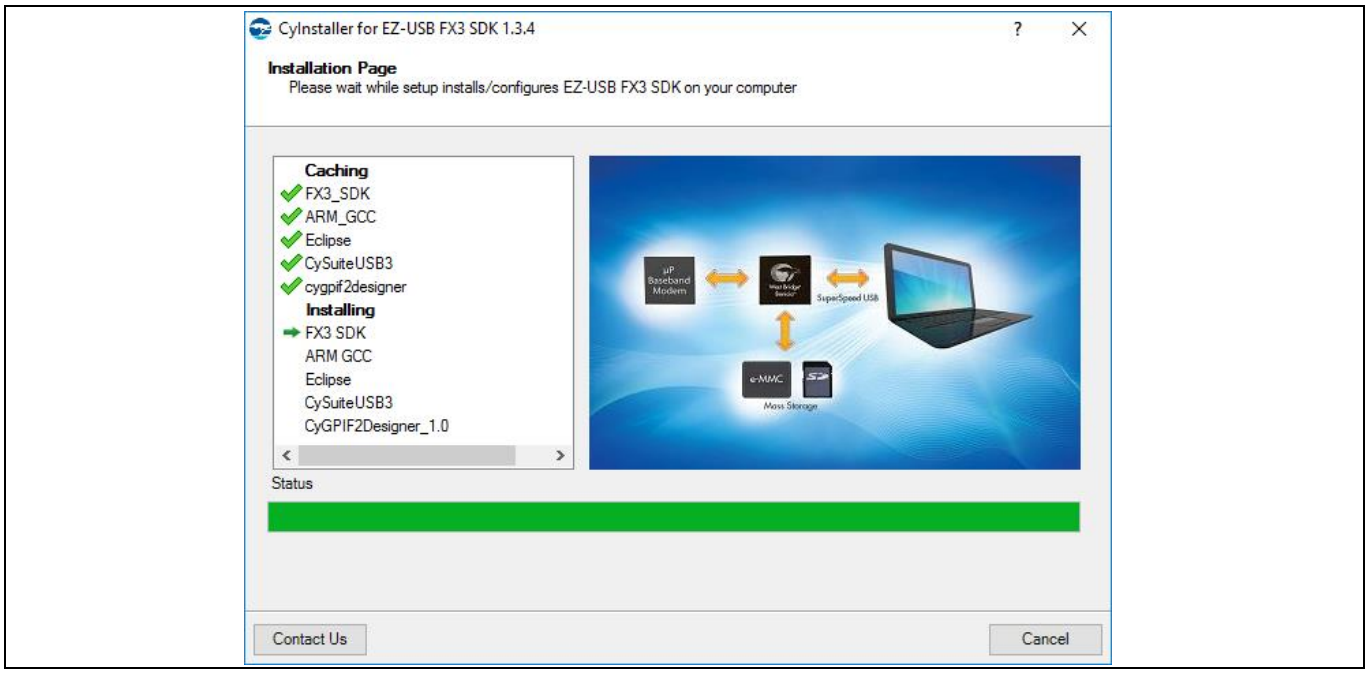


Figure 6 FX3 SDK Setup Installation

8. Upon successful SDK installation, the page shown in **Figure 7** will be displayed. Click **Finish** to finalize the installation.

Figure 7 FX3 SDK Installation Completed

9. Download and install the latest version of FX2LP DVK setup from **EZ-USB FX2LP Development Kit**. The FX2LP DVK package installation includes Keil uVision IDE and GPIF Designer, required for firmware development.

10. Run the downloaded setup file and InstallShield Wizard. The installation wizard displays the CY3684 EZ USB FX2LP DVK. CY3684 and CY3689 are FX2LP development kits that share a common development plaform, and CY3684 DVK installation includes the FX2LP DVK package required for CY3689 also. The default installation location will be displayed in the tool; if required, change the installation folder. Click **Next**.

Figure 8 FX2LP SDK Setup Installation - Installation Folder Selection

11. Choose Installation type as **Typical** and click **Next**.

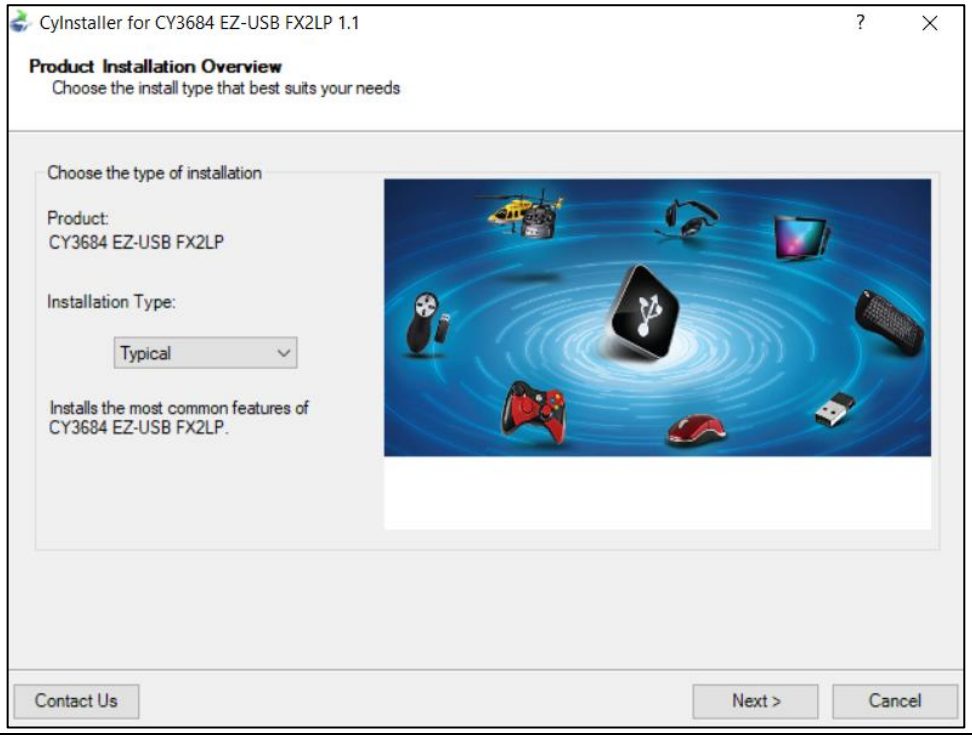


Figure 9 FX2LP SDK Setup Installation - Installation Type Selection

12. Accept the **Cypress End User License Agreement** and click **Next** to continue installation

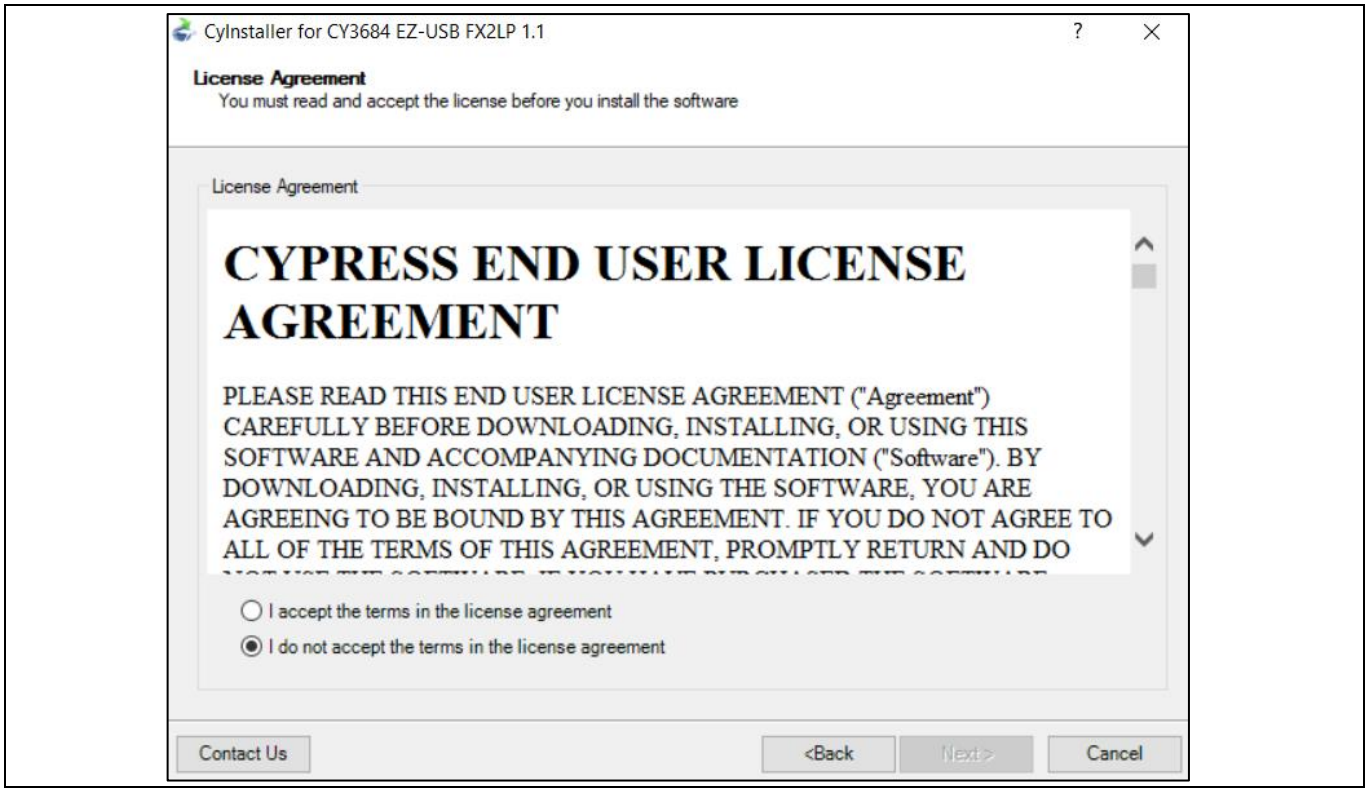


Figure 10 FX2LP SDK Setup Installation-Cypress End User License Agreement

13. The Installation Page will display as in **Figure 11**.

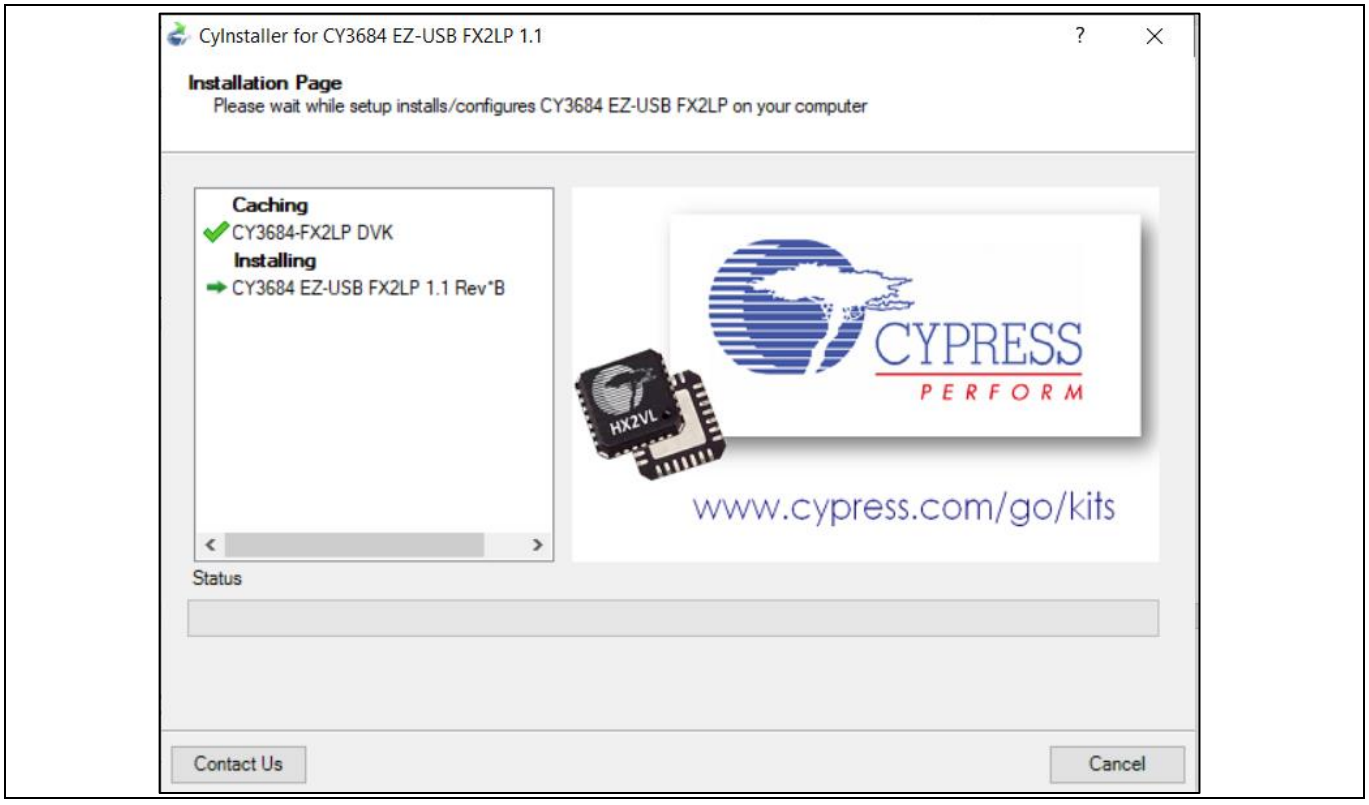


Figure 11 FX2LP SDK Setup Installation

14. uVision installation setup will begin automatically on successful completion of step 13. Click **Next** to continue installation.

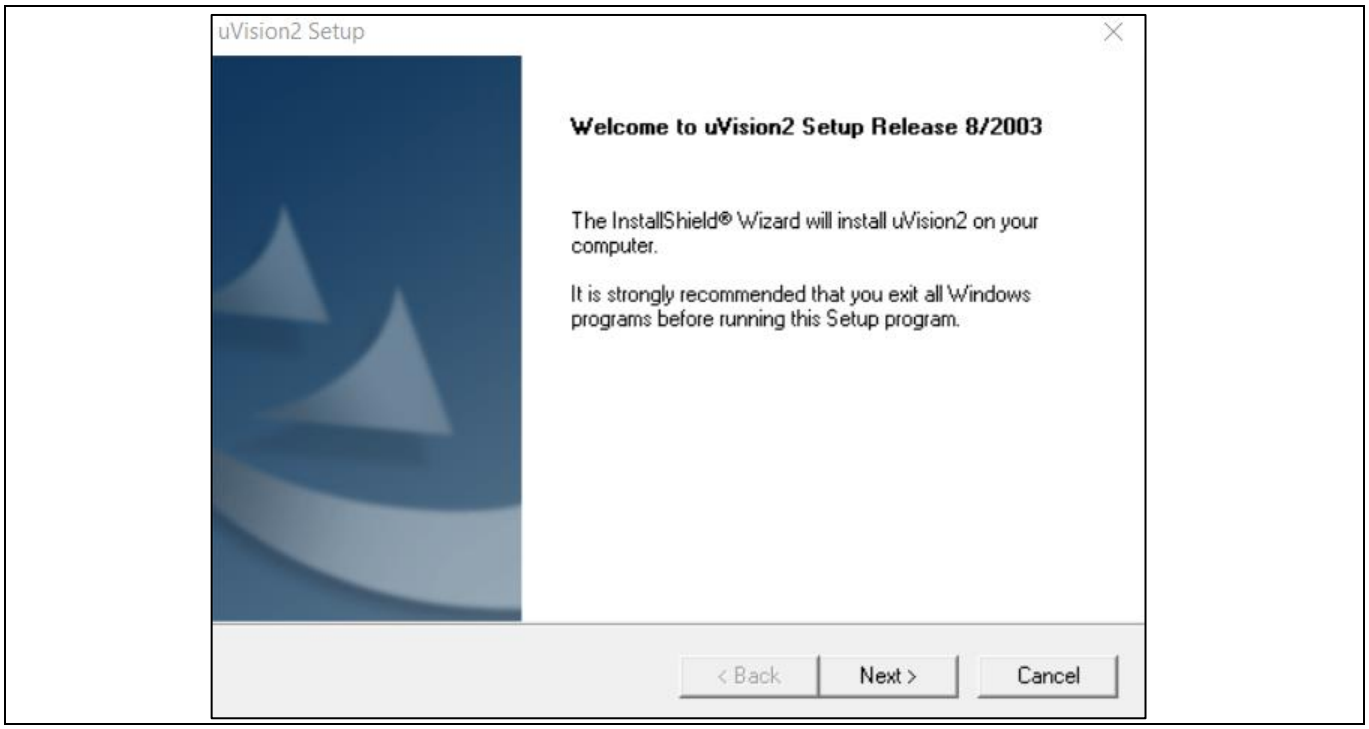


Figure 12 uVision Installation - Welcome Screen

15. Click **Yes** to accept the Keil software end-user license agreement

Figure 13 Keil Software End User License

16. Restrictions of evaluation versions are listed on the installation window. The uVision included in the SDK has a 4kB code size restriction. For code sizes greater than 4KB, a licensed version of uVision is required. Click **Next** to continue installation. Refer to section **6.7.3** for using a free SDCC compiler that does not have a limit on code size.

