CY3687

MoBL-USB® FX2LP18 Development Kit User Guide

Doc. # 001-68582 Rev. *B

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
www.cypress.com

Contents

1. Introduction

1.1 Introduction

The CY3687 MoBL-USB™ FX2LP18 Development Kit (DVK) is the best starting point for developing a MoBL-USB based product. The DVK includes a development board, code example, a generic device driver, documentation, and assorted tools. This guide provides a general overview and installation help for the DVK. The software installation of the kit includes additional help files and documentation more specific to the various components in the kit.

1.2 Kit Contents

The following list shows the components supplied in the CY3687 MoBL-USB FX2LP18 Development Kit. They represent most of the development tools required to build a USB system.

1.2.1 Hardware

- MoBL-USB FX2LP18 development board
- Prototyping board ('breadboard'). This board is compatible with the EZ-USB (FX1/FX2LP) developments kits as well.
- USB cable
- RS-232 cable
- Software Installation CD-ROM
- Quick start Guide Booklet
- 3 sample MOBL-USB FX2LP18 IC(CY7C68053-56BAXI)

1.2.2 Software on CD-ROM

- Firmware library and firmware frameworks
- Firmware sample code
- Microsoft certified Signed Cypress generic USB driver (3.4.5.000) for Windows XP, Vista and 7 OS platforms.
- Cypress USB C++ library (CyApi.lib)
- SuiteUSB 3.4.7 Development tools for Visual Studio
- Cypress GPIF Designer
- Cypress firmware and Keil monitor download driver sample
- MOBL-USB FX2LP18 documentation
- Keil uVision2 trail version IDE with 4k code limit

1.2.3 Required Tools Not Included

- Full retail Keil Development System (Keil uVision2)
- Microsoft Visual C++ (all PC sample codes are developed on this platform)
- USB capable PC running Windows XP or 2000

1.2.4 Other Suggested Tools

- CATC USB Protocol Analyzer.

1.3 Document Revision History

Table 1-1. Revision History

Revision	PDF Creation Date	Origin of Change	Description of Change
**	02/07/2011	ROSM	Initial version of user guide
*A	06/21/2012	NMMA	The document has to be updated with the OOB review comments.
*B	06/28/2012	NMMA	Minor text edits as per IC samples listed in UG

1.4 Documentation Conventions

Table 1-2. Document Conventions for Guides

Convention	Usage
`Courier New`	Displays file locations, user entered text, and source code: `C:\ ...cd\icc\`
Italics	Displays file names and reference documentation: Read about the *sourcefile.hex* file in the *PSoC Designer User Guide*.
[**Bracketed, Bold**]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]
File > Open	Represents menu paths: File > Open > New Project
Bold	Displays commands, menu paths, and icon names in procedures: Click the **File** icon and then click **Open**.
Times New Roman	Displays an equation: $2 + 2 = 4$
Text in gray boxes	Describes Cautions or unique functionality of the product.

2. Getting Started

This chapter describes the installation of the CY3687 MoBL-USB FX2LP18 development Kit CD/DVD Software.

2.1 Kit Installation

To install the kit software, follow these steps:

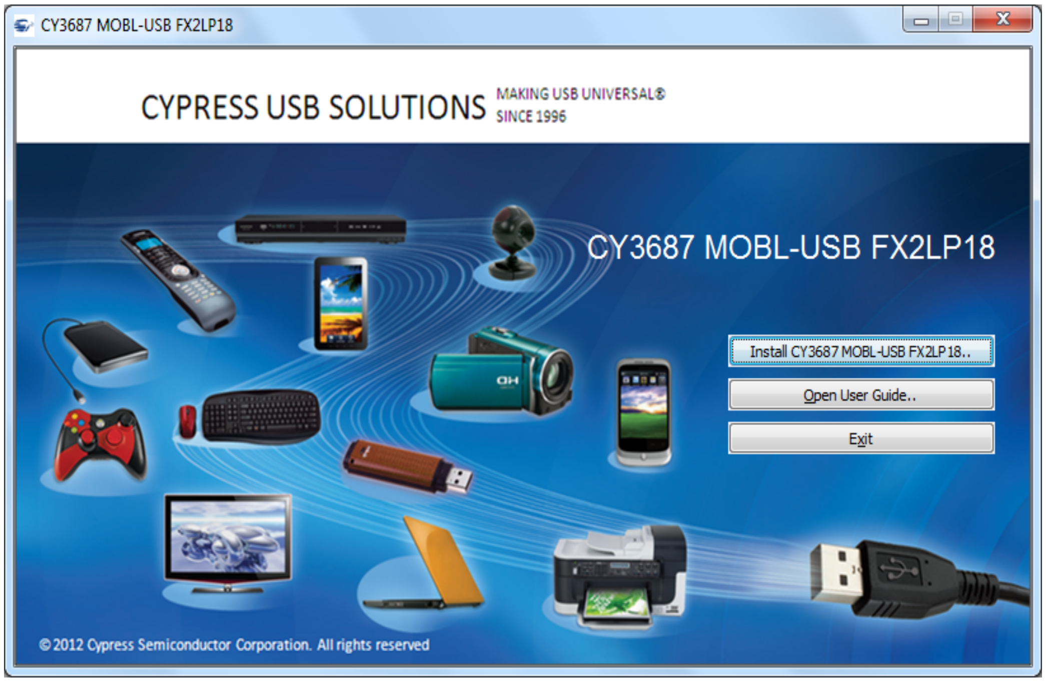
1. Insert the kit CD/DVD into the CD/DVD drive of your PC. The CD/DVD is designed to auto-run and the kit installer startup screen appears.

   You can also download the latest kit installer ISO file for CY3687

   Create an installer CD/DVD or extract the ISO using WinRar and install the executables.

2. Click **Install CY3687 MoBL-USB™ FX2LP18 DVK** to start the installation, as shown in Figure 2-1.

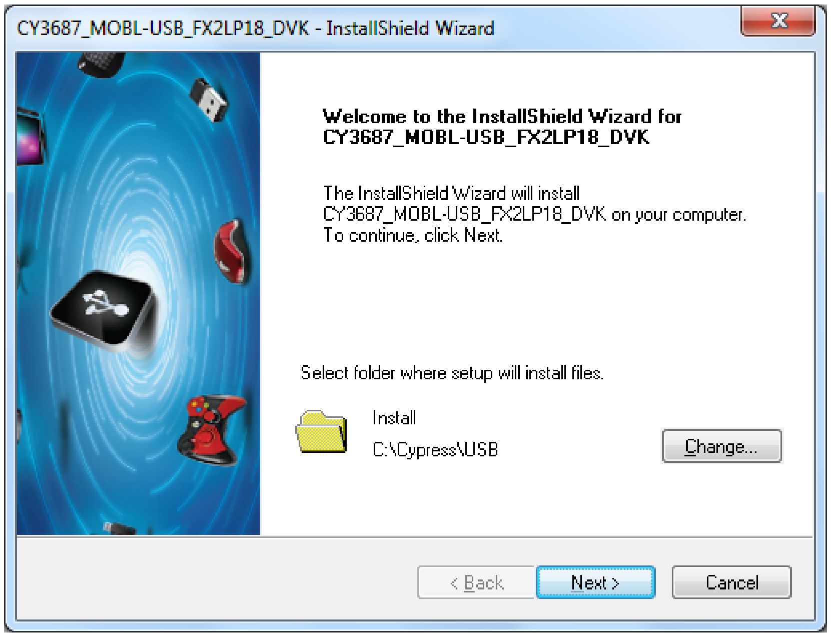
   Figure 2-1. Kit Installer Startup Screen



   Note: If auto-run does not execute, double-click on the cyautorun.exe file in the root directory of the CD.

3. The InstallShield Wizard screen appears. The default location for setup is shown on the InstallShield Wizard screen. You can change the location for setup using **Change**, as shown in Figure 2-2. Click **Next** to launch the kit installer.