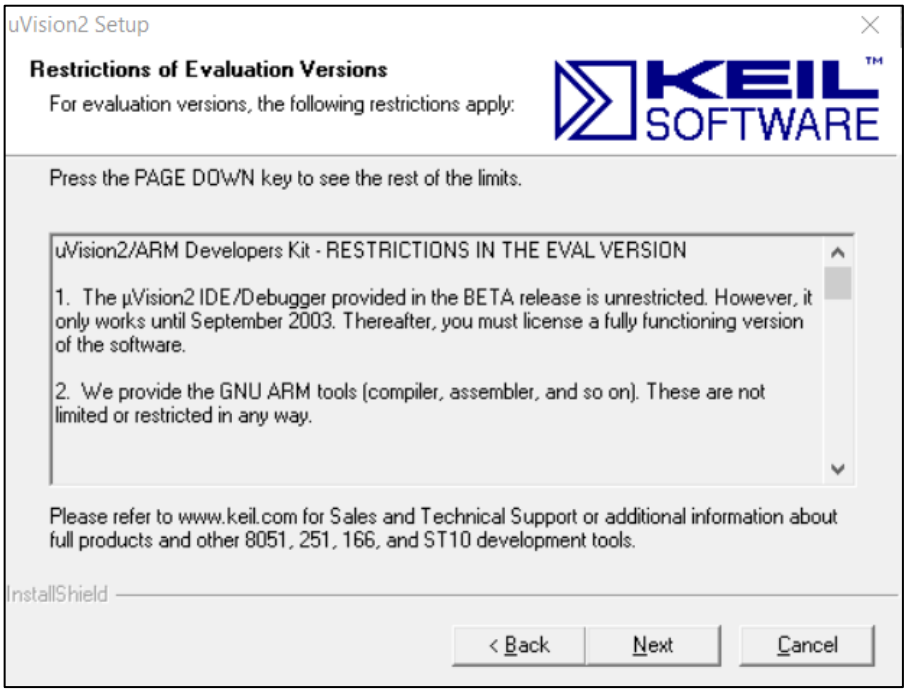


Figure 14 Restrictions of Evaluation Version Window

17. Choose the installation folder and click **Next**. The default installation location is ***C:\Keil***.

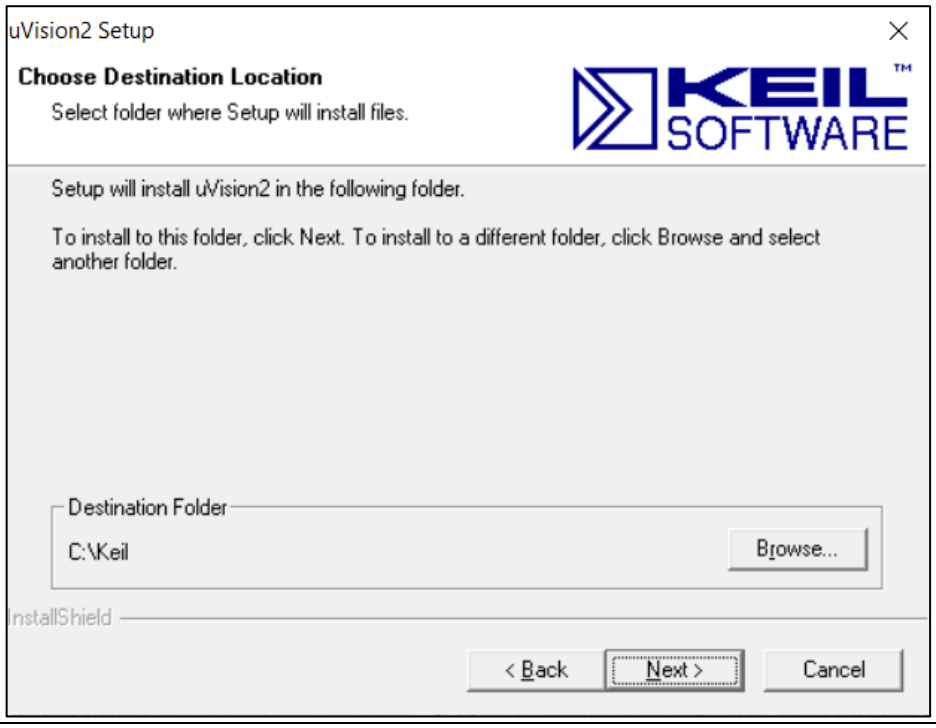


Figure 15 uVision Installation Folder Selection

18. Enter **User Name** and **Company Name** in the Customer Information window and click **Next**.

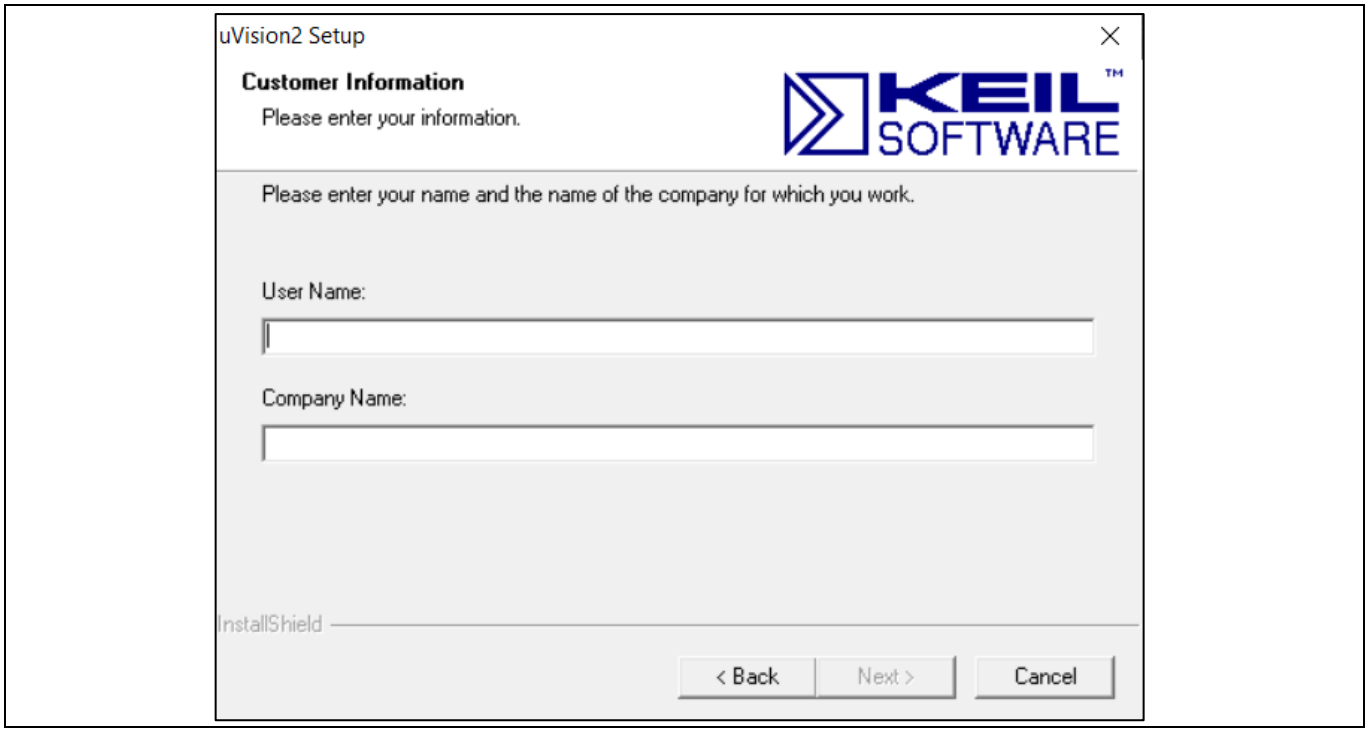


Figure 16 Customer Information Window

19. Review settings for copying files in the Start Copying Window and click **Next**.

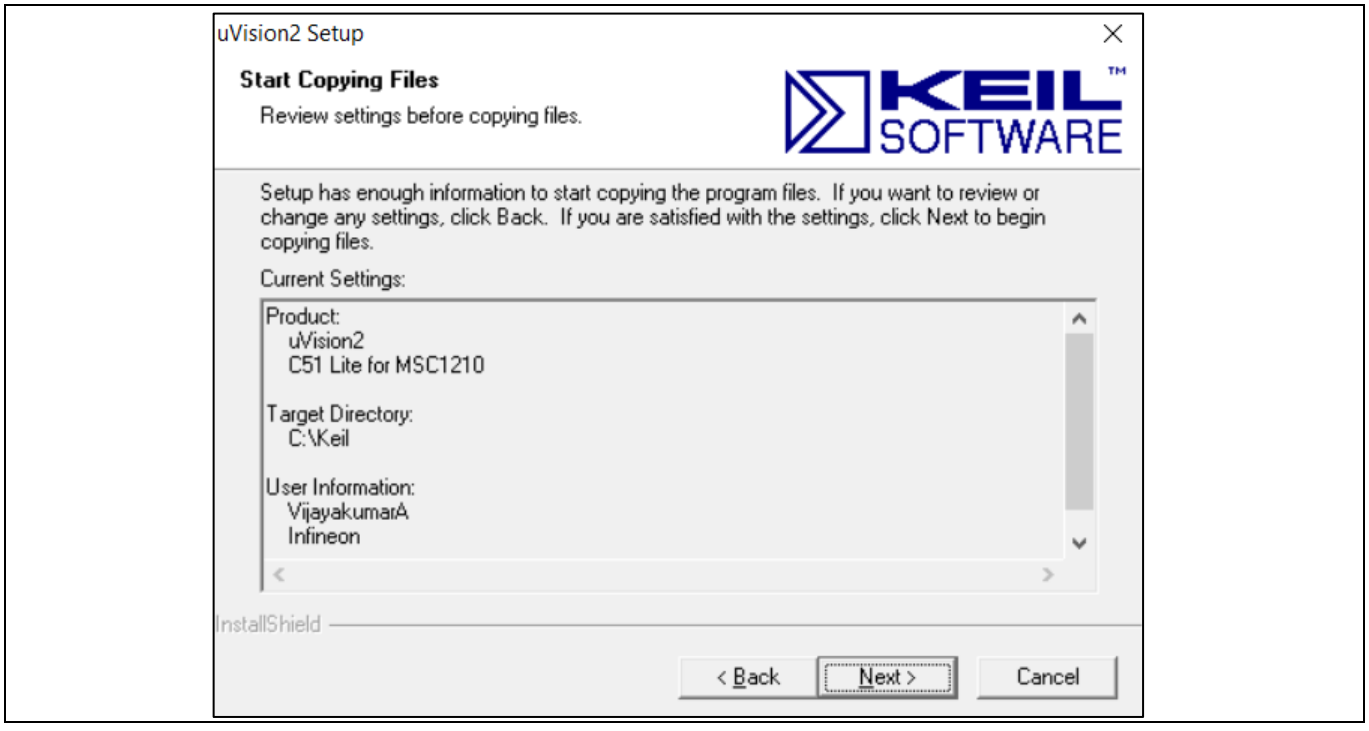


Figure 17 Start Copying Files Window

20. Installation will start and Setup Status will display the progress

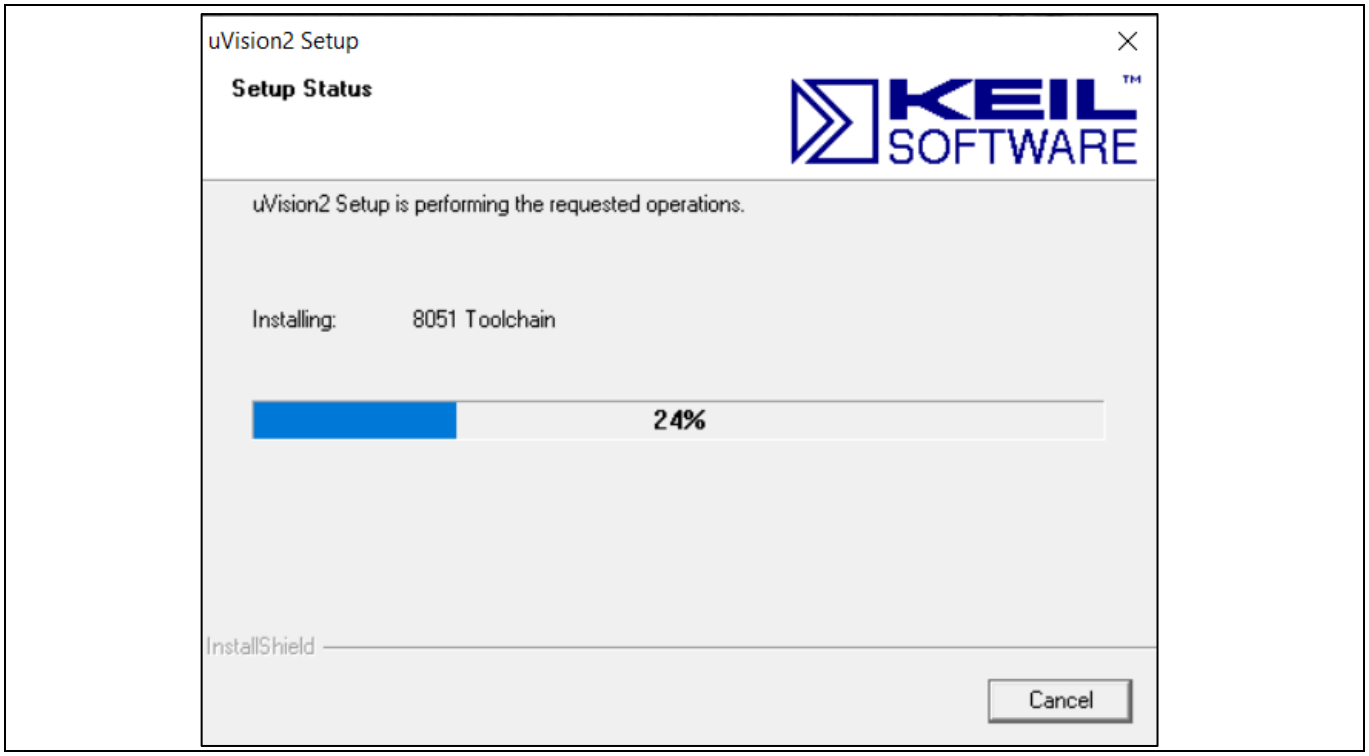


Figure 18 uVision Installation-Setup Status

21. Successful installation displays **InstallShield Window Complete** window. Click **Finish** to complete the uVision installation.

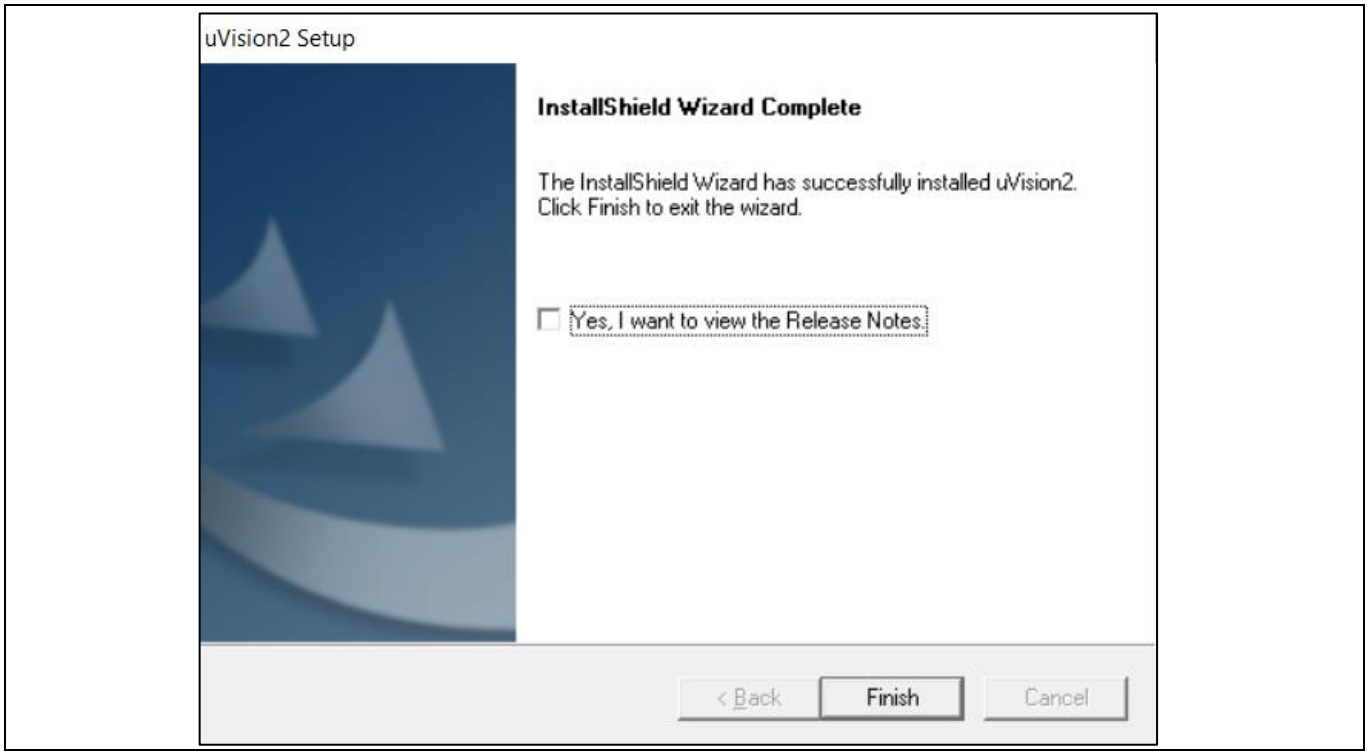


Figure 19 InstallShield Window Complete

22. Cypress GPIF Designer installation automatically starts after step 21. Click **Next** on the GPIF Designer Installshield welcome window.

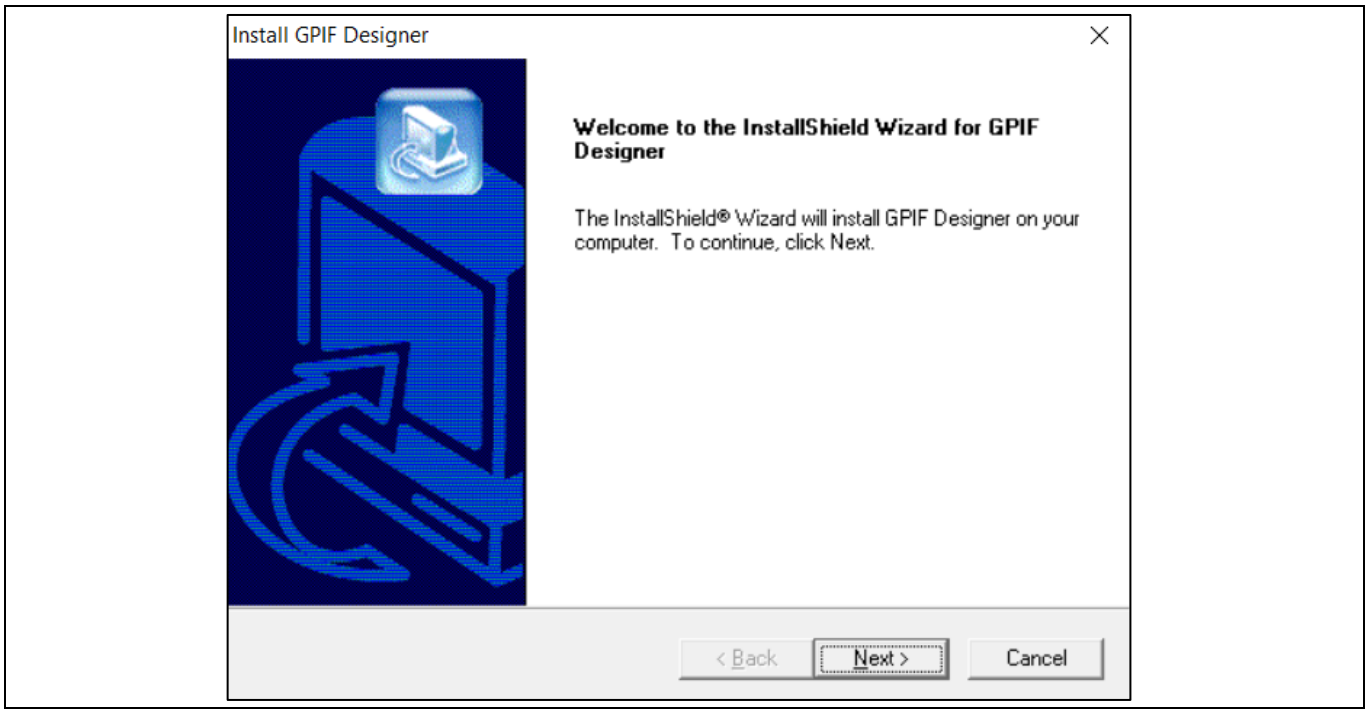


Figure 20 GPIF Designer InstallShield Welcome Window

23. Accept the license agreement and click **Next**. Choose the installation folder and click **Next** to continue installation.

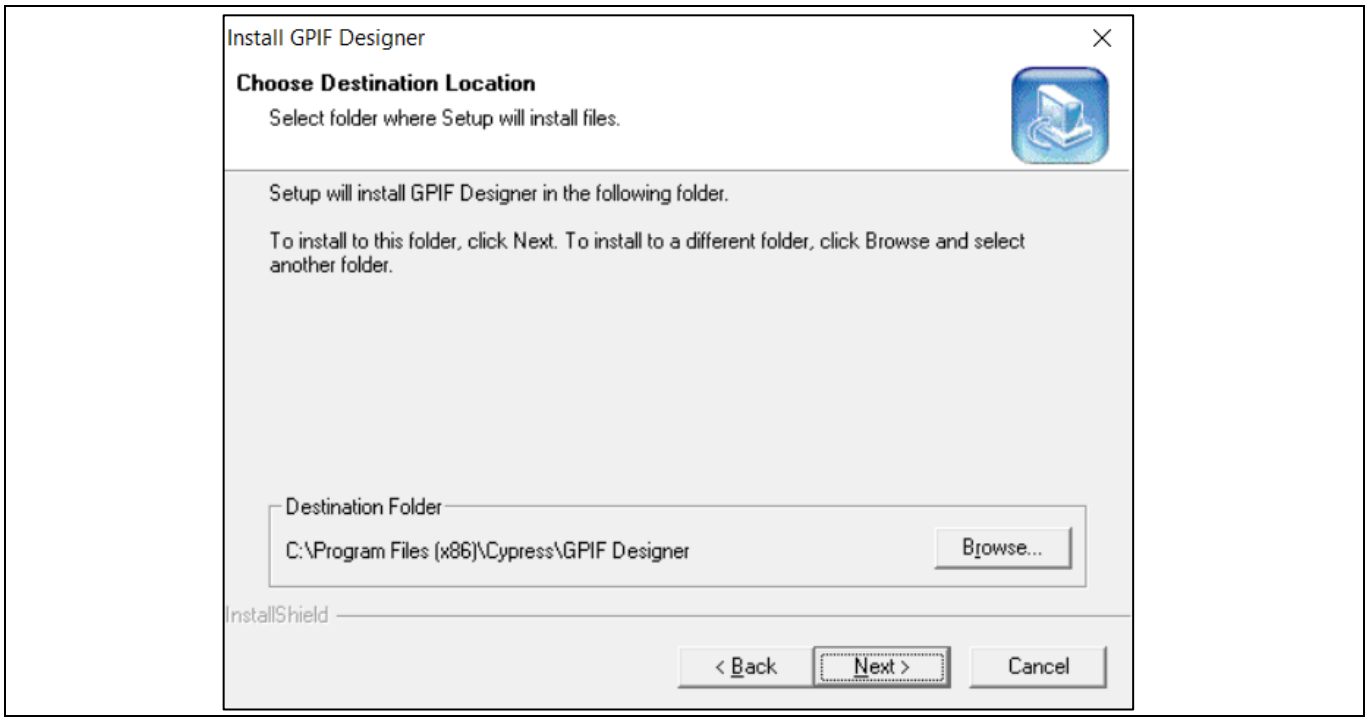


Figure 21 GPIF Designer-Installation Folder

24. Click **Next** in the window that appears to start installation. Successful installation displays InstallShield Wizard Complete window. Click **Finish** to complete the installation.

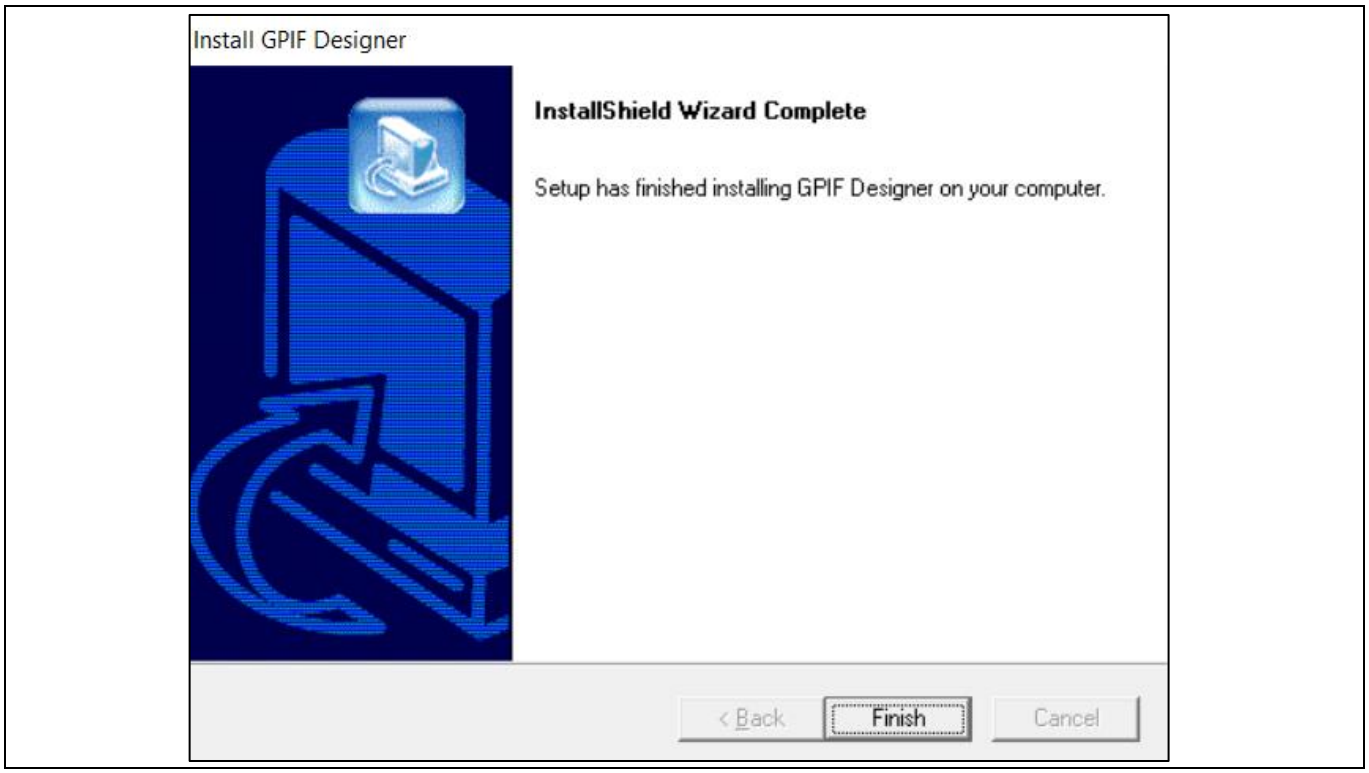


Figure 22 InstallShield Wizard Complete-GPIF Designer

3.2 Downloading Example Projects

The example projects help to understand the capabilities of the CY3689 FX2LP Discovery Kit and can be used as a reference for new application development using the Discovery Kit.

The example projects are downloadable from the CY3689 FX2LP Discovery Kit website in zip format. Visit CY3689 FX2LP Discovery Kit page or click **http://www.cypress.com/cy3689** to download the example projects. Details of example projects are explained in the **Firmware Examples** section. The following firmware examples are included in example projects.

3.2.1 Bulkloop Example

This project illustrates the configuration of FX2LP to accept bulk data from the host and loop it back to the host. Bulk packets received on an OUT endpoint are looped back to an IN endpoint. FX2LP double buffering is demonstrated. This project is explained in the **Bulkloop Example** section.

3.2.2 Bulksrc

FX2LP provides an endless source and sink of the USB bulk data. This program demonstrates how to service bulk endpoints and manage endpoint buffers. Refer to the **Bulksrc Firmware Examples** section for detailed description of the project.

3.2.3 EP_Interrupts

This is the bulkloop firmware that uses interrupts instead of polling status bits. It demonstrates the use of endpoint interrupts. Detailed explanation of the example project is included in the **EP_Interrupts Example** section.

3.2.4 IBN Example

This project performs bulk loopback of EP2OUT to EP6IN and EP4OUT to EP8IN using the IN-BULK-NAK (IBN) interrupt to initiate the transfer. Refer to the **IBN Firmware Example** section for the project description.

3.2.5 Pingnak

This project performs bulk loopback of EP2OUT to EP6IN and EP4OUT to EP8IN using the PINGNAK interrupt to initiate the transfer. Refer to the **Pingnak Firmware Example** section for a detailed description.

3.2.6 Vend_ax

The USB specification provides a mechanism called “vendor requests” to create your own custom USB commands. This project shows how to implement these requests. Application note **AN45471** contains detailed information. Refer to the **Vend_ax Firmware Example** section for the description.

3.3 Prepare FX2LP Discovery Kit

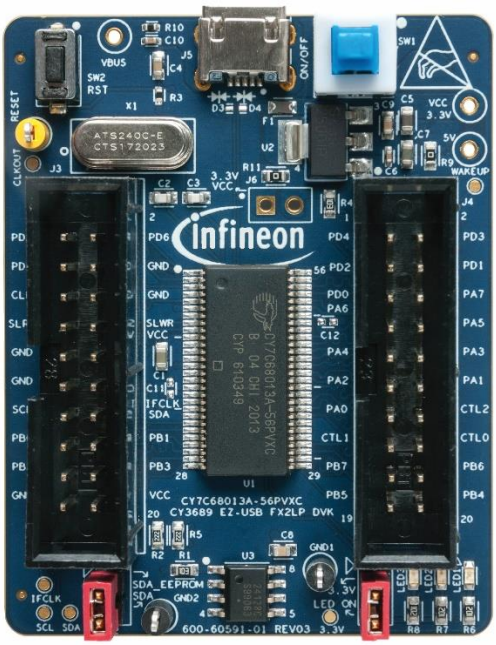


Figure 23 CY3689 EZ-USB FX2LP Discovery Kit

1. Check whether Jumper J1 and J2 are populated. If they are not, populate the jumpers. The functionality of the jumpers is explained in **Jumper Settings**. SW1 should be in the ON position (pressed).
2. Connect the CY3689 EZ-USB FX2LP Discovery Kit to the PC/Desktop through Micro-USB-to-A cable. The board is powered through the J5 (Micro-USB) connector. Power LED “LED1” turns-on, indicating that the Discovery Kit is powered and the power supply regulator is working.
3. If this is the first time you have plugged it in, you should see pop-up messages to install a USB driver. Navigate to the driver folder and select the subfolder corresponding to your Windows OS version.
4. Open the **Device Manager**; the device will be listed as **Cypress FX2LP No EEPROM Device** on successful installation of driver.

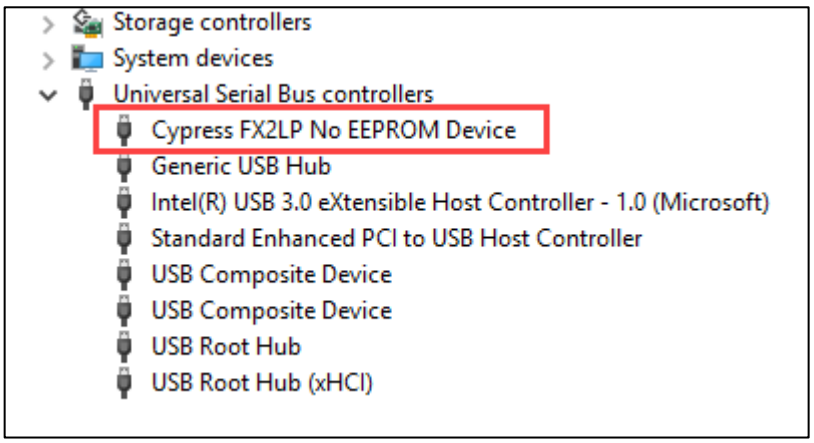


Figure 24 Control Panel

3.4 Binding Driver Manually

If automatic driver binding failed for some reason, follow this procedure to bind the driver manually.

1. Open Windows **Device Manager** and locate **Unknown device** under Other devices

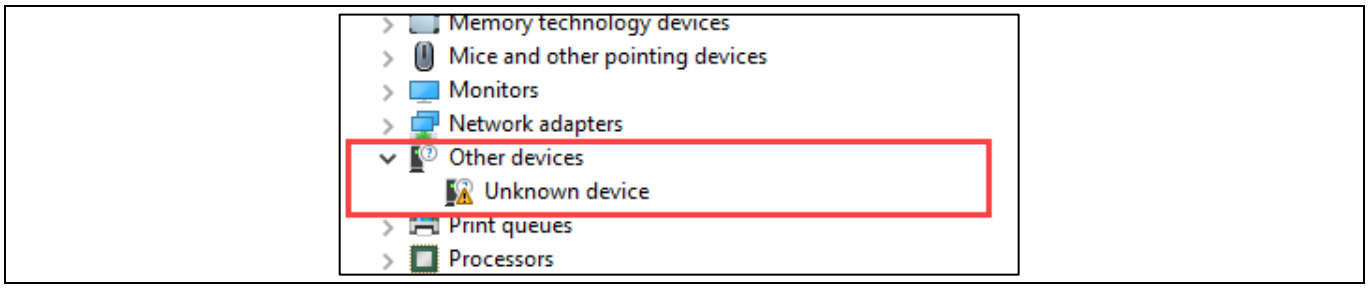


Figure 25 Manual Driver Binding-Step 1

2. Right-click on **Unknow device** and select **Update Driver Software…**

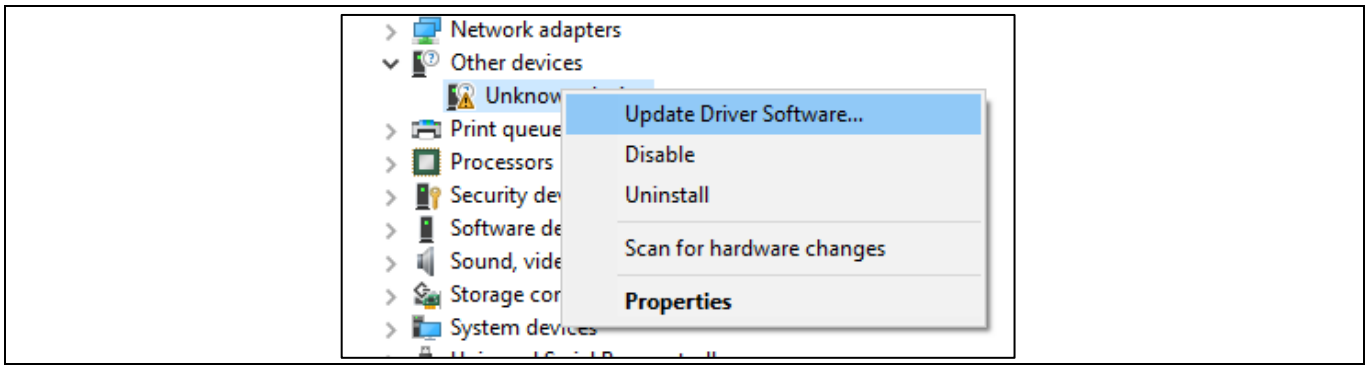


Figure 26 Manual Driver Binding-Step 2

3. Select **Browse my computer** for driver software and locate the driver corresponding to the operating system from *<FX3 SDK Installation Directory>\driver\bin*. This is the same installed directory chosen in step 2 of section **3.1**. Select a driver based on the operating system. For example, for Win 10 (64-bit), select **Win10\x64**.

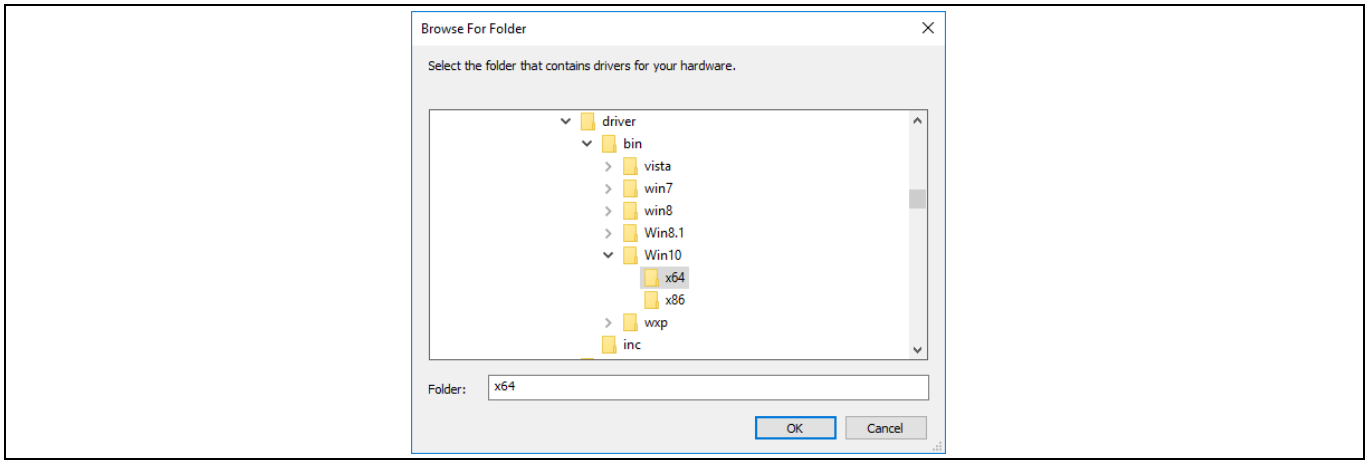


Figure 27 Manual Driver Binding-Step 3

4. Click **OK** and press **Next** as in **Figure 27**. The operating system acknowledges successful driver installation.

4 CY3689 EZ-USB FX2LP Kit Hardware

4.1 CY3689 EZ-USB FX2LP Discovery Kit Architecture

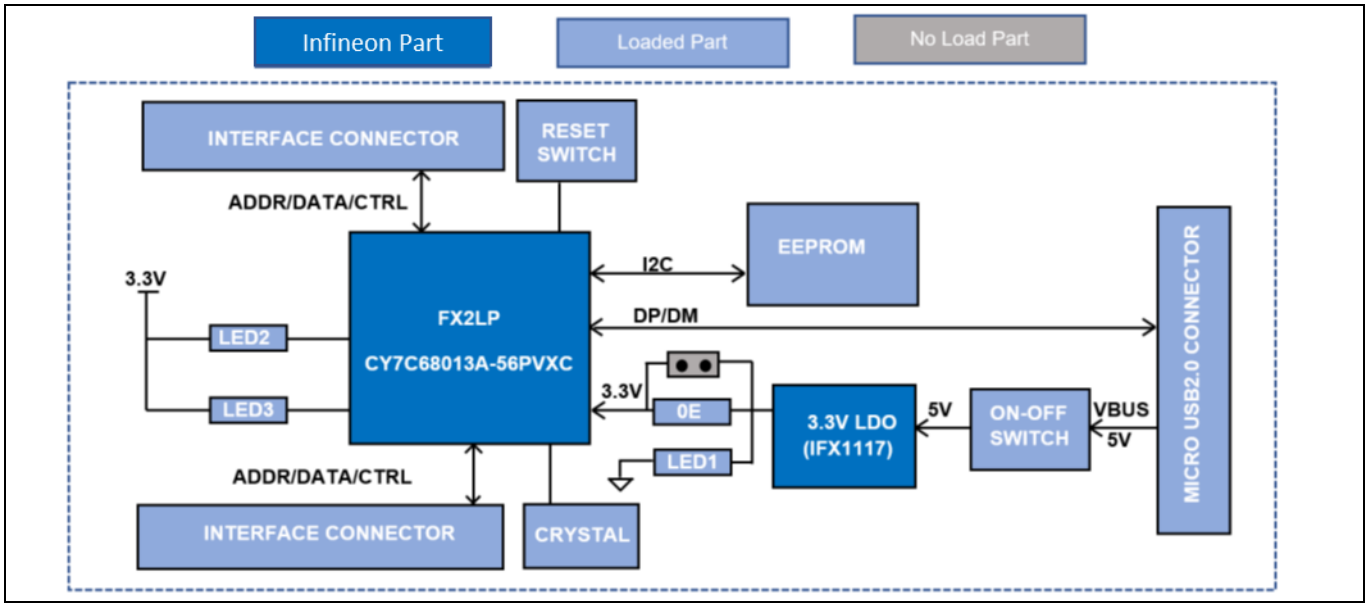


Figure 28 CY3689 EZ-USB FX2LP Discovery Kit System-level Block Diagram

The Infineon CY7C68013A-56PVXC device, referred to as FX2LP in this document, is a low-power, highly integrated USB 2.0 microcontroller in the FX2LP High-Speed USB Peripheral Controller family.

Infineon IFX25001 is a monolithic integrated fixed NPN type voltage regulator, which regulates the device input voltage to 3.3 V from 5 V input available from the micro-USB connector.

U3 is the onboard 128Kb I2C EEPROM to store firmware image. FX2LP accesses EEPROM through the I2C interface.

CY3689 is powered through a micro-USB connector, J5. The kit requires the USB port to program the device and also for interfacing the kit with the USB host controller on the PC/Desktop.

Three onboard LEDs are available, one power LED and two firmware/application driven LEDs. Power LED (LED1) is connected to the output of the 5 V-to-3.3 V regulator. LED2 and LED3 are firmware-controlled LEDs, connected to the GPIOs and the user can configure the functionality.

GPIF connectors J3 and J4 help to mount add-on boards/module to extend the functionality, and allow development of various applications using the kit. Various signals required to interface the FX2LP kit with accessory boards, add-on modules, and FPGAs are routed to the onboard GPIF interface connectors.

4.2 CY3689 EZ-USB FX2LP Discovery Kit Connectors/Jumpers

Figure 29 shows the CY3689 EZ-USB FX2LP Discovery Kit with the interface connectors, jumpers, and critical components.

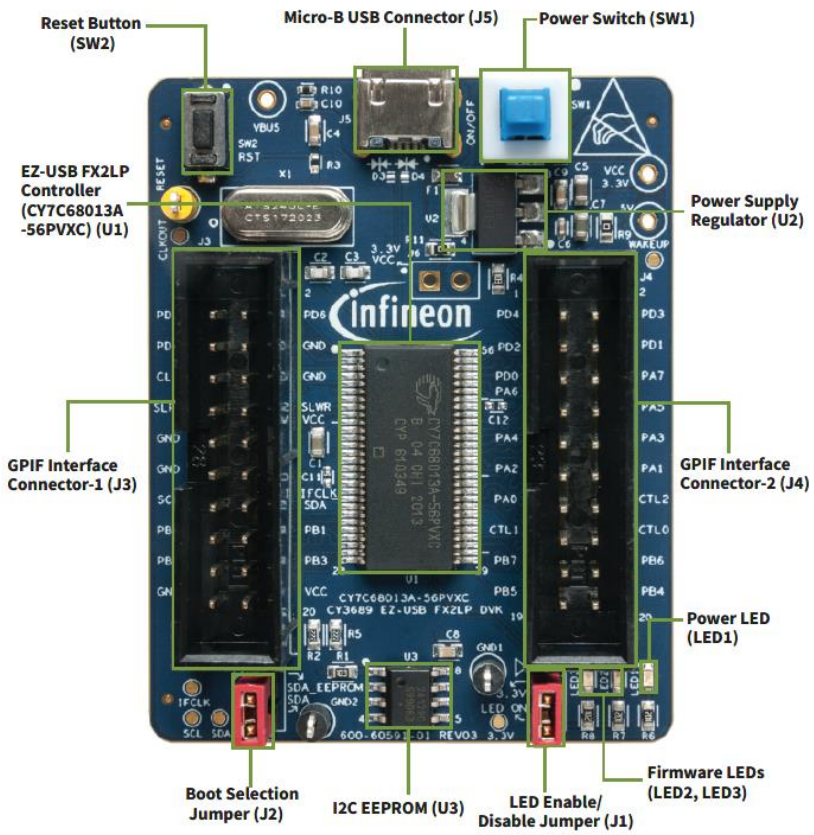


Figure 29 CY3689 EZ-USB FX2LP Discovery Kit

4.2.1 Jumper Settings

J1 and J2 are 2-pin jumpers mounted on the board. **Table 1** explains their functionalities.

Table 1 Jumper Settings

Jumper	Function	Default Setting
J1	Enables or disables the application/firmware LEDs (LED1 and LED2) J1 Closed: Enable LEDs J1 Open: Disable LEDs or disconnect LEDs from VCC	Closed
J2	Boot selection jumper J2 Closed: SDA of the EEPROM is connected to I2C SDA from FX2LP and firmware is running from the EEPROM J2 Open: Comes up in bootloader mode with default USB descriptors	Closed

4.2.2 GPIF Connectors

CY3689 EZ-USB FX2LP Discovery Kit is mounted with two 20-pin GPIF connectors J3 and J4. The FX2LP GPIOs and pins essential for interfacing the kit with add-on modules, accessory boards, FPGA boards or any other boards, are routed to the two interface connectors J3 and J4. **Table 2** explains the functionality of pins in the interface connectors.

Refer the **FX2LP datasheet** for more details on the pins/pin descriptions.

CY3689 EZ-USB FX2LP Kit Hardware

Table 2 FX2LP Interface Connector Pinout

Interface Connector	CY3689 Pin Names	FX2LP Signal/GPIO	Description
J3	1: PD5	1: PD5/FD13	Multiplexed pin whose function is selected by the IFCONFIG [1..0] and EPxFIFOCFG.0 (word-wide) bits.
	2: PD6	2: PD6/FD14	
	3 PD7	3: PD7/FD15	
	4: GND		
	5: CLK	5: CLKOUT	CLKOUT: 12-, 24- or 48-MHz clock, phase-locked to the 24-MHz input clock.
	6: GND		
	7: SLRD	8: RDY0/*SLRD	SLRD is the input-only read strobe with programmable polarity (FIFOPINPOLAR.3) for the slave FIFOs connected to FD[7..0] or FD[15..0]
	8: SLWR	9: RDY1/*SLWR	SLWR is the input-only write strobe with programmable polarity (FIFOPINPOLAR.2) for the slave FIFOs connected to FD[7..0] or FD[15..0].
	9: GND		
	10: VCC		Connect to the 3.3-V power source.
	11: GND		
	12: IFCLK	20: IFCLK	Interface clock, used for synchronously clocking data into or out of the slave FIFOs.
	13: SCL	22: SCL	Clock for the I2C interface. Connect to EEPROM I2C SCL. Pulled-up to VCC with a 2.2-kΩ resistor.
	14: SDA	23: SDA	Data for I2C compatible interface. Connect to EEPROM I2C SDA. Pulled-up to VCC with a 2.2-kΩ resistor.
	15: PB0	25: PB0/FD0	Multiplexed pin whose function is selected by the IFCONFIG [1..0] bits. PB0, PB1, PB2 and PB3 are bidirectional I/O port pins.
	16: PB1	26: PB1/FD1	
	17: PB2	27: PB2/FD2	
	18: PB3	28: PB3/FD3	
	19: GND		
	20: VCC		Connect to the 3.3-V power source.
J4	1: PD4	56: PD4/FD12	Multiplexed pin, whose function is selected by the IFCONFIG [1..0] and EPxFIFOCFG.0 (word-wide) bits.
	2: PD3	55: PD3/FD11	
	3: PD2	54: PD2/FD10	
	4: PD1	53: PD1/FD9	
	5: PD0	52: PD0/FD8	
	6: PA7	47: PA7/*FLAGD/SLCS#	Multiplexed pin, whose function is selected by the IFCONFIG [1:0] and PORTACFG.7 bits. PA7, PA5 and PA6 are bidirectional I/O port pins.
	7: PA6	46: PA6/PKTEND	
	8: PA5	45: PA5/FIFOADR1	
	9: PA4	44: PA4/FIFOADR0	Multiplexed pin whose function is selected by IFCONFIG [1..0]. PA4 is a bidirectional I/O port pin.