Figure 2-2. InstallShield Wizard



CY3687_MOBL-USB_FX2LP18_DVK - InstallShield Wizard

Welcome to the InstallShield Wizard for CY3687_MOBL-USB_FX2LP18_DVK

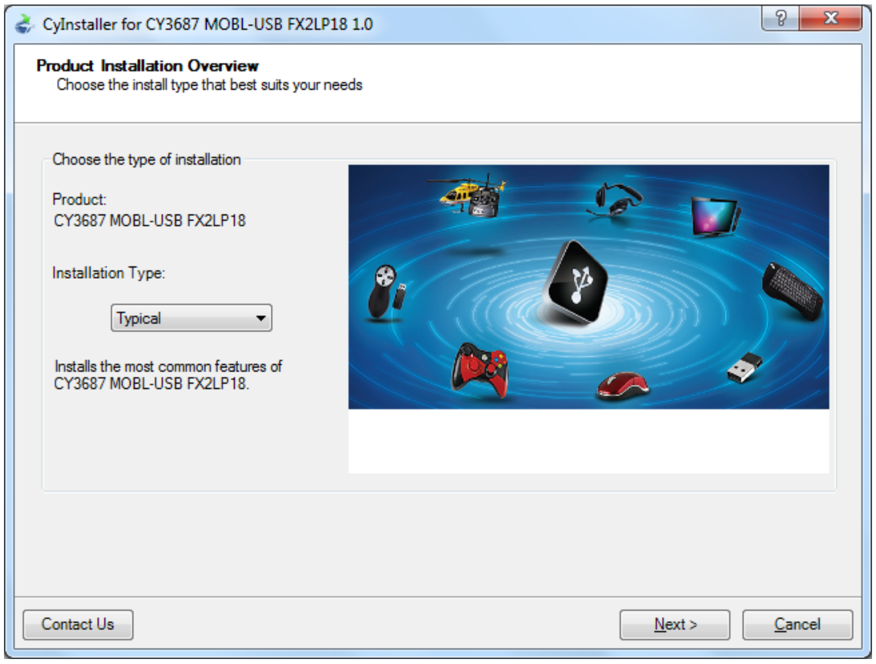
The InstallShield Wizard will install CY3687_MOBL-USB_FX2LP18_DVK on your computer. To continue, click Next.

Select folder where setup will install files.

Install
C:\Cypress\USB

4. On the Product Installation Overview screen, select the installation type that best suits your requirement. The drop-down menu has three options - **Typical**, **Complete**, and **Custom**, as shown in Figure 2-3. In the current installer all 3 installation types would result in same set of softwares getting installed.Select the default **Typical** installation and click **Next**.