Interface Connector	CY3689 Pin Names	FX2LP Signal/GPIO	Description
	10: PA3	43: PA3/*WU2	Multiplexed pin whose function is selected by WAKEUP.7 and OEA.3. PA3 is a bidirectional I/O port pin.
	11: PA2	42: PA2/*SLOE	Multiplexed pin whose function is selected by IFCONFIG [1..0]. PA4 is a bidirectional I/O port pin.
	12: PA1	41: PA1/INT1#	Multiplexed pin whose function is selected by PORTACFG.1. PA1 is a bidirectional I/O port pin.
	13: PA0	40: PA0/INT0#	Multiplexed pin whose function is selected by two bits, PORTACFG.0 PA2 is a bidirectional I/O port pin.
	14: CTL2	38: CTL2/*FLAGC	CTL2 is a GPIF control output.
	15: CTL1	37: CTL1/*FLAGB	CTL1 is a GPIF control output.
	16: CTL0	36: CTL0/*FLAGA	CTL0 is a GPIF control output.
	17: PB7	32: PB7/FD7	Multiplexed pin whose function is selected by the IFCONFIG [1..0] bits. PB4, PB5, PB6, and PB7 are bidirectional I/O port pins.
	18: PB6	31: PB6/FD6	
	19: PB5	30: PB5/FD5	
	20: PB4	29: PB4/FD4	

4.3 Accessory Board/Add-on Modules Specification

Accessory boards or add-on modules are used to extend the functionality of the CY3689 FX2LP Discovery Kit. The modules are interfaced to the Discovery Kit through the onboard GPIF connectors. The GPIF connectors can source only 3.3 V at 250 mA (maximum) for the accessory boards/add-on modules.

5 Programming the FX2LP Device in CY3689

This chapter explains the methods to update firmware through the USB interface from a host PC/Desktop. The USB Control Center is used to download and test FX2LP code. The USB Control Center is part of a larger suite of tools, SDK, which is available for free download. The suite includes extensive tools and documentation for writing Windows programs in the C++ or .NET languages that communicate with the FX2LP Discovery Kit via the Cypress (Infineon) driver

As mentioned in the **Getting Started** section, download and install the latest version of the SDK software. Select the **Typical** installation type. Use the downloaded firmware code examples referred to in section **3.2**. The CY3689 Dicovery Kit supports firmware download to external EEPROM (using .iic file) or internal RAM (using .hex file). Users can choose either EERPOM or RAM for firmware download target; the procedure to download firmware is explained below.

5.1 Download Firmware Image to External I2C EEPROM

1. Browse to the FX3 SDK installation directory. Open the USB Control Center application (CyControl.exe) from the location *<Installed directory>\<version>\application\c_sharp\controlcenter\bin\Release*. Choose the Cypress FX2LP device as in **Figure 30**.

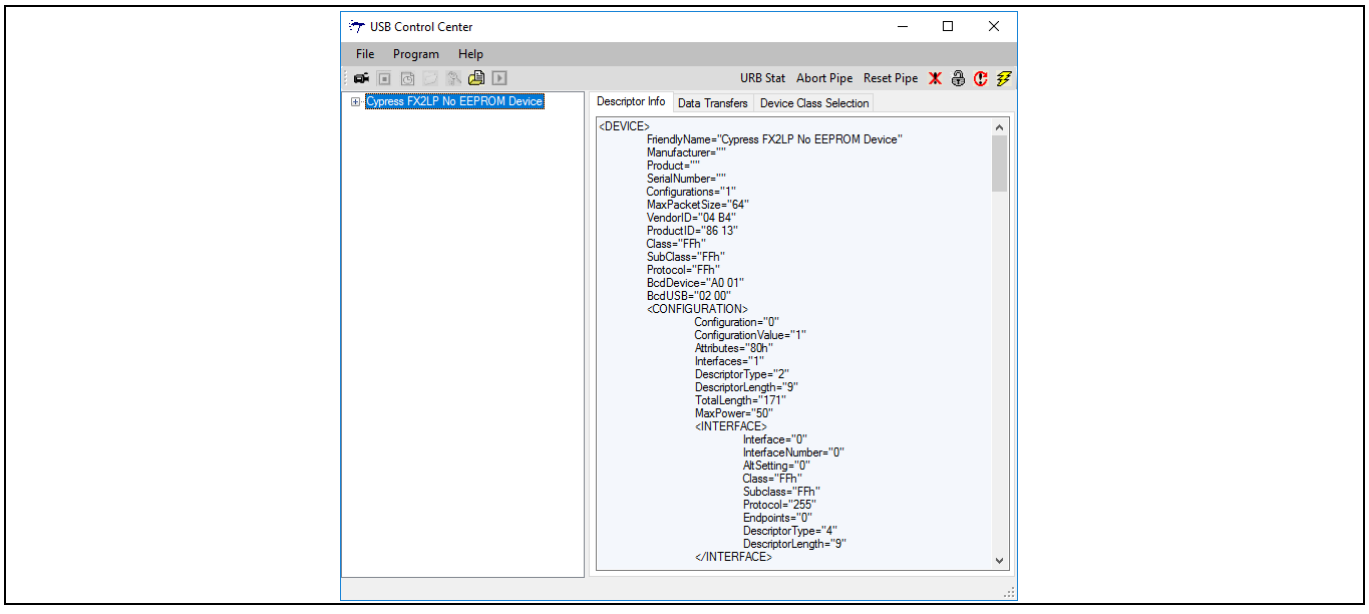


Figure 30 USB Control Center Tool

2. Connect the J2 jumper to program the onboard EERPOM (see **Jumper Settings**)
3. In the USB Control Center application, select **Program** > **FX2** > **64KB EEPROM**.

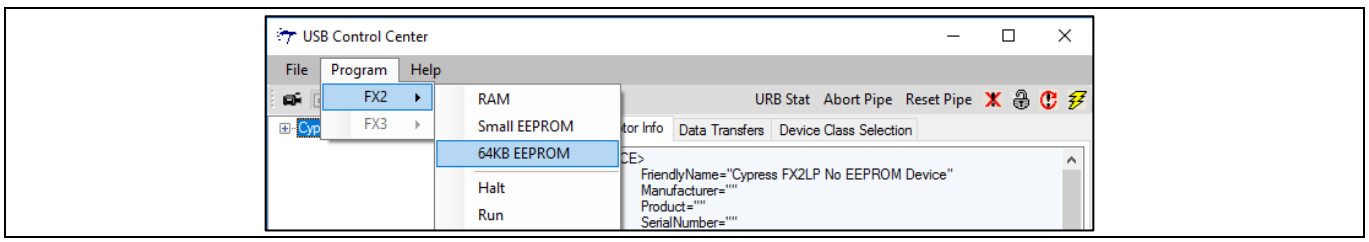


Figure 31 Selecting EEPROM

Programming the FX2LP Device in CY3689

4. Choose the desired firmware code example (iic file) from the location where the code examples were unzipped and saved (section **3.2**). To download this code example to the external EEPROM, click **Open** (**Figure 32**).

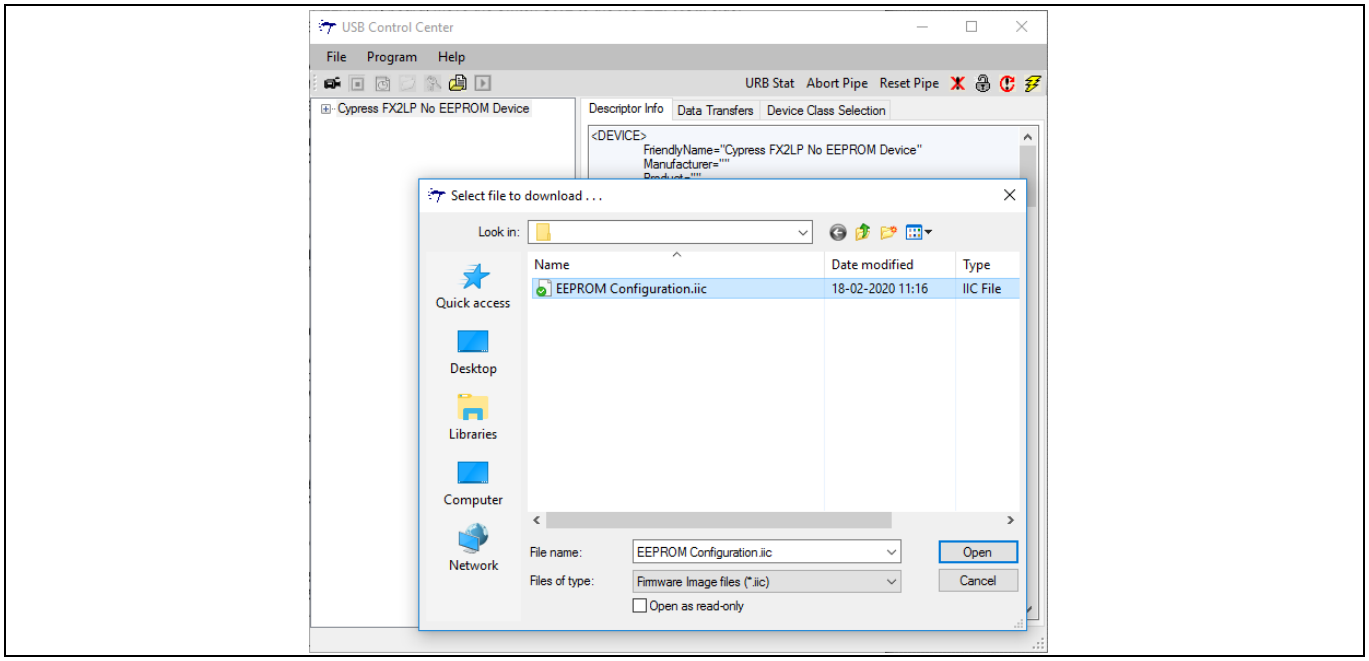


Figure 32 Selecting Firmware Image

5. Successful programming is indicated at the bottom left corner of the tool as "**Programming Succeeded**" (**Figure 33**).

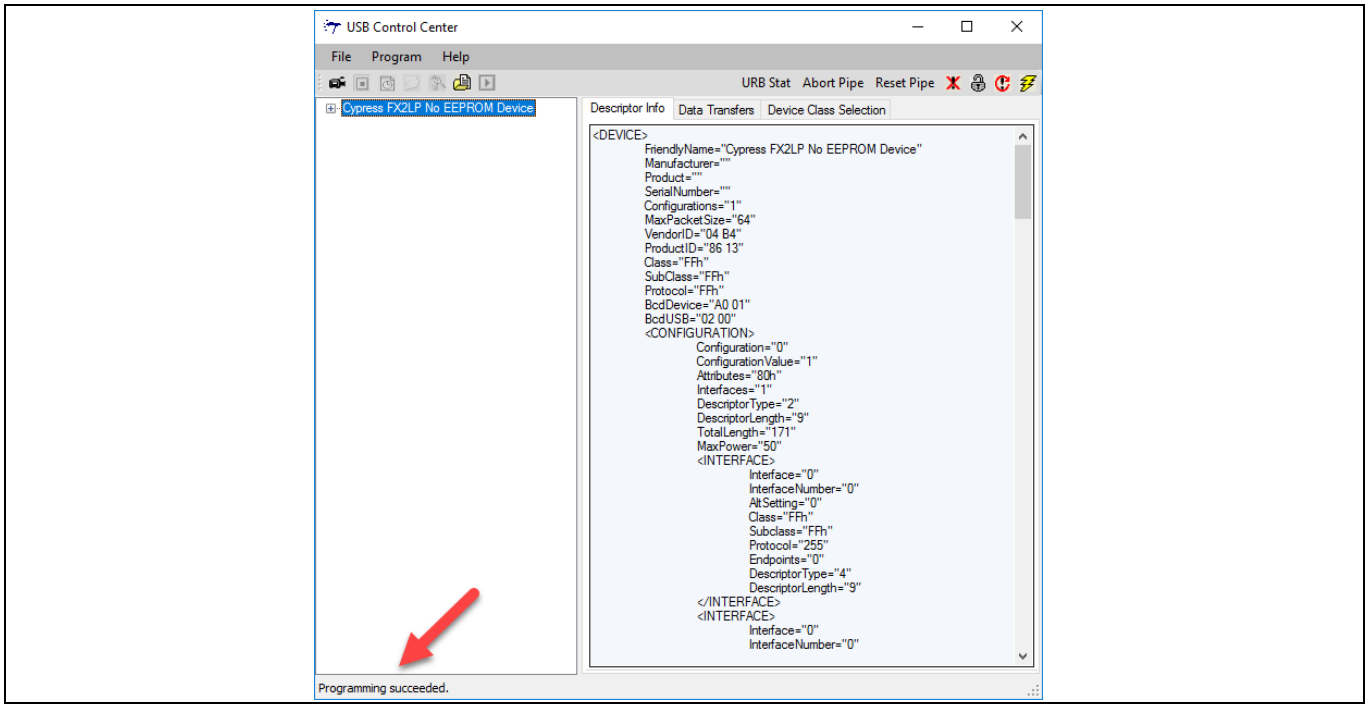


Figure 33 Programming Succeeded

6. Press reset button to re-enumerate Discovery kit with latest firmware.

5.2 Download Firmware Image to Internal RAM

1. Remove jumper J2 to put the kit in bootloader mode (see **Jumper Settings**)
2. Open the USB Control Center application (CyControl.exe) from the location *<Installed directory>\<version>\application\c_sharp\controlcenter\bin\Release*.
3. In the USB Control Center application, follow the path **Program** -> **FX2** -> **RAM** as in **Figure 34** to program the firmware code examples (.hex file).

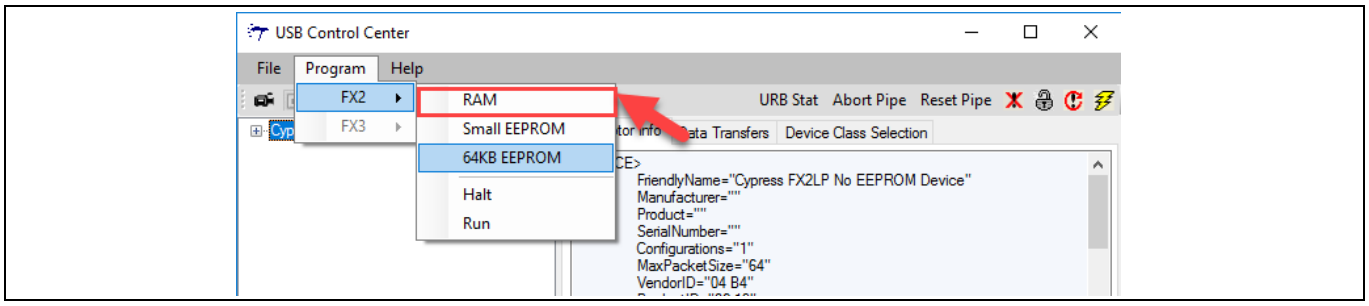


Figure 34 Selecting RAM as Programming Target

4. Choose the desired code example (.hex file) from the location where the firmware code examples were unzipped and saved (section **3.2**). To download this code example to the internal RAM, click **Open**.
5. Successful programming is indicated in the bottom left corner of the tool as "**Programming Succeeded**".

6 Firmware Examples

This chapter describe the steps to run the firmware examples provided with the example projects. The tools and utilities required for development and testing are available with the FX3 and FX2LP SDK installation. Keil IDE is used to open the source code and USB Control Center utility downloads the firmware into FX2LP device.

6.1 Bulkloop Example

6.1.1 Description

This project illustrates the configuration of FX2LP to accept bulk data from the host and loop it back to the host. Click **bulkloop.Uv2** at *<example project>\Firmware\Bulkloop* and observe the source code. Two endpoints are configured in the `TD_init()` function of bulkloop.c to handle bulk transfer: one OUT endpoint and one IN endpoint. The two endpoints defined in the descriptor file must be configured in this function in the following statements:

```
EP2CFG = 0xA2;
SYNCDELAY;
EP6CFG = 0xE2;
SYNCDELAY;
```

The key characteristics of each endpoint are as follows:

- Endpoint 2 - OUT, bulk, double buffered
- Endpoint 6 - IN, bulk, double buffered

After configuration, the OUT endpoints are armed to accept data from the host. An OUT endpoint is said to be armed if it is ready to accept data from the host. Each endpoint is configured as double buffered.

```
SYNCDELAY;
EP2BCL = 0x80;// arm EP2OUT by writing byte count w/skip.
SYNCDELAY;
EP2BCL = 0x80;
SYNCDELAY;
```

The previous lines arm the two OUT endpoints by skipping two packets of data, making the buffers available to receive OUT data.

```
AUTOPTRSETUP |= 0x01;
```

This enables the auto-pointer used for data transfer in the `TD_Poll` function. The data loopback is implemented in the `TD_Poll` function that is called repeatedly when the device is idle. Endpoint 2 is armed to accept data from the host. This data is transferred to endpoint 6. First, endpoint 2 is checked to see if it has data by reading the endpoint 2 empty bit in the endpoint status register (EP2468STAT). If endpoint 2 has data (sent from the host), the capability of endpoint 6 to receive the data is checked by reading the endpoint 6 full bit in the endpoint status register. If endpoint 6 is not full, then the data is transferred. This decision is executed by the following statements:

```
if (!(EP2468STAT & bmEP2EMPTY))
{// check EP2 EMPTY (busy) bit in EP2468STAT (SFR), core set's this bit when
// FIFO is empty
if (!(EP2468STAT & bmEP6FULL))
```

Firmware Examples

```
{// check EP6 FULL (busy) bit in EP2468STAT (SFR), core set's this bit
// when FIFO is full
```

The data pointers are initialized to the corresponding buffers. The first auto-pointer is initialized to the first byte of the endpoint 2 FIFO buffer. The second auto-pointer is initialized to the first byte of the endpoint 6 FIFO buffer. The number of bytes to be transferred is read from the byte count registers of endpoint 2. The registers EP2BCL and EP2BCH contain the number of bytes written into the FIFO buffer by the host. These two registers give the byte count of the data transferred to the FIFO in an OUT transaction as long as the data is not committed to the peripheral side. This data pointer initialization and loading of the count is done in the following statements:

```
APTR1H = MSB( &EP2FIFOBUF ); // Initializing the first data pointer
APTR1L = LSB( &EP2FIFOBUF );
AUTOPTRH2 = MSB( &EP6FIFOBUF); // Initializing the second data pointer
AUTOPTRL2 = LSB( &EP6FIFOBUF );
count = (EP2BCH << 8) + EP2BCL; // The count value is loaded from the byte
// count registers
```

The data transfer is carried out by the execution of the following loop:

```
for( i = 0x0000; i < count; i++ )
{
// setup to transfer EP2OUT buffer to EP6IN buffer using AUTOPOINTER(s)
EXTAUTODAT2 = EXTAUTODAT1;
}
```

Because auto-pointers are enabled, the pointers increment automatically and the statement

```
EXTAUTODAT2 = EXTAUTODAT1;
```

transfers data from endpoint 2 to endpoint 6. Each time the statement is executed, the auto-pointer is incremented. It is executed repeatedly to transfer each byte from endpoint 2 to 6.

After the data is transferred, endpoint 2 must be rearmed to accept a new packet from the host. Endpoint 6 is committed to make the FIFO buffers available to the host for reading data from endpoint 6.

This is accomplished by the following statements:

```
EP6BCH = EP2BCH;
SYNCDELAY;
EP6BCL = EP2BCL; // commit EP6IN by specifying the number of bytes the host
can read
//from EP6
SYNCDELAY;
EP2BCL = 0x80; // re (arm) EP2OUT
```

6.1.2 Building Bulkloop Firmware Example Code for EZ-USB RAM and EEPROM

- Click the **Build Target** button at the top right corner of the IDE. Build window of the Keil IDE displays the successful compilation of the entire project.
- Firmware output for EZ-USB RAM: The output of the Build Target is *bulkloop.hex*, which is the relevant file for downloading to EZ-USB RAM.

- Firmware output for external EEPROM: To generate an EEPROM-compatible firmware image, the Keil IDE invokes the *hex2bix.exe* utility to convert the output file *bulkloop.hex* into *bulkloop.iic*. Right-click **Target1** in the project window and select **Options for Target 'Target1'**. This results in a pop-up of the Keil settings for this project. Select the **Output** tab and observe at the bottom of IDE that the hex2bix utility is invoked in the Run User program#1 section and that the hex2bix utility is invoked in the following manner

```
hex2bix -i -c 0x00 -f 0xC2 -o bulkloop.iic bulkloop.hex
```

6.1.3 Download Bulkloop Firmware Image to Internal RAM or EEPROM

Refer to sections **Download Firmware Image to External I2C EEPROM** and **Download Firmware Image to Internal RAM**.

6.1.4 Binding Cypress USB Driver for the Downloaded Firmware Image

The bulkloop firmware uses vendor class (0xFF) with VID/PID 0x04B4/1004. This example should bind with generic USB driver ***cyUSB3.sys*** and driver information file ***CyUSB3.inf***, which contains the relevant VID/PID for this example. Follow the procedure outlined to manually bind the driver using the Windows Hardware Wizard. If you have performed the binding process for any one of the firmware examples, you can skip it for this example.

6.1.5 Testing Bulkloop Firmware Functionality

The bulkloop firmware functionality can be tested using the following applications:

- USB Control Center
- Bulkloop (C# .NET)

6.1.5.1 Test Using USB Control Center PC Application

1. Locate the USB Control Center folder and navigate to its bin directory (*<FX3 SDK installation folder>\<version>\application\c_sharp\controlcenter\bin\Release*). Double-click the *CyControl.exe* file in either the Debug or Release folder.
2. Expand the **Cypress FX2LP Sample Device** entry to reveal the device's bulk endpoints EP2-OUT and EP6-IN.

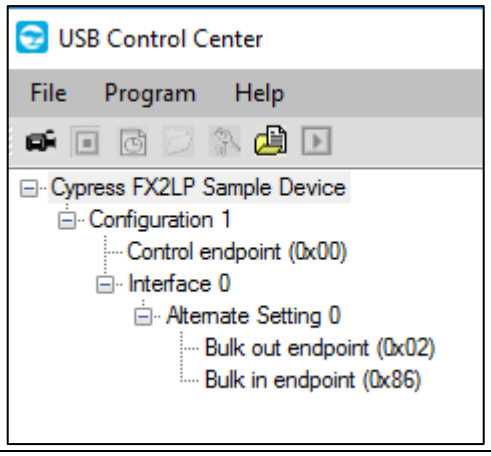


Figure 35 Bulkloop Example Device Internals

3. Select the **Data Transfers** tab. Click the **Bulk out endpoint (0x02)** entry in the left-hand panel; notice that the Transfer Data button appears as **Transfer Data-OUT**. Click this button and observe a total of 512 bytes with zero default values transfer from the PC to the FX2LP board. Click the **Transfer Data-OUT** button again. The PC dispatches a second packet to FX2LP.

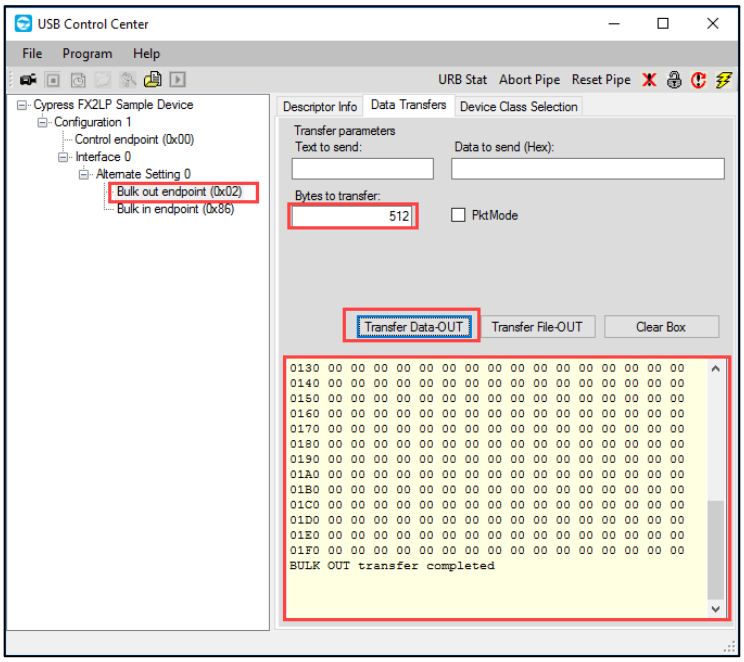


Figure 36 Successful Bulk OUT Transfer

4. Highlight the **Bulk in endpoint (0x86)** entry. The Transfer Data button now appears as **Transfer Data-IN**. Click this button and observe a total of 512 bytes transfer from FX2LP to the host, which it displays as hexadecimal values.
5. Click the **Transfer Data-IN** button again. The second queued FX2LP packet transfers to the host and this sequence confirms the double-buffered operation of the two endpoints.

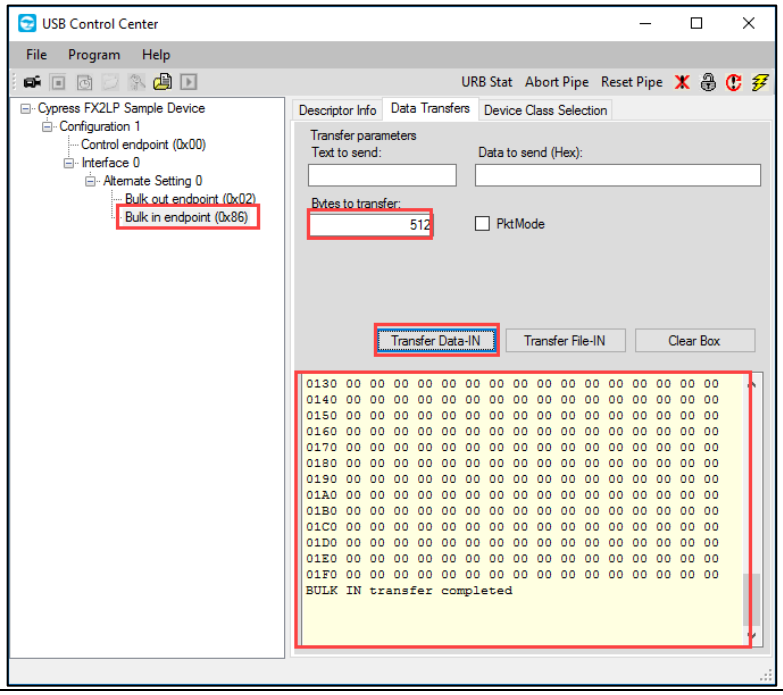


Figure 37 Successful Bulk IN Transfer

6. Select the **Bulk out endpoint (0x02)** again. Position the mouse cursor inside the **Text to send** box and type “1234”. The hex values display in the **Data to send** box, and the **Bytes to Transfer** box increments for every digit typed. Click **Transfer Data-OUT**.
7. In the **Text to send** box, type “abcd”, and then click the **Transfer Data-OUT** button.
8. Select the **Bulk in endpoint (0x86)**, and leave the default **Bytes to Transfer** value of 512 bytes. Click the **Transfer Data-IN** button twice.

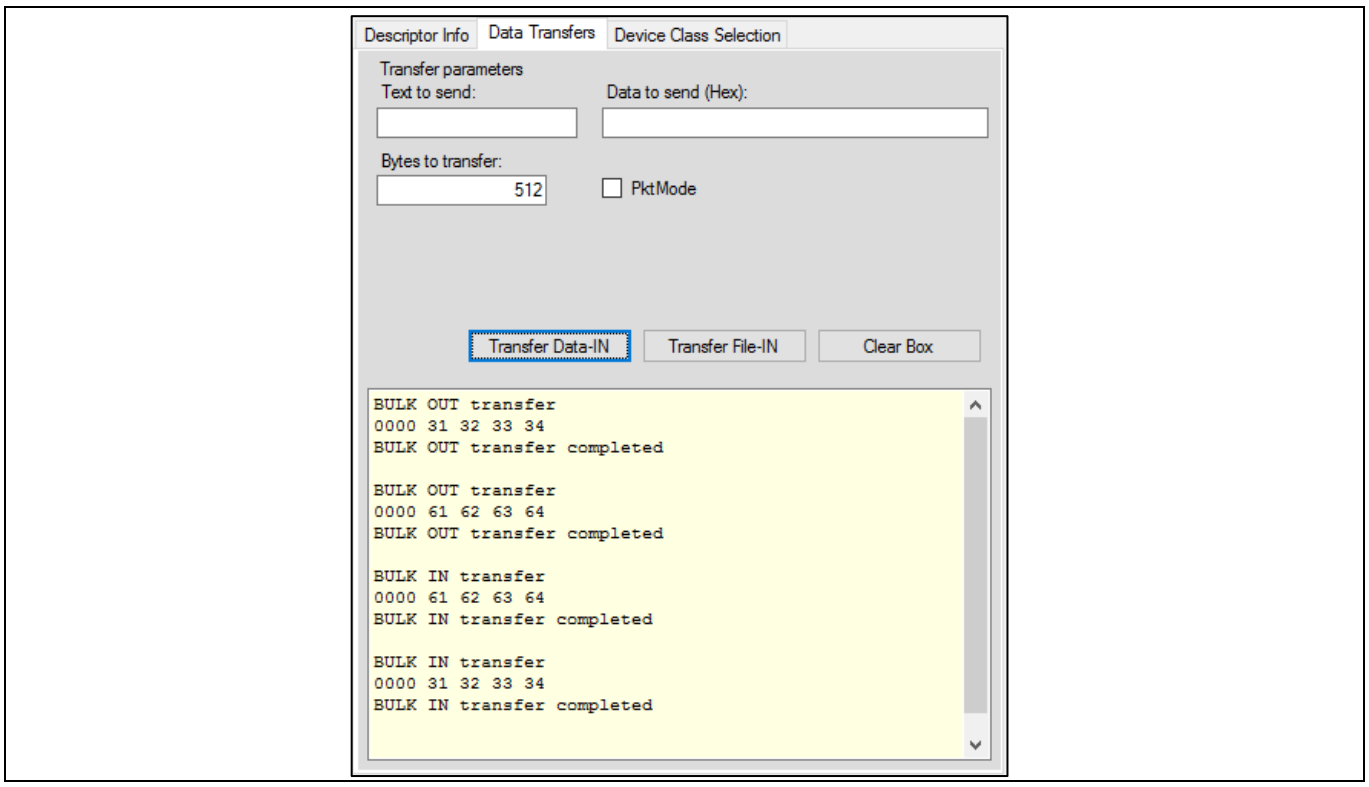


Figure 38 Two Packets OUT, Two Packets IN

6.1.5.2 Test using Bulkloop C#.NET Application

The bulkloop firmware can be tested using the Bulkloop C# .NET application. For the Windows OS, bulkloop can be accessed at *<FX3 SDK Installed directory>\<version>\application\c_sharp\bulkloop\bin\Release*. Select the bulkloop OUT and bulkloop IN endpoint pairs EP2 and EP6. Click **Start** and observe the number of successful bulk IN and bulk OUT transfers.

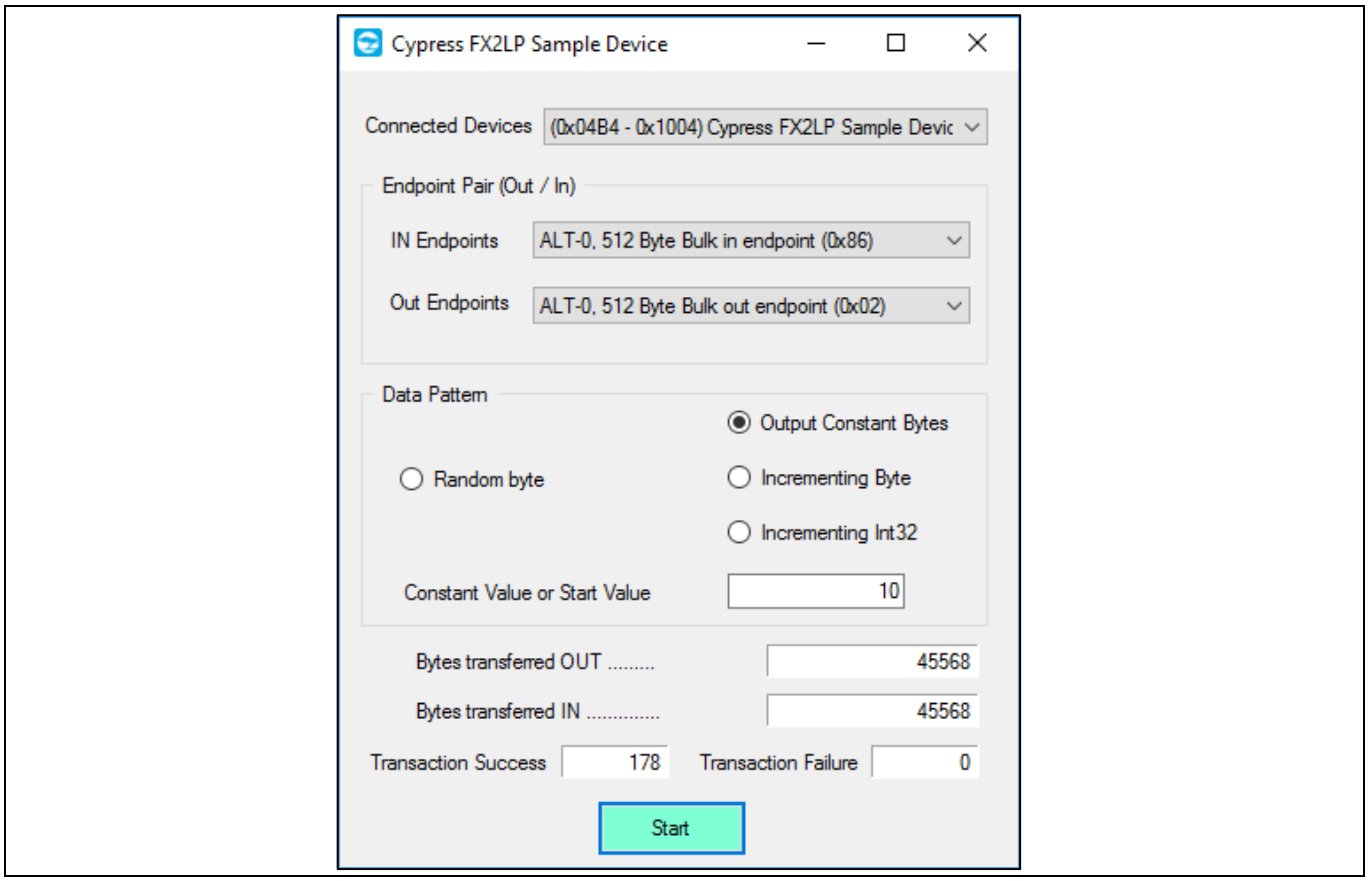


Figure 39 Data loopback using Bulkloop C# .NET Application

6.2 Bulksrc Firmware Examples

6.2.1 Description

This project illustrates the configuration of the EZ-USB device to accept bulk data from the host and loop it back to the host. Click the **bulksrc.Uv2** project located at *<Example project folder>\Firmware\Bulksrc* and observe the code. Five endpoints are configured in the `TD_init()` function of *bulksrc.c* to handle bulk transfer. Two OUT (EP2/EP4) endpoints and two IN (EP6/EP8) endpoints are double-buffered pairs. The fifth endpoint is EP1, which acts as both the bulk IN and bulk OUT endpoint with a 64-byte buffer. These are defined in the descriptor file (dscr.a51). The endpoints are configured in this `TD_init` function in the following statements:

```
EP1OUTCFG = 0xA0; EP1INCFG = 0xA0;
SYNCDELAY;
EP2CFG = 0xA2;
SYNCDELAY;
EP4CFG = 0xA0;
SYNCDELAY;
EP6CFG = 0xE2;
SYNCDELAY;
EP8CFG = 0xE0;
```

After configuration, the OUT endpoints are armed to accept data from the host. An OUT endpoint is said to be armed if it is ready to accept data from the host. Each endpoint is configured as double buffered.

Firmware Examples

```
/* since the defaults are double buffered we must write dummy byte counts
twice */
SYNCDELAY;
EP2BCL = 0x80;                // arm EP2OUT by writing byte count w/skip.
SYNCDELAY;
EP4BCL = 0x80;
SYNCDELAY;
EP2BCL = 0x80;          /* arm EP4OUT by writing byte count w/skip. */
SYNCDELAY;
EP4BCL = 0x80;
```

The previous lines arm the two OUT endpoints by loading byte counts for two data packets with the skip bit set, making the buffers available to receive OUT data. The IN endpoint, EP6, is armed with an incrementing pattern of data starting with 0x2, regardless of the data sent on the EP2 bulk OUT endpoint, as shown in the following code.

```
for (i=0;i<512;i++)
EP6FIFOBUF[i] = i+2;
SYNCDELAY;
EP6BCH = 0x02;
SYNCDELAY;
EP6BCL = 0x00;
```

In the TD_poll() function, if there is packet content in EP2, then it is rearmed, discarding the current data.

```
// if there is some data in EP2 OUT, rearm it
if(!(EP2468STAT & bmEP2EMPTY))
{

SYNCDELAY;
EP2BCL = 0x80;
}
```

Endpoint EP6 is rearmed with an incrementing pattern of data starting with 0x2.

```
// if EP6 IN is available, rearm it
If(!(EP2468STAT & bmEP6FULL))
{
SYNCDELAY; EP6BCH = 0x02; SYNCDELAY; EP6BCL = 0x00;
}
```

The contents received from the EP4 OUT endpoint are copied to a temporary buffer, myBuffer[], and rearmed.

```
// if there is new data in EP4FIFOBUF, then copy it to a temporary buffer
if(!(EP2468STAT & bmEP4EMPTY))
{
APTR1H = MSB( &EP4FIFOBUF ); APTR1L = LSB( &EP4FIFOBUF );
```

Firmware Examples

```
AUTOPTRH2 = MSB( &myBuffer ); AUTOPTRL2 = LSB( &myBuffer );
myBufferCount = (EP4BCH << 8) + EP4BCL;
for( i = 0x0000; i < myBufferCount; i++ )
{
EXTAUTODAT2 = EXTAUTODAT1;
}
SYNCDELAY;
EP4BCL = 0x80; // re(arm) EP4OUT
}
```

If the EP8 bulk IN endpoint is empty, then the contents of the temporary buffer are transferred to an auto-pointer and copied to the EP8 IN buffer, as shown in the following code.

```
/* if there is room in EP8IN, then copy the contents of the temporary buffer
to it */
if(!(EP2468STAT & bmEP8FULL) && myBufferCount)
{
APTR1H = MSB( &myBuffer ); APTR1L = LSB( &myBuffer );
AUTOPTRH2 = MSB( &EP8FIFOBUF ); AUTOPTRL2 = LSB( &EP8FIFOBUF );
for( i = 0x0000; i < myBufferCount; i++ )
{
/* setup to transfer EP4OUT buffer to EP8IN buffer using AUTO- POINTER(s) in
SFR space */
EXTAUTODAT2 = EXTAUTODAT1;
}

SYNCDELAY;
EP8BCH = MSB(myBufferCount); SYNCDELAY;
EP8BCL = LSB(myBufferCount);// arm EP8IN
}
```

6.2.2 Building Bulksrc Firmware Example Code for EZ-USB RAM and EEPROM

Click the **Project>Build Target** option at the top left corner of the IDE. The **Total Code Bytes** of the Bulksrc firmware example is less than the 4-KB code limit of the Keil µVision2 IDE provided with the kit. The output of the Build Target is the *bulkext.hex* and *bulkext.iic* files.

6.2.3 Download Firmware Image to EZ-USB Internal RAM and EEPROM

Refer to sections **Download Firmware Image to External I2C EEPROM** and **Download Firmware Image to Internal RAM**.

6.2.4 Binding Cypress USB Driver for the Downloaded Firmware Image

The Bulksrc firmware uses vendor-class (0xFF) with VID/PID 0x04B4/1004. This example should bind with the generic USB driver, ***CyUSB3.sys***, and driver information file, ***CyUSB3.inf*** which contains the relevant VID/PID of this example. Follow the procedure outlined in Binding Cypress USB Driver for the Downloaded Firmware

Firmware Examples

Image to manually bind the driver using the Windows Hardware Wizard. If you have performed the binding process for any one of the previous firmware examples, you can skip it for this example.

6.2.5 Testing Bulksrc Firmware Functionality

The Bulksrc firmware functionality can be tested using the USB Control Center utility as follows.

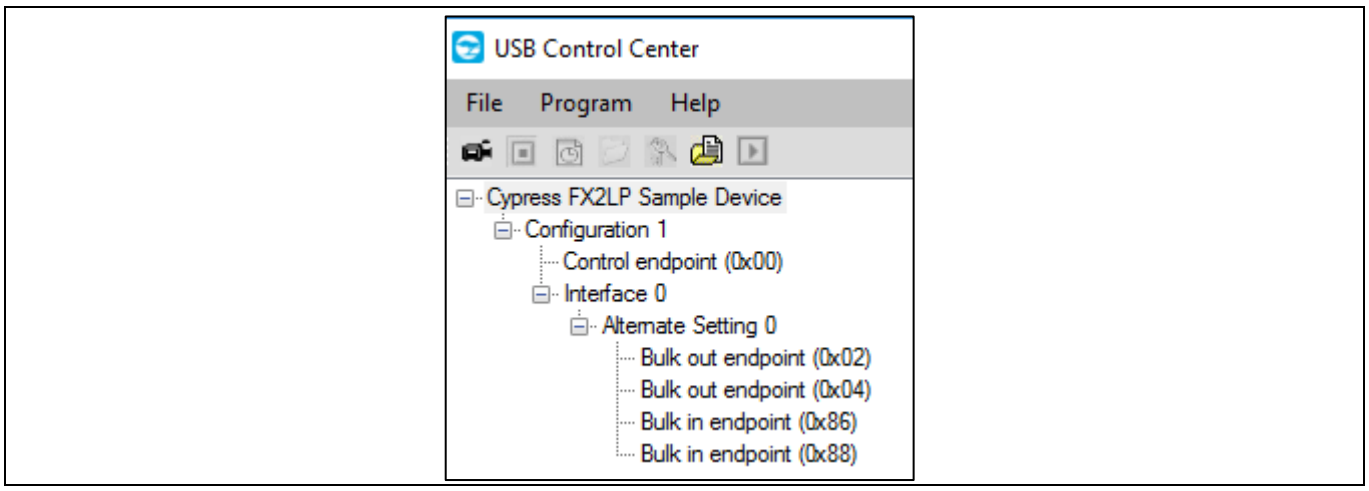


Figure 40 Bulksrc Firmware Internals

Open the USB Control Center application (CyControl.exe) from the location *<FX3 SDK Installed directory>\<version>\ application\c_sharp\controlcenter\bin\Release*. Observe that the EZ-USB FX2LP is listed as "**Cypress FX2LP Sample Device**".

1. Select **Bulk out endpoint (0x02)** in the left pane of the USB Control Center and keep clicking the **Transfer Data-OUT** button in the Data Transfers tab on the right. You can perform any number of OUT transfers on bulk out endpoint (0x02).
2. Similarly, select **Bulk In endpoint (0x86)** and keep clicking the **Transfer Data-IN** button in the Data Transfers tab. You can see the USB Control Center receiving multiple packets (incremental data from 02) from this bulk IN endpoint.

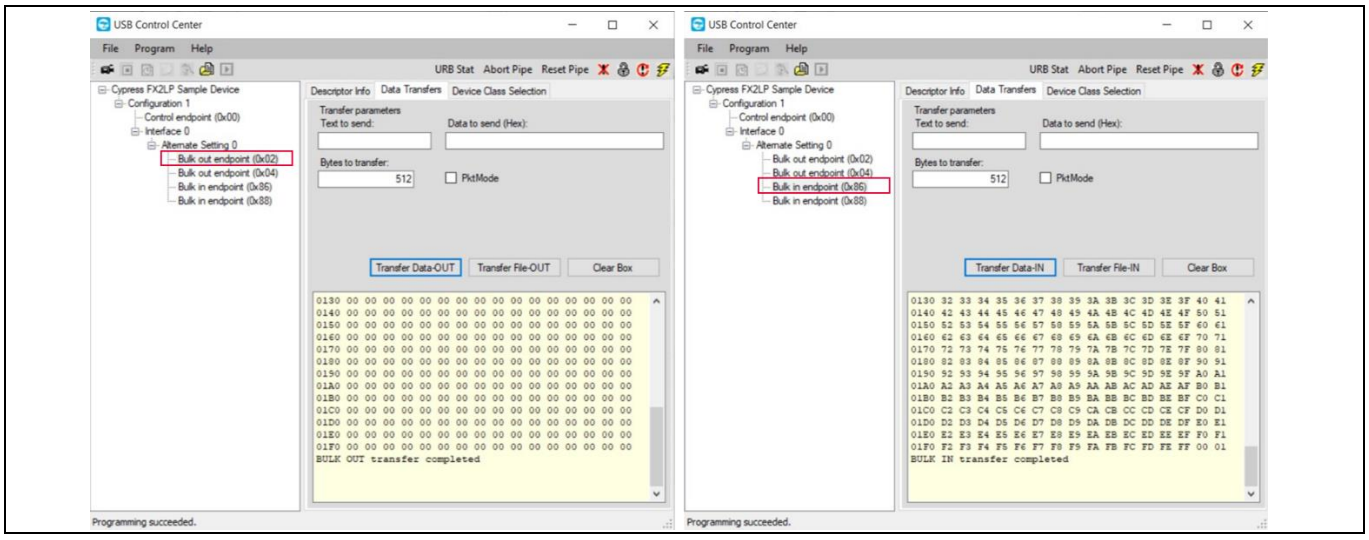


Figure 41 Bulksrc Firmware- EP2 to EP

Firmware Examples

3. Select **Bulk out endpoint (0x04)** and keep clicking the **Transfer Data-OUT** button in the Data Transfers tab. You can see the USB Control Center sending multiple packets to this bulk OUT endpoint.
4. Select **Bulk In endpoint (0x88)** and keep clicking the **Transfer Data-IN** button in the Data Transfers tab. The data transferred on EP4 is exactly looped back to EP8. Internally, the loopback is performed through a temporary buffer (myBuffer [512]).