Figure 2-3. Installation Type Options



CyInstaller for CY3687 MOBL-USB FX2LP18 1.0

Product Installation Overview
Choose the install type that best suits your needs

Choose the type of installation

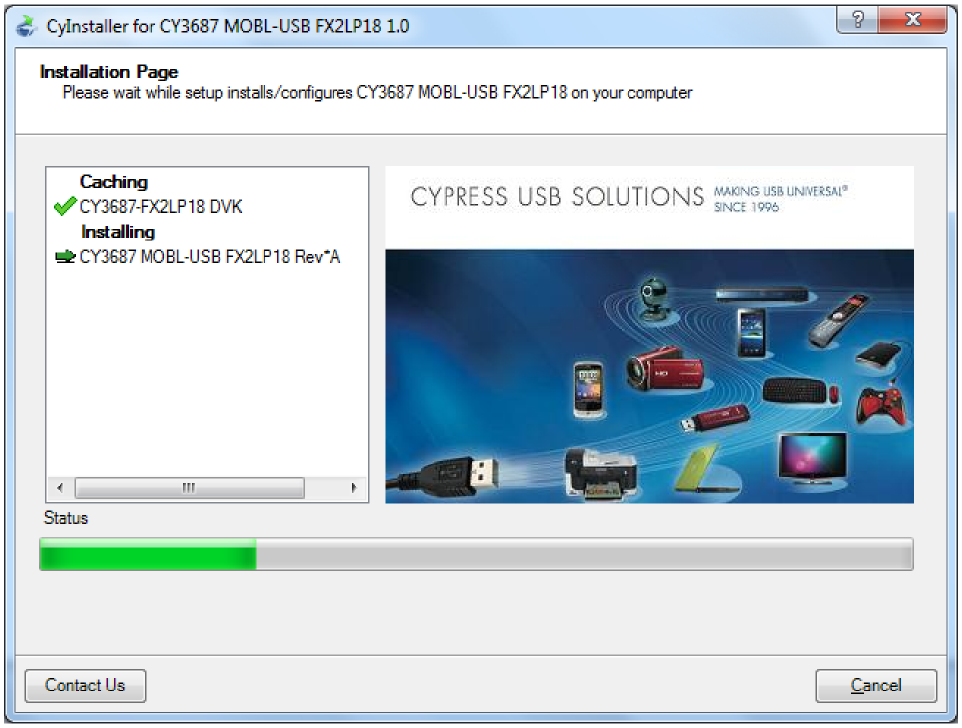
Product:
CY3687 MOBL-USB FX2LP18

Installation Type:
Typical

Installs the most common features of CY3687 MOBL-USB FX2LP18.

5. When the installation begins, all packages are listed on the Installation page. A green check mark appears adjacent to every package that is downloaded and installed, as shown in Figure 2-4. Wait until all the packages are downloaded and installed successfully.

Figure 2-4. Installation Page



CyInstaller for CY3687 MOBL-USB FX2LP18 1.0

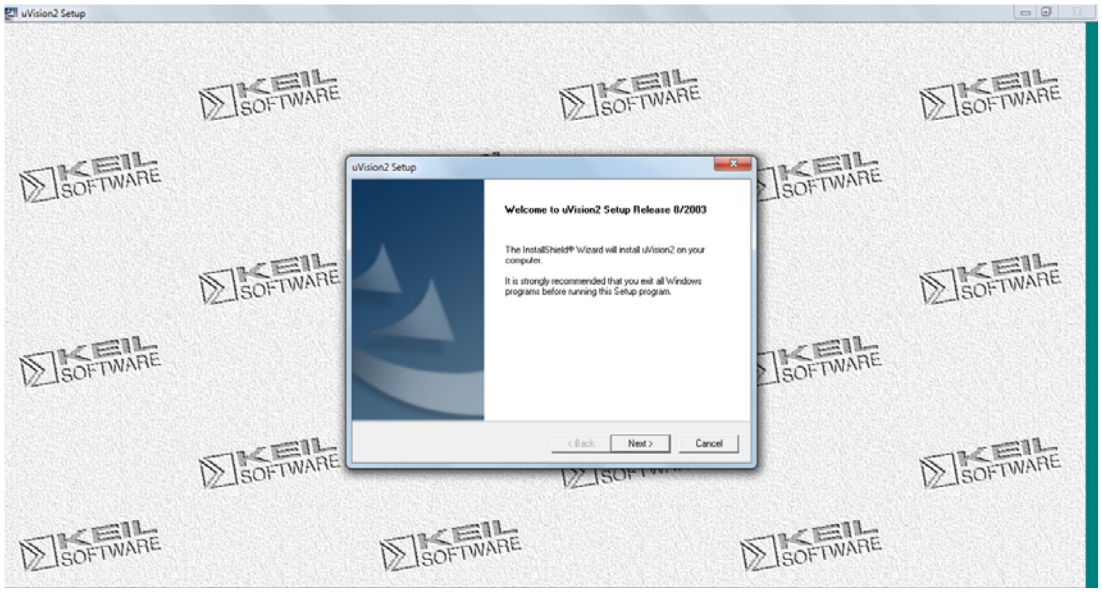
Installation Page
Please wait while setup installs/configures CY3687 MOBL-USB FX2LP18 on your computer

Caching
✓ CY3687-FX2LP18 DVK
Installing
CY3687 MOBL-USB FX2LP18 Rev*A

Status

6. Keil uVision2 trial version IDE triggers at this stage. If the PC already has the software installed then the installer will not trigger the installation. If the PC does not contain the software then keil welcome screen appears as shown in Figure 2-5.Click **Next**.