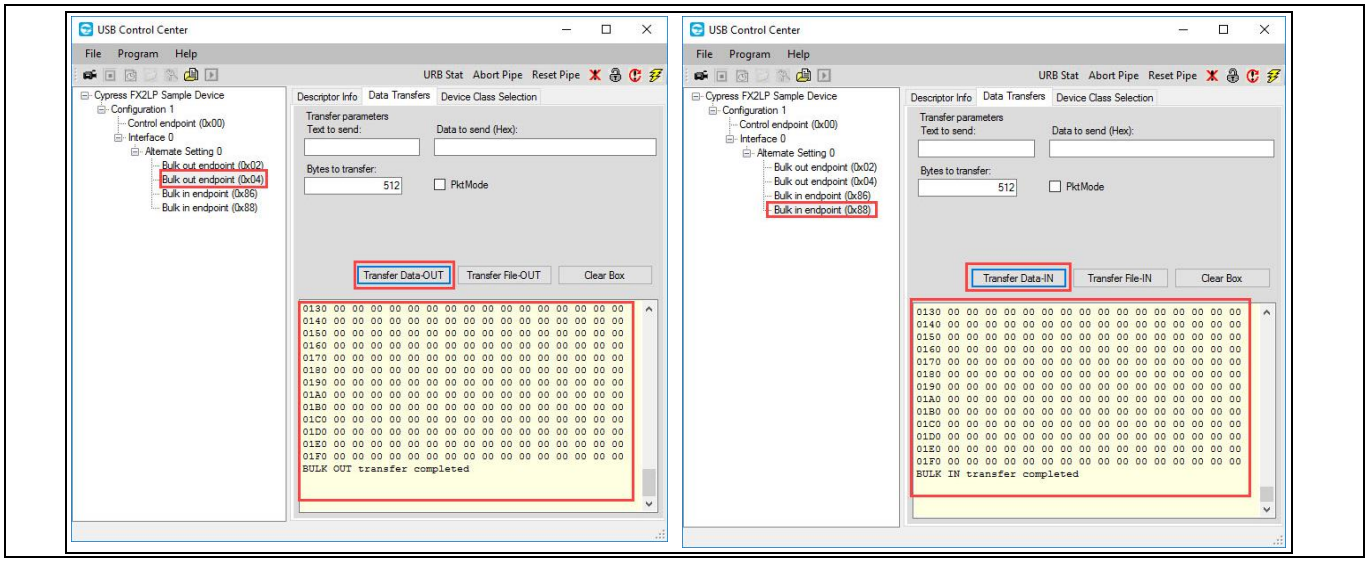


Figure 42 Bulksrc Firmware- EP4 to EP8

6.3 EP_Interrupts Example

6.3.1 Description

The EP_interrupts example works in a similar manner as bulkloop on EZ-USB FX2LP. The major differences include the addition of 64-byte EP1 as a bulk OUT/IN endpoint to the existing four endpoints: EP2, EP4, EP6, and EP8. The endpoints are rearmed using their respective ISRs. Following are the interrupts for these endpoints, which are used to schedule the data transfers.

- EP1: 64-byte bulk OUT/IN – ISR_Ep1in() and ISR_Ep1out()
- EP2: 512-byte bulk OUT – ISR_Ep2inout()
- EP4: 512-byte bulk IN – ISR_Ep4inout()
- EP6: 512-byte bulk OUT – ISR_Ep6inout()
- EP8: 512-byte bulk OUT – ISR_Ep8inout()

6.3.2 Building EP_Interrupts Firmware Example Code for EZ-USB RAM and EEPROM

Click the **Project** > **Build Target** option at the top left corner of the IDE. The “Total Code Bytes” of the EP_Interrupts firmware example is less than the 4-KB code limit of the Keil µVision2 IDE provided with the kit. The output of the Build Target is the *EP_Interrupts.hex* and *EP_Interrupts.iic* files

6.3.3 Method to Program EP_Interrupts Firmware Image to EZ-USB Internal RAM and EEPROM

Refer to sections **Download Firmware Image to External I2C EEPROM** and **Download Firmware Image to Internal RAM**. After downloading, the firmware re-enumerates with the PC using its internal VID/PID 0x04B4/0x1004.

6.3.4 Binding Cypress USB Driver for the Downloaded Firmware Image

The EP_Interrupts firmware uses vendor class (0xFF) with VID/PID 0x04B4/1004. This example should bind with the generic USB driver ***cyUSB3.sys*** and driver information file ***cyUSB3.inf***, which contains the relevant VID/PID of this example. Follow the procedure outlined to manually bind the driver using the Windows Hardware Wizard. If you have performed the binding process for any one of the previous firmware examples, you can skip it for this example.

6.3.5 Testing EP_Interrupts Firmware Functionality

The example firmware should be tested in a manner similar to the bulkloop example. The bulk data transfers on EP1 are tested with a length of 64 bytes and 512 bytes for EP2, EP4, EP6, and EP8.

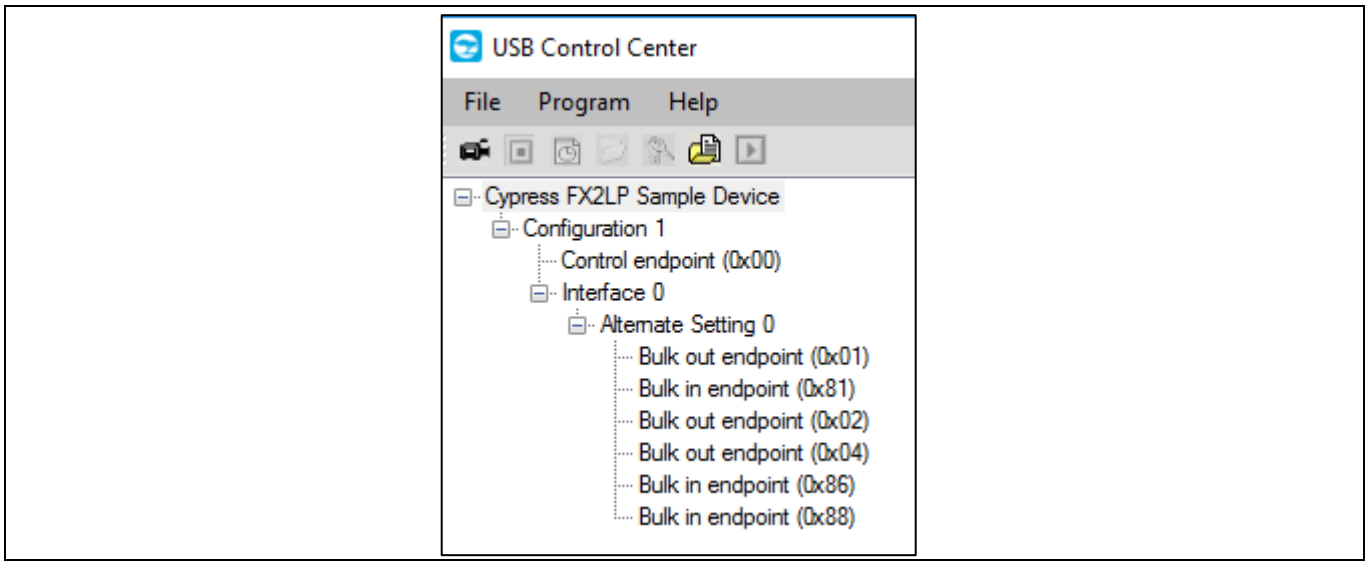


Figure 43 EP_Interrupts Firmware Device Internals

1. Select **Bulk out endpoint (0x01)** in the left pane of the USB Control Center, enter 64 in **Byte to transfer** text box and click the **Transfer Data-OUT** button in the Data Transfers tab. Perform only one OUT transfers on bulk out endpoint (0x01); multiple tries will cause a transfer failure.
2. Select **Bulk in endpoint (0x81)** in the left pane of the USB Control Center and click the **Transfer Data-IN** button in the Data Transfers tab. EP1 is IN/OUT endpoint, and same data will be received.
3. Select **Bulk out endpoint (0x02)** in the left pane of the USB Control Center and click the **Transfer Data-Out** button in the Data Transfers tab. Observe a total of 512 bytes with zero default values transfer from the PC to the FX2LP board. Click the **Transfer Data-OUT** button again. The PC dispatches a second packet to FX2LP.
4. Highlight the **Bulk in endpoint (0x86)** entry. The Transfer Data button now appears as **Transfer Data-IN**. Click this button and observe a total of 512 bytes transfer from FX2LP to the host, which it displays as hexadecimal values. Click the **Transfer Data-IN** button again. The second queued FX2LP packet transfers to the host and this sequence confirms the double-buffered operation of the two endpoints.
5. Repeat steps 3 and 4 for Bulk out endpoint (0x04) and Bulk in endpoint (0x88).

6.4 IBN Firmware Example

6.4.1 Description

This example illustrates the configuration of EZ-USB to accept bulk data from the host and loop it back to the host using an IN-BULK-NAK (IBN) interrupt. Click the **ibn.Uv2** project file at *<example project>\Firmware\ibn*. In the `TD_init()` function of the *ibn.c* file four endpoints are configured to handle bulk transfer: two OUT endpoints and two IN endpoints. The four endpoints defined in the descriptor file must be configured in this function with the following statements:

```
EP2CFG = 0xA2;
SYNCDELAY;
EP4CFG = 0xA0;
SYNCDELAY;
EP6CFG = 0xE2;
SYNCDELAY;
```

Firmware Examples

```
EP8CFG = 0xE0
```

The key characteristics of each endpoint are as follows:

- Endpoint 2: OUT, bulk, double-buffered
- Endpoint 4: OUT, bulk, double-buffered
- Endpoint 6: IN, bulk, double-buffered
- Endpoint 8: IN, bulk, double-buffered

Writing to these registers typically takes more than the two clock cycles needed for a MOVX instruction. Therefore, the SYNCDELAY macro is added. The **EZ-USB Technical Reference Manual** provides the list of registers that need this delay function when writing to them. After they are configured, the OUT endpoints need to be armed to accept packets from the host. Because the endpoints are double buffered, you must arm the endpoint twice. Arming essentially frees up the buffers, making them available to receive packets from the host.

By writing a '1' to bit7 of the byte count register, the endpoint is armed.

```
EP2BCL = 0x80; // arm EP2OUT by writing byte count w/skip.
SYNCDELAY;
EP2BCL = 0x80;
SYNCDELAY;
EP4BCL = 0x80; // arm EP4OUT by writing byte count w/skip.
SYNCDELAY;
EP4BCL = 0x80;
```

The previous lines arm the two OUT endpoints by skipping two packets of data, making the buffers available to receive OUT data.

```
NAKIRQ = bmBIT0; // clear the global IBN IRQ
NAKIE |= bmBIT0; // enable the global IBN IRQ
IbnFlag = 0x00; // clear our IBN flag
IBNIRQ = 0xFF; // clear any pending IBN IRQ
IBNIE |= bmEP6IBN | bmEP8IBN; // enable the IBN interrupt for EP6 and EP8
```

The firmware clears the IBN flags of all endpoints and any pending IBN interrupts and enables the IBN interrupt for EP6 and EP8.

```
AUTOPTRSETUP |= 0x01;
```

This enables the auto-pointer used for data transfer in the `TD_Poll()` function. The loopback is implemented in the `TD_Poll` function, which is called repeatedly when the device is idle. Endpoints 2 and 4 are armed to accept data from the host. This data is transferred to endpoint 6 and endpoint 8 respectively. To implement this, endpoint 2 is first checked to see if it has data by reading the endpoint 2 empty bit in the endpoint status register (EP2468STAT). If endpoint 2 has data (sent from the host), then check if the host has requested data on EP6 by reading the EP6 In-Bulk-Flag bit in the IbnFlag variable. If the host has requested data on EP6, then the data is transferred.

This decision is executed by the following statement:

```
if (!(EP2468STAT & bmEP2EMPTY) && (IbnFlag & bmEP6IBN) )
// if there is new data in EP2FIFOBUF and the IBN flag for EP6 has been set,
//then copy the data from EP2 to EP6
```

Firmware Examples

The data transfer is carried out by the execution of the following loop:

```
for( i = 0x0000; i < count; i++ )
{
// setup to transfer EP2OUT buffer to EP6IN buffer using AUTOPOINTER(s)
EXTAUTODAT2 = EXTAUTODAT1;
}
```

As auto pointers are enabled, the pointers increment automatically.

```
EXTAUTODAT2 = EXTAUTODAT1;
```

After this statement transfers the data, endpoint 2 must be rearmed to accept a new packet from the host. Endpoint 6 must be committed, making the FIFO buffers available to the host for reading data from endpoint 6. This is accomplished by the following statements:

This is accomplished by the following statements:

```
EP6BCH = EP2BCH;
SYNCDELAY;
EP6BCL = EP2BCL; // commit EP6IN by specifying the number of bytes the host
can read //from EP6
SYNCDELAY;
EP2BCL = 0x80; // re (arm) EP2OUT
```

The EP6 IBN flag bit in the IbnFlag variable is cleared. The EP6 IBN interrupt request is cleared by setting the corresponding bit in the IBNIRQ register. Finally, the EP6 IBN interrupt is enabled by setting the corresponding bit in the IBNIE register.

```
IbnFlag &= ~bmEP6IBN; // clear the IBN flag
IBNIRQ = bmEP6IBN; // clear the IBN IRQ
IBNIE |= bmEP6IBN; // enable the IBN IRQ
```

The same operation is carried out to implement a data loop with endpoints 4 and 8. When the host requests an IN packet from an EZ-USB bulk endpoint, the endpoint NAKs (returns the NAK PID) until the endpoint buffer is filled with data and armed for transfer, at which point the EZ-USB at which point FX2LP answers the IN request with data. Until the endpoint is armed, a flood of IN-NAKs can tie up bus bandwidth. Therefore, if the IN endpoints are not always kept full and armed, it may be useful to know when the host is “knocking at the door, requesting IN data.” The IBN interrupt provides this notification. It fires whenever a bulk endpoint NAKs an IN request. The IBNIE/IBNIRQ registers contain individual enable and request bits for each endpoint, and the NAKIE/NAKIRQ registers each contain a single-bit, IBN, that is the ORed combination of the individual bits in IBNIE/IBNIRQ, respectively. The EZ-USB FX2LP firmware framework provides hooks for all the interrupts that it implements. The example project uses the ISR_Ibn ISR to handle the IBN interrupt for EP6 and EP8.

```
void ISR_Ibn(void) interrupt 0
{
int i;
// disable IBN for all endpoints
IBNIE = 0x00;
EZUSB_IRQ_CLEAR(); // clear the global USB IRQ
// Find the EP with its IBN bit set
```

Firmware Examples

```
for (i=0;i<8;i++)
{
if (IBNIRQ & (1 << i))
{
IbnFlag |= (1 << i); // set the appropriate IBN flag bit
IBNIRQ |= (1 << i); // clear the IBN IRQ for this endpoint
}
}
NAKIRQ |= bmBIT0; // clear the global IBN IRQ
// re-enable IBN interrupt for any endpoints that don't already have
// an IBN pending in IbnFlag
IBNIE = (bmEP6IBN | bmEP8IBN) & ~IbnFlag;
}
```

6.4.2 Building Firmware Example Code for EZ-USB RAM and EEPROM

Click the **Project** > **Build Target** option at the top left corner of the IDE. The firmware example builds successfully since the “Total Code Bytes” of the IBN firmware example is less than the 4-KB code limit of the Keil µVision2 IDE provided with the kit. The output of the Build Target is the *ibn.hex* and *ibn.iic* files.

6.4.3 Method to Download Firmware Image to EZ-USB Internal RAM and External EEPROM

Refer to sections **Download Firmware Image to External I2C EEPROM** and **Download Firmware Image to Internal RAM**. After downloading, the firmware re-enumerates with the PC using its internal VID/PID 0x04B4/0x1004.

6.4.4 Binding Cypress USB Driver for the Downloaded Firmware Image

The IBN project uses vendor-class (0xFF) with VID/PID 0x04B4/1004. This example should bind with the generic USB driver, ***CyUSB3.sys***, and the driver information file, ***CyUSB3.inf***, which contains the relevant VID/PID of this example.

6.4.5 Testing the IBN Firmware Functionality

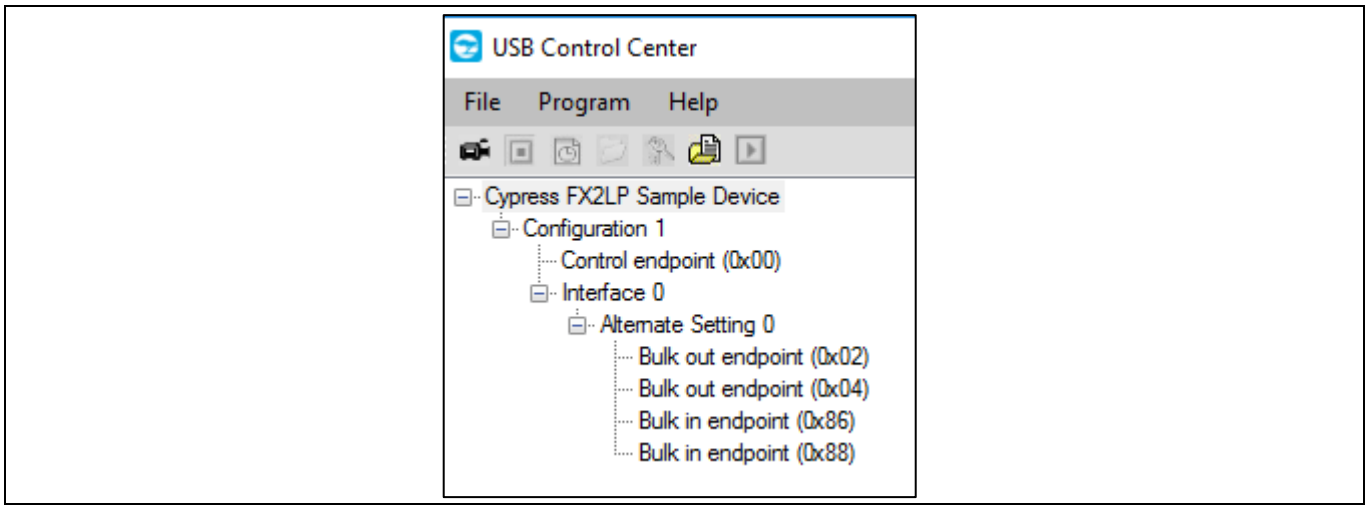


Figure 44 IBN Firmware Device Internals

Following are the detailed steps to test the functionality.

1. After the board enumerates, use the USB Control Center to send 512 bytes from EP2 to EP6. The data received should be the same as the data sent. The host can send 512 bytes of user-defined data to endpoint 2 using the USB Control Center. For example, select **Bulk out endpoint (0x02)** in the left pane of the USB Control Center and select the Data Transfers tab. In the Data Transfers tab, enter the **Bytes to Transfer** as '512' and then click the **Transfer Data-OUT** button.
2. This data can be read back from endpoint 6 using the USB Control Center. For example, select **Bulk in endpoint (0x86)** in the left pane of the USB Control Center and select the Data Transfers tab. In the Data Transfers tab, enter the number of **Bytes to Transfer** as '512' and then click the **Transfer Data-IN** button to read back the data. Similarly, loopback using endpoints 4 and 8 can also be tested. Since EP2 and EP4 are double buffered, they can contain only two packets of data.
3. On sending a packet to these endpoints when both the buffers are full, the endpoints NAK the transfer because there is no space available. If an IN transfer is requested on either EP6 or EP8, the corresponding IBN interrupt is asserted and data is transferred from EP2 to EP6 or from EP4 to EP8. This data appears in the USB Control Center window.
4. You can test the previous steps by trying to send data to EP2 and EP4 without reading the data out of EP6 or EP8. After the first two transfers, all the successive OUT transfers fail. This persists until an IN transfer is made on EP6 or EP8.

6.5 Pingnak Firmware Example

6.5.1 Description

This project illustrates the configuration of the EZ-USB device to accept bulk data from the host and loop it back to the host and the use of the PING-NAK interrupt. Click **pingnak.Uv2** located at *<example project>\Firmware\pingnak* and observe the code. Four endpoints are configured in the `TD_init()` function of *pingnak.c* to handle bulk transfer: two OUT endpoints and two IN endpoints. The four endpoints defined in the descriptor file must be configured in this function with the following statements:

```
EP2CFG = 0xA2;
SYNCDELAY;
EP4CFG = 0xA0;
```

Firmware Examples

```
SYNCDELAY;
EP6CFG = 0xE2;
SYNCDELAY;
EP8CFG = 0xE0
```

The key characteristics of each endpoint are as follows:

- Endpoint 2: OUT, bulk, double buffered
- Endpoint 4: OUT, bulk, double buffered
- Endpoint 6: IN, bulk, double buffered
- Endpoint 8: IN, bulk, double buffered

Writing to these registers typically takes more than the two clock cycles needed for a MOVX instruction. Therefore, the SYNCDELAY macro is added. The **EZ-USB Technical Reference Manual** the list of registers that need this delay function when writing to them. After they are configured, the OUT endpoints need to be armed to accept packets from the host. Because the endpoints are double buffered, you must arm the endpoint twice. Arming essentially frees up the buffers, making them available to receive packets from the host. By writing a '1' to bit 7 of the byte count register, the endpoint is armed.

```
EP2BCL = 0x80; // arm EP2OUT by writing byte count w/skip.
SYNCDELAY;
EP2BCL = 0x80;
SYNCDELAY;
EP4BCL = 0x80; // arm EP4OUT by writing byte count w/skip.
SYNCDELAY;
EP4BCL = 0x80;
```

After configuration, the OUT endpoints are 'armed' to accept data from the host. An OUT endpoint is said to be armed if it is ready to accept data from the host. Each endpoint is configured as double buffered. The OUT endpoints are armed by setting the skip bit in the byte count registers. This leaves them empty to receive a new packet from the host. It also clears any pending PING-NAK interrupts and enables the PING-NAK interrupt for EP2 and EP4. The loopback is implemented in the `TD_Poll()` function that is called repeatedly when the device is idle. Endpoints 2 and 4 are armed to accept data from the host. This data is transferred to endpoint 6 and endpoint 8 respectively. First, endpoint 2 is checked to see if it has data by reading the endpoint 2 empty bit in the endpoint status register (EP2468STAT). If endpoint 2 has data (sent from the host), the capability of endpoint 6 to receive the data is checked by reading the endpoint 6 full bit in the endpoint status register. If endpoint 6 is not full, then the data is transferred. This decision is executed by the following statements:

```
if (!(EP2468STAT & bmEP2EMPTY))
{// check EP2 EMPTY (busy) bit in EP2468STAT (SFR), core set's this bit when
// FIFO is empty
if (!(EP2468STAT & bmEP6FULL))
{// check EP6 FULL (busy) bit in EP2468STAT (SFR), core set's this bit
// when FIFO is full
```

The data pointers are initialized to the corresponding buffers. The first auto-pointer is initialized to the first byte of the endpoint 2 FIFO buffer. The second auto-pointer is initialized to the first byte of the endpoint 6 FIFO buffer. The number of bytes to be transferred is read from the byte count registers of endpoint 2. The registers EP2BCL and EP2BCH contain the number of bytes written into the FIFO buffer by the host. These two registers give the byte count of the data transferred to the FIFO in an OUT transaction as long as the data is not

committed to the peripheral side. This data pointer initialization and loading of the count is done in the following statements:

```
APTR1H = MSB( &EP2FIFOBUF ); // Initializing the first data pointer
APTR1L = LSB( &EP2FIFOBUF );
AUTOPTRH2 = MSB( &EP6FIFOBUF ); // Initializing the second data pointer
AUTOPTRL2 = LSB( &EP6FIFOBUF );
count = (EP2BCH << 8) + EP2BCL; // The count value is loaded from the byte //
count registers
```

The data transfer is carried out by the execution of the following loop:

```
for( i = 0x0000; i < count; i++ )
{
// setup to transfer EP2OUT buffer to EP6IN buffer using AUTOPOINTER(s)
EXTAUTODAT2 = EXTAUTODAT1;
}
```

Because auto-pointers have been enabled, the pointers increment automatically, and the statement

```
EXTAUTODAT2 = EXTAUTODAT1;
```

transfers data from endpoint 2 to endpoint 6. Each time the statement is executed, the auto-pointer is incremented. It is executed repeatedly to transfer each byte from endpoint 2 to endpoint 6. After the data is transferred, endpoint 2 has to be rearmed to accept a new packet from the host. Endpoint 6 has to be committed, making the FIFO buffers available to the host for reading data from endpoint 6. This is accomplished by the following statements:

```
EP6BCH = EP2BCH;
SYNCDELAY;
EP6BCL = EP2BCL; // commit EP6IN by specifying the number of bytes the
host can read from EP6
SYNCDELAY;
EP2BCL = 0x80; // re (arm) EP2OUT
```

The same operation is carried out to implement a data loop with endpoints 4 and 8.

High-speed USB implements a PING-NAK mechanism for (bulk and control) OUT transfers. When the host wishes to send OUT data to an endpoint, and the previous data transfer was answered with a NYET, it first sends a PING token to see if the endpoint is ready (for example, if it has an empty buffer). If a buffer is not available, the FX2LP returns a NAK handshake. PING-NAK transactions continue to occur until an OUT buffer is available, at which time the FX2LP answers a PING with an ACK handshake and the host sends the OUT data to the endpoint. EZ-USB implements PING-NAK interrupt as EP0PING, EP1PING, and so on, one for each endpoint. The EPxPING interrupt is asserted when the host PINGs an endpoint and the FX2LP responds with a NAK because the particular endpoint buffer memory is not available. The FX2LP firmware framework provides hooks for all the interrupts that it implements. The example project uses the ISR_Ep2pingnak and ISR_Ep4ping-nak ISRs to handle EP2PING and EP4PING interrupts respectively.

```
void ISR_Ep2pingnak(void) interrupt 0
{
SYNCDELAY; // Rearm endpoint 2
EP2BCL = 0x80;
```

Firmware Examples

```
EZUSB_IRQ_CLEAR(); // clear the EP2PING interrupt
NAKIRQ = bmEP2PING;
}
```

The ISR_Ep2pingnak discards the previous data that is stored in one of the buffers of endpoint 2 by rearming the endpoint (that is, EP2BCL = 0x80). Therefore, EP2 can now receive the data that is currently being sent by the host because there is space available in one of its buffers. It then clears the interrupt by setting a particular bit in NAKIRQ because it has been serviced. The same operation is carried out to service the EP4PING interrupt in ISR_Ep4pingnak.

6.5.2 Building Firmware Example Code for EZ-USB RAM and EEPROM

Click the **Build Target** button at the top right corner of the IDE. The “Total Code Bytes” of the pingnak firmware example is less than the 4-KB code limit Keil µVision2 IDE provided with the kit. The output of the Build Target is *pingnak.hex* and *pingnak.iic* files.

6.5.3 Method to Download Firmware Image to EZ-USB Internal RAM and External EEPROM

Refer to sections **Download Firmware Image to External I2C EEPROM** and **Download Firmware Image to Internal RAM**. After downloading, the firmware re-enumerates with the PC using its internal VID/PID 0x04B4/0x1004.

6.5.4 Binding Cypress USB Driver for the Downloaded Firmware Image

The pingnak project uses vendor class (0xFF) with VID/PID 0x04B4/1004. This example should bind with the generic USB driver, ***cyUSB3.sys***, and the driver information file, ***CyUSB3.inf***, which contains the relevant VID/PID of this example. Follow the procedure outlined to manually bind the driver using the Windows Hardware Wizard. If you have performed the binding process for any one of the firmware examples, you can skip it for this example.

6.5.5 Testing the Pingnak Firmware Functionality

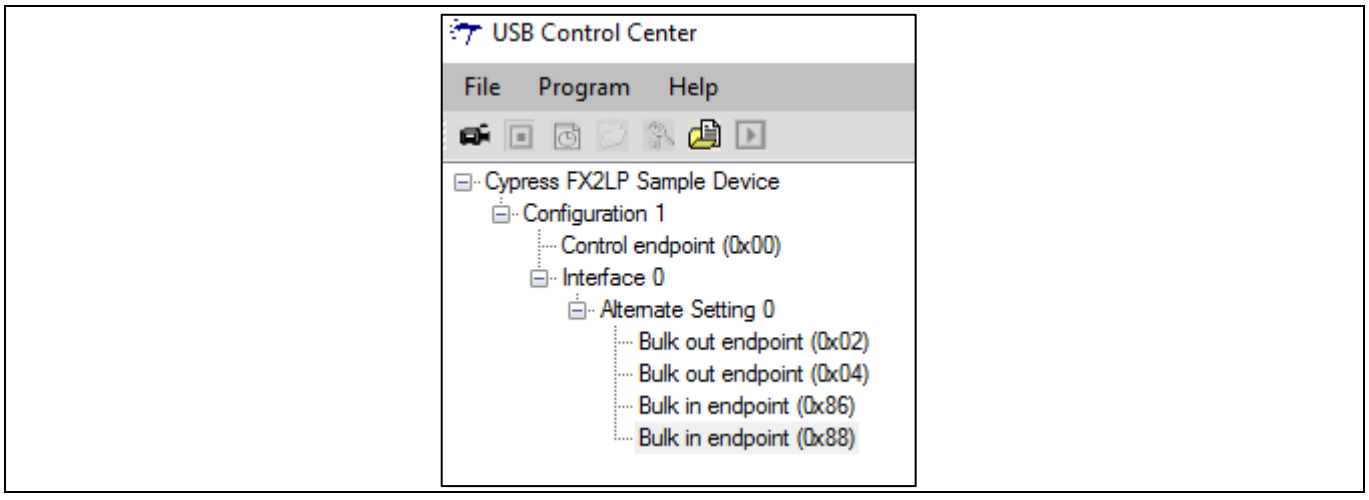


Figure 45 Pingnak Firmware Device Internals

Firmware Examples

Follow these steps to test the pingnak firmware with a USB 2.0 connection.

1. After the board re-enumerates, use the USB Control Center to send 512 bytes from EP2 to EP6. The data received should be the same as the data sent. You can send 512 bytes of user-defined data from the host to endpoint 2 using the USB Control Center. For example, select **Bulk out endpoint (0x02)** in the left pane of the USB Control Center and select the **Data Transfers** tab. In the Data Transfers tab, enter the **Bytes to Transfer** as '512' and then click the **Transfer Data-OUT** button.
2. This data can be read back from endpoint 6 using the USB Control Center. For example, select **Bulk in endpoint (0x86)** in the left pane of the USB Control Center and select the **Data Transfers** tab. In the Data Transfers tab, enter the number of **Bytes to Transfer** as '512' and then click the **Transfer Data-IN** button to read back the data. Similarly, loopback using endpoint 4 and 8 can also be tested. Because EP2 and EP4 are double buffered, they can contain only two packets of data. After sending a packet to these endpoints when both the buffers are full, the endpoints NAK the transfer because there is no space available. This asserts the PING-NAK interrupt of the NAKing endpoint.

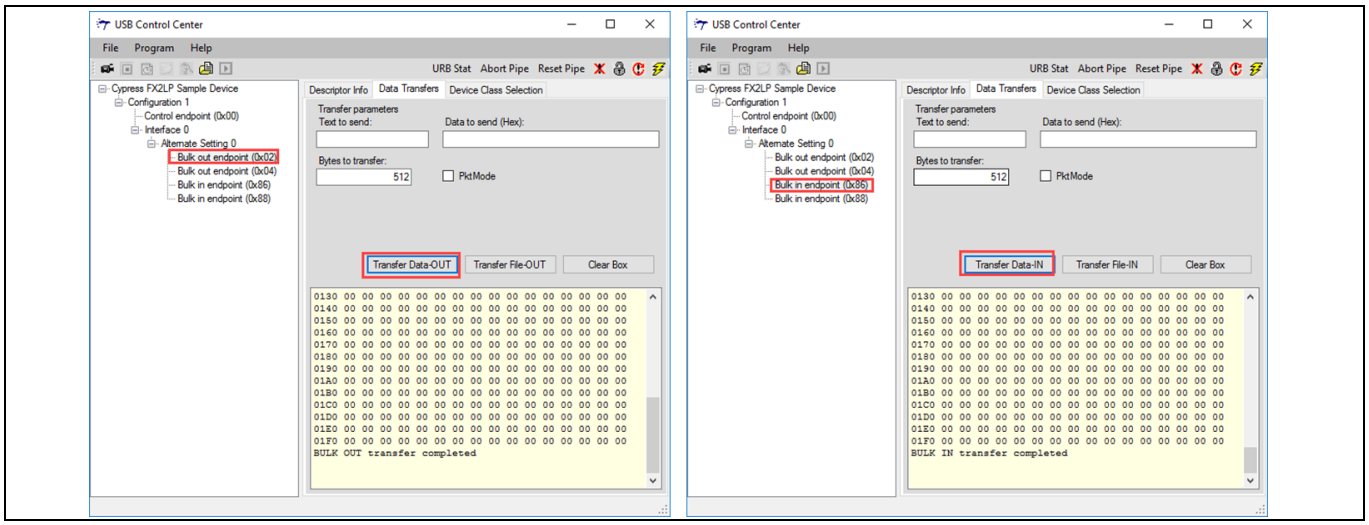


Figure 46 Pingnak Firmware Transfer EP2-EP6

3. The ISRs that handle the PING-NAK interrupt (ISR_Ep2pingnak and ISR_Ep4pingnak) discard the previous data that is stored in one of the endpoint buffers by rearming the endpoint. Therefore, the endpoints can receive the data that is currently sent by the host because there is space in one of its buffers.
4. You can test the previous steps by continuously sending data to EP2 and EP4 without reading the data out of EP6 or EP8. Because the PING-NAK ISR rearms the endpoints, you can continuously transmit data to EP2 and EP4, and the transfer always succeeds. The data present in the buffers of EP2 and EP4 at any point of time will be the latest two packets of data sent from the host.

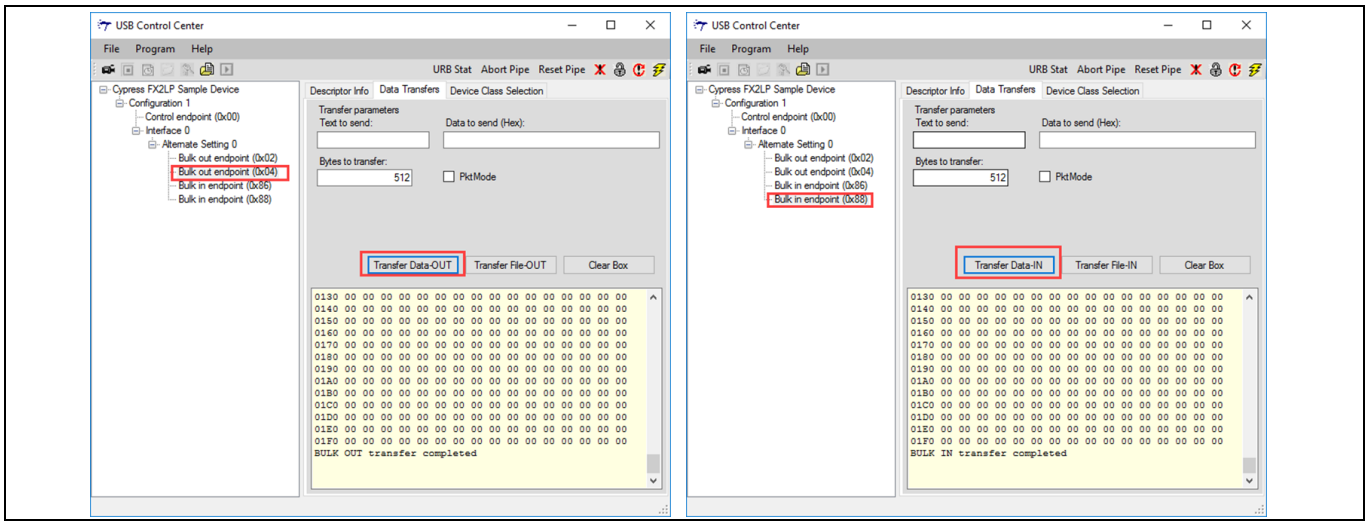


Figure 47 Pingnak Firmware Transfer EP4-EP8

6.6 Vend_ax Firmware Example

6.6.1 Description

This example demonstrates the use of different vendor commands. Vendor commands are used to accomplish unique tasks, such as EZ-USB reset, RAM download, setting a different frequency for the FX2LP I2C interface, communicate with an external SRAM, and so on. The vendor commands are defined in the *vend_ax.c* source file of the example. Open the project by clicking on *vend_ax.uv2*, located at *<example project>\Firmware\vend_ax* and observe the vendor commands implemented in the C routine - DR_VendorCmnd (void). Following are the vendor commands defined in the *vend_ax.c* file:

Table 3 Vendor Command Definitions in vend_ax Example

Sl. No.	Vendor Command/Macro Definition	Function
1	0xA3/ VR_RAM	Downloads data to internal or external RAM
2	0xA6/VR_GET_- CHIP_REV	Retrieves the current revision of EZ-USB FX2LP
3	0xA8/VR_RENUM	The EZ-USB device disconnects and reconnects.
4	0xA9/VR_DB_FX	Selects double-byte addressed large EEPROM U5 and the contents can be uploaded or downloaded to EEPROM
5	0xAA/VR_I2C_100	Sets the I2C interface to 100 kHz
6	0xAB/VR_I2C_400	Sets the I2C interface to 400 kHz

6.6.2 Building Firmware Example Code for EZ-USB RAM and EEPROM

Click the **Build Target** button at the top right corner of the IDE. The “Total Code Bytes” of the vend_ax firmware example is less than the 4-KB code limit Keil µVision2 IDE provided with the kit. The output of the Build Target is *vend_ax.hex* and *vend_ax.iic* files.

6.6.3 Method to Download Firmware Image to EZ-USB Internal RAM and External EEPROM

Refer to sections **Download Firmware Image to External I2C EEPROM** and **Download Firmware Image to Internal RAM**. After downloading, the firmware re-enumerates with the PC using its internal VID/PID 0x04B4/0x1004.

6.6.4 Testing the vend_ax Example

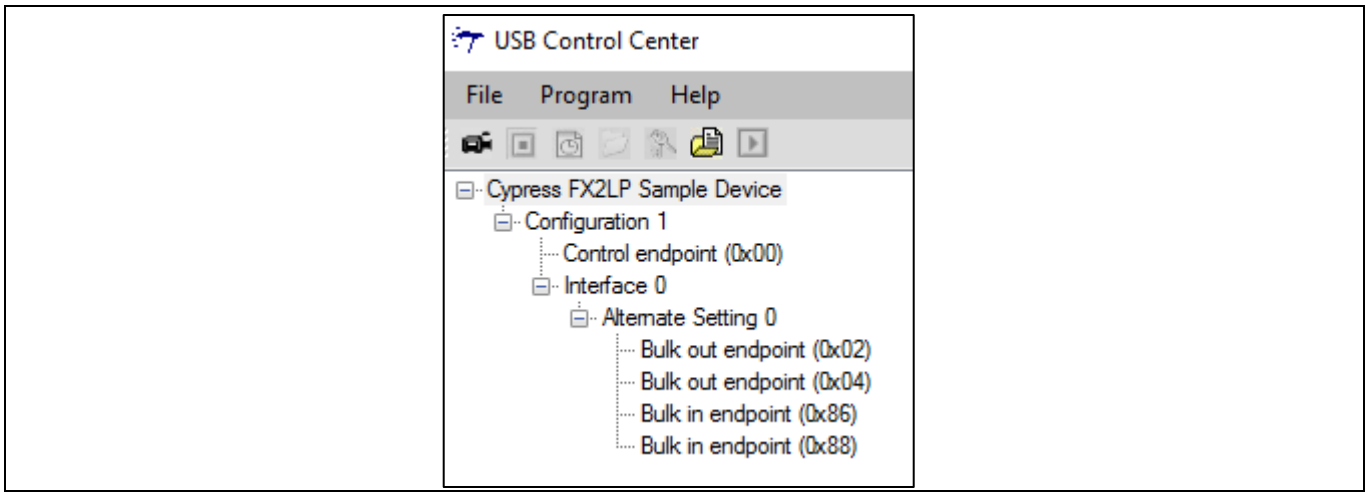


Figure 48 Vend_ax Firmware Device Internals

6.6.4.1 0xA3 command-Download data to RAM

This command is used to download data to either the EZ-USB internal (0x0000–0x3FFFF) RAM or the external RAM memory.

To write the contents to RAM, select Direction = Out, Req Type = Vendor, Target = Device, Bytes to Transfer = 8, and Req Code = 0xA3, and enter data to send as **C2 B4 04 84 00 01 00 11** in the **Data to send (Hex)** box. Click the **Transfer Data** button and observe the RAM getting programmed. **Figure 49** summarizes the entire operation.

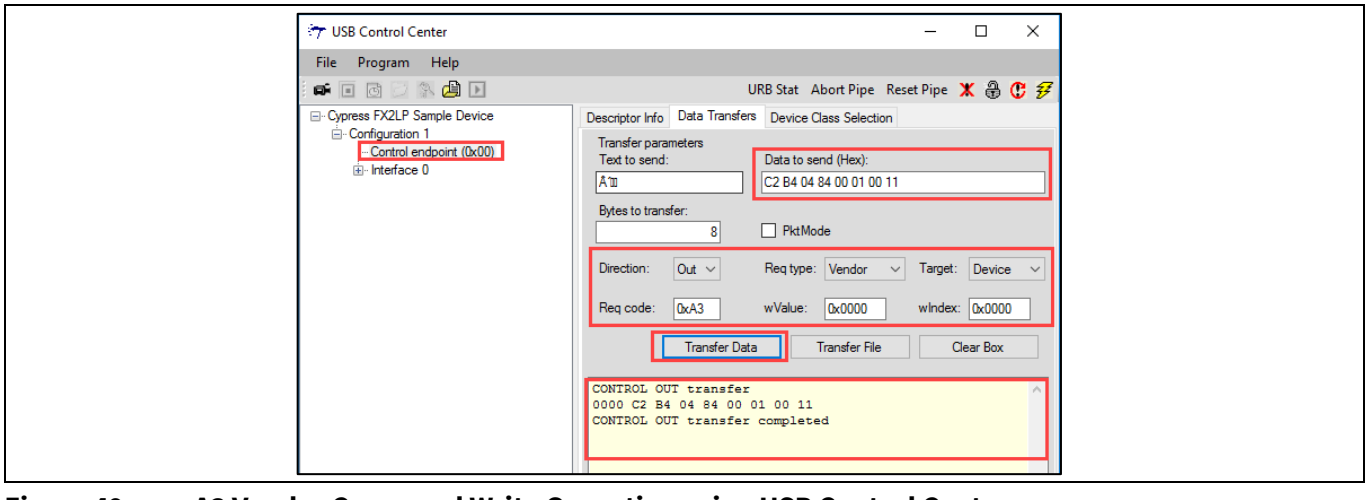


Figure 49 A3 Vendor Command Write Operation using USB Control Center

Firmware Examples

To read the contents from RAM, select Direction = In, Req Type = Vendor, Target = Device, Bytes to Transfer = 8, and Req Code = 0xA3. Click the **Transfer Data** button and observe that the RAM written previously matches the read data. **Figure 50** summarizes the entire operation.

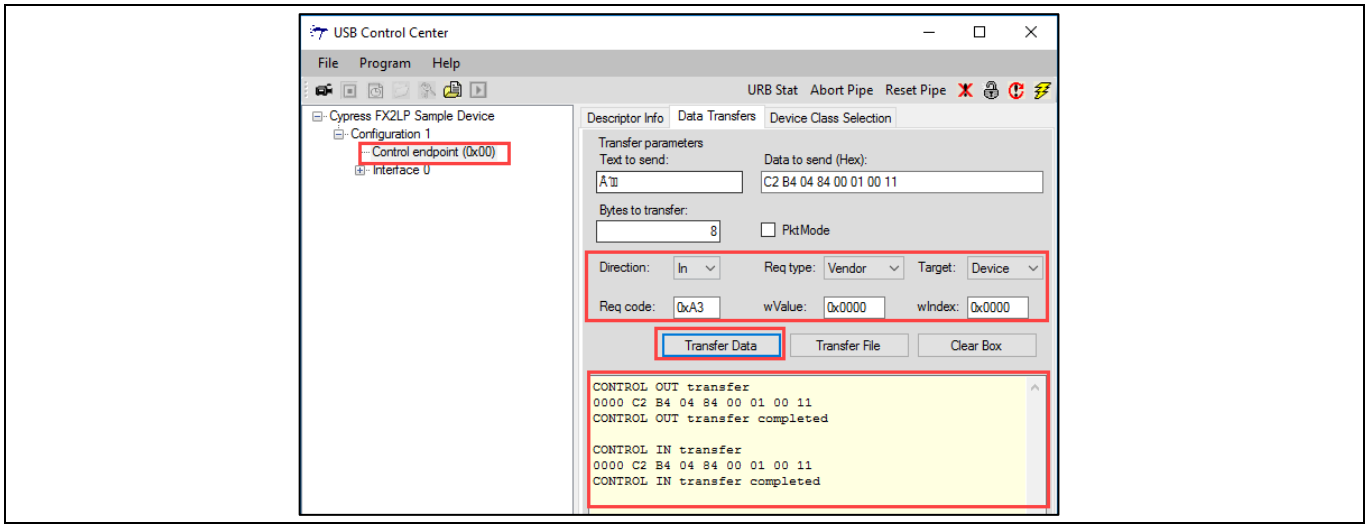


Figure 50 A3 Vendor Command Read Operation using USB Control Center

6.6.4.2 0xA6 command – Get Chip Revision

To retrieve the current revision of the EZ-USB FX2LP) device, this command is used. **Figure 51** summarizes the entire operation using the USB Control Center.

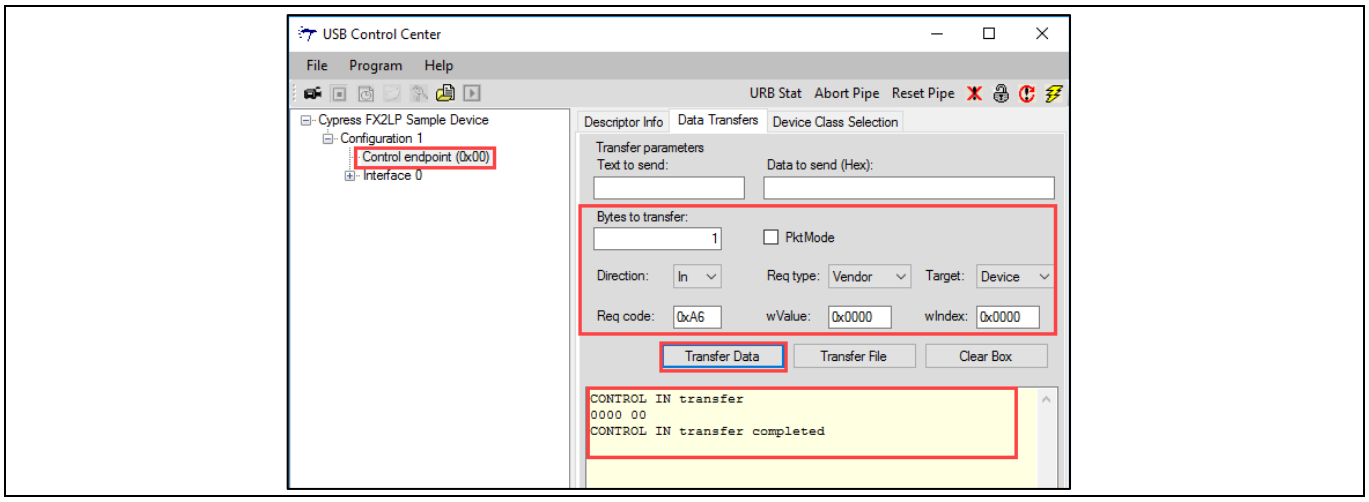


Figure 51 A6 Vendor Command using USB Control Center

6.6.4.3 0xA8 command – EZ-USB Re-enumeration

This command is used to disconnect and re-connect the EZ-USB FX2LP device using the CPUCS register. The EZ-USB re-enumerates. Observe the Cypress device disappearing from the USB Control Center window and reappearing in the same window. **Figure 52** summarizes the command trigger using the USB Center.

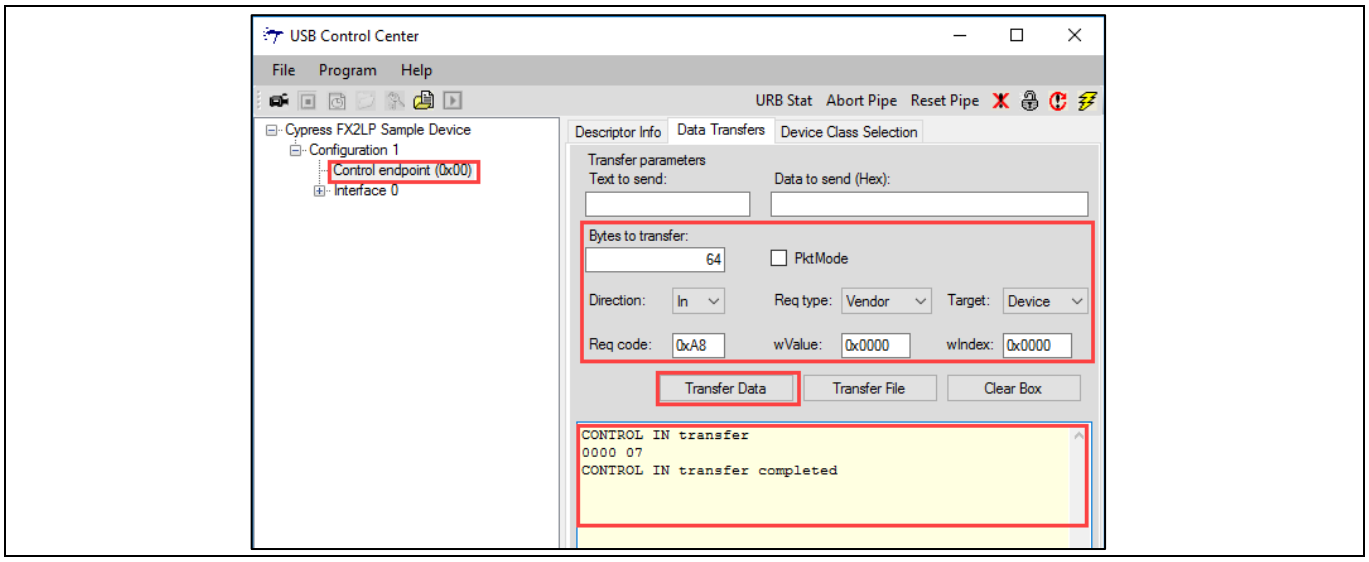


Figure 52 A8 Vendor Command Operation using USB Control Center

6.6.4.4 0xA9 command – Read/Write Large EEPROM

To read/write the contents of Large EEPROM, select Direction = In/OUT, Req Type = Vendor, Target = Device. Bytes to Transfer automatically gets updated if there is pre-defined data.

C2 47 05 31 21 00 00 04 00 03 00 00 02 14 00 00 41 14 00 75 81 5F 91 07 80 FC 90 88 00 E0 91 38 90 98 00 E0 91 38 90 A8 00 E0 91 38 90 B8 00 E0 91 38 90 80 00 E0 91 38 90 90 00 E0 91 38 90 A0 00 E0 91 38 90 B0 00 E0 91 38 22 7A 04 D8 FE D9 FC DA FA 22 80

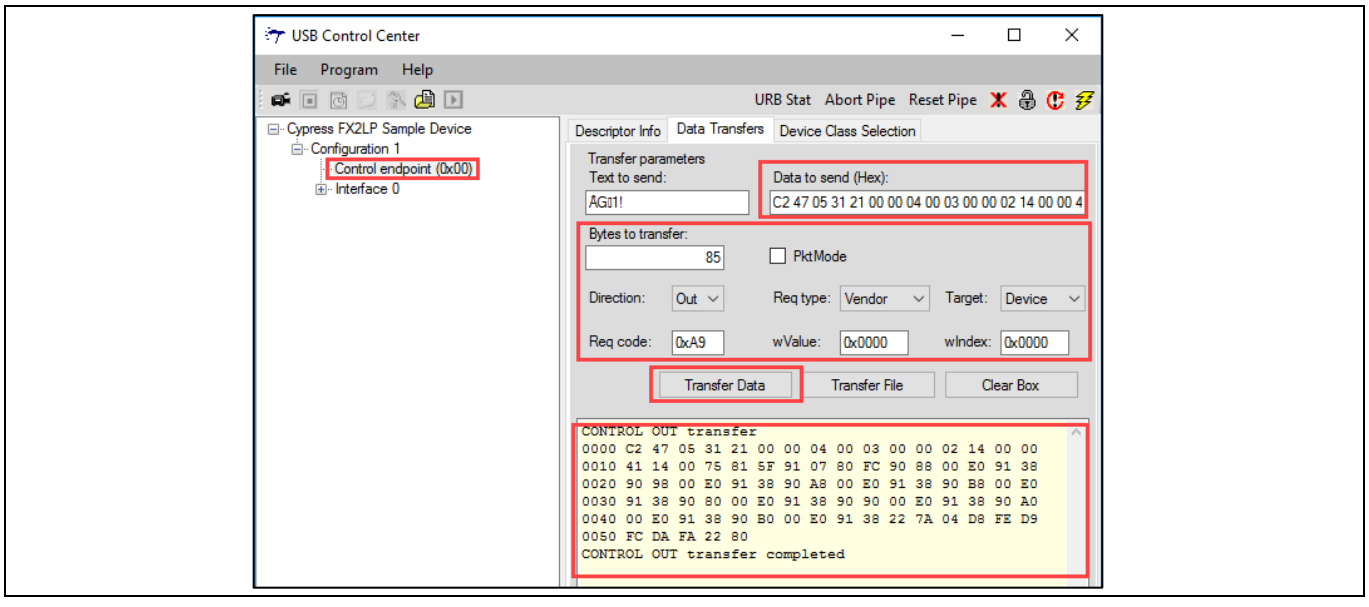


Figure 53 A9 Vendor Command Write Operation using USB Control Center

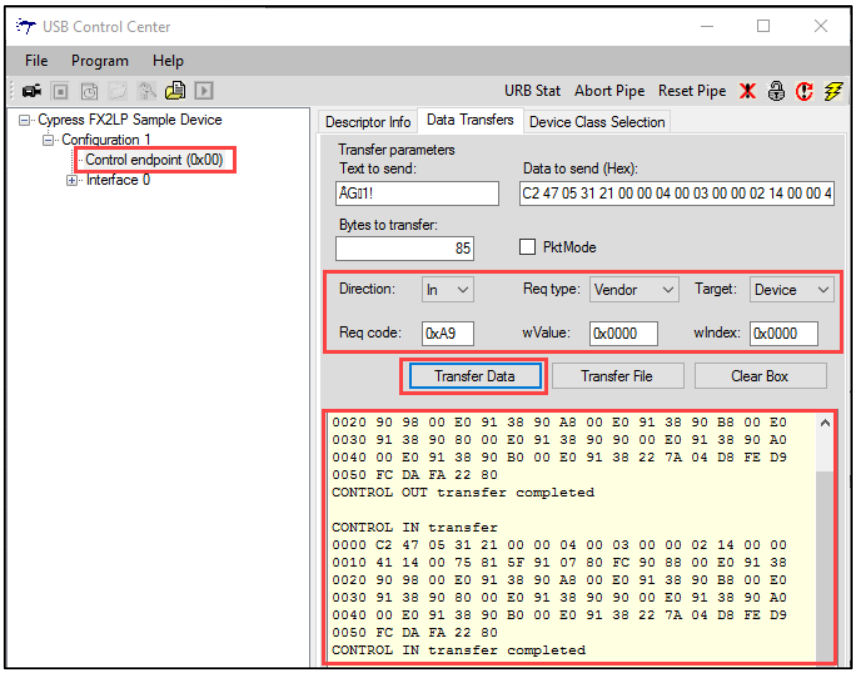


Figure 54 A9 Vendor Command Read Operation using USB Control Center

6.6.4.5 0xAA/0xAB – Setting I2C interface frequency

Using this command, the I2C interface frequency can be set to 100 kHz or 400 kHz. 0xAA command sets the I2C interface frequency to 100 kHz and 0xAB command sets the I2C interface frequency to 400 kHz. **Figure 55** summarizes the command trigger using the USB Control Center.

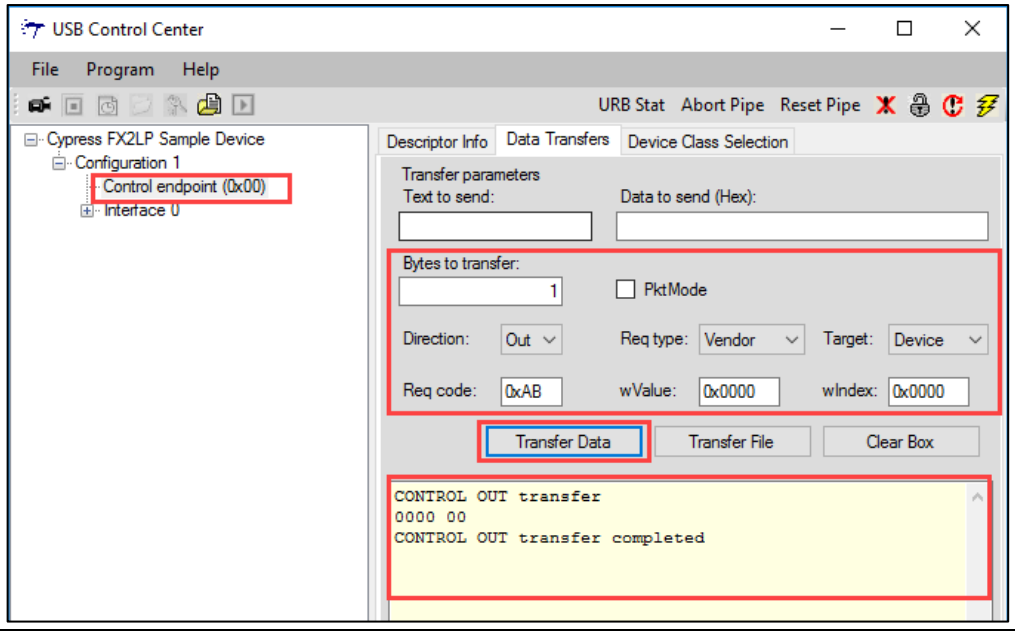


Figure 55 AA/AB Vendor Command Operation using USB Control Center

6.7 KBAs Associated with the Discovery Kit

This section explains the knowledge base articles (KBA) associated with the CY3689 FX2LP Discovery Kit.

6.7.1 KBA229176 - Cypress EZ-USB FX2LP-based Logic Analyzer

This **KBA** explains the procedure to use the CY3689 FX2LP Discovery Kit as a Logic Analyzer using the sigrok PulseView signal analysis tool.

6.7.2 KBA229175 - Debugging of FX2LP Firmware using I2C

The FX2LP I2C controller acts as a master and can be connected to an I2C slave for sending debug messages. This **KBA** discusses the steps to configure and receive debug messages from FX2LP over an I2C interface.

6.7.3 KBA229648 - FX2LP I2C-based Debug with SDCC on Eclipse IDE

This **KBA** discusses the steps to configure the project on Eclipse IDE to receive debug messages from FX2LP over an I2C interface.

6.7.4 KBA30768 - Lattice Crosslink FPGA Configuration and UVC Color Bar Video Streaming

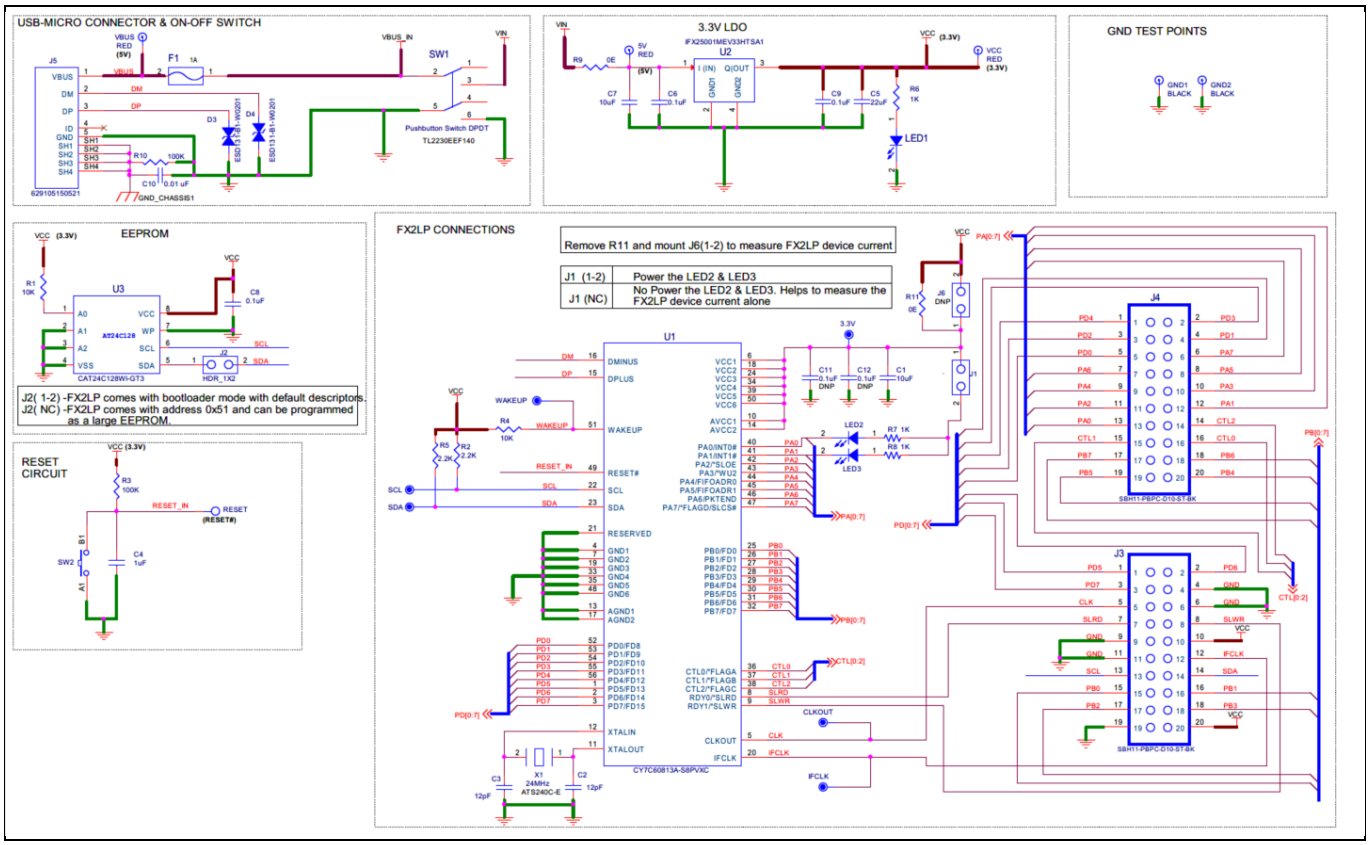
This **KBA** documents the details about implementation of FPGA configuration and UVC framework on FX2LP firmware. It uses Lattice Crosslink FPGA to generate and stream color bar video. The firmware example attached with this KBA binds with the standard UVC driver of the host PC/mobile phone. The color bar is generated on FPGA and streamed by FX2LP firmware.

7 Troubleshooting the Kit

Table 4 Troubleshooting

#	Issue	Possible Cause	Troubleshooting Procedure
1	Discovery kit failed to enumerate as a **Cypress FX2LP No EEPROM Device**	Automatic driver binding failed	Manual Driver Binding. Follow the procedure in **Binding Driver Manually**
		Incorrect driver binding	Right-click the enumerated device and uninstall driver Perform manual driver binding
2	No enumeration	Discovery Kit not powered	Check whether power LED (LED1) is ON; if LED1 is not ON, press SW1 to power the kit
		Faulty USB cable	Change the USB Micro-B cable
		Faulty USB port on the PC	Try on a different USB port on the PC
		Firmware programmed in the kit having different VID/PID	• Fix the firmware with compatible logic and VID/PID • Rebuild the project • Remove J2 jumper and reset the kit • Kit enumerates as a Cypress FX2LP No EEPROM Device • Reconnect J2 and program with correct firmware
		Firmware programmed in the kit is not compatible with kit resources or firmware logic incorrect	
3	Programming failed	Incorrect jumper setting	Refer to **Jumper Settings** for the correct jumper settings to program external EEPROM or internal RAM
		Incorrect file format	EEPROM firmware update requires ***.iic*** file and RAM firmware update requires ***.hex file.*** Choose the correct file and retry programming

Appendix – Board Schematic



Revision History

Document version	Date of release	Description of changes
**	06-11-2020	Initial version of guide