Figure 2-5. Keil Welcome screen



uVision2 Setup

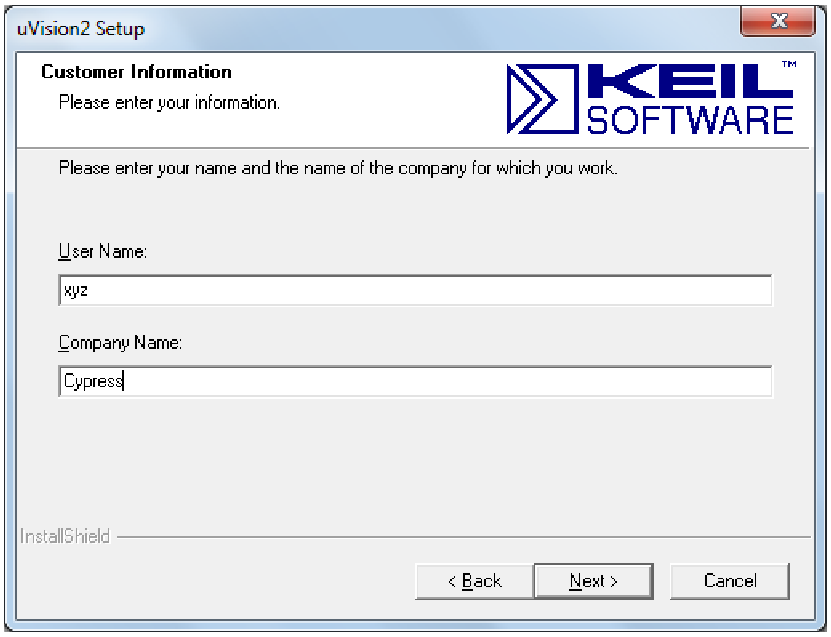
Welcome to uVision2 Setup Release 8/2003

The InstallShield® Wizard will install uVision2 on your computer.

It is strongly recommended that you exit all Windows programs before running this Setup program.

7. Enter the User name and company Name credentials as shown in Figure 2-6 to proceed further with the installation.

Figure 2-6. Keil User Information Input Window



uVision2 Setup

Customer Information

Please enter your information.

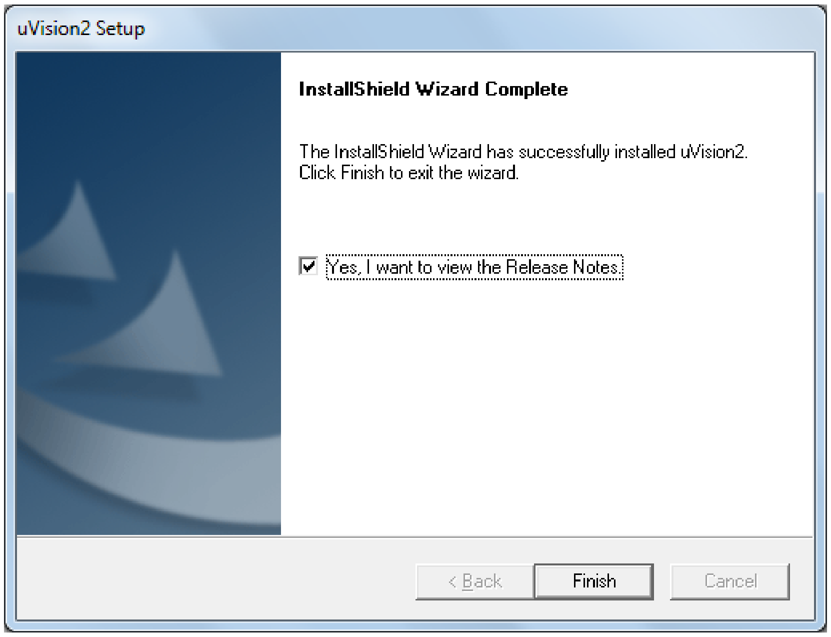
Please enter your name and the name of the company for which you work.

User Name: xyz

Company Name: Cypress

8. The keil software proceeds with the installation and copies necessary packages at default directory **C:\Keil**. After completion click on **Finish** as shown in Figure 2-7.

Figure 2-7. Keil User Information Input Window



uVision2 Setup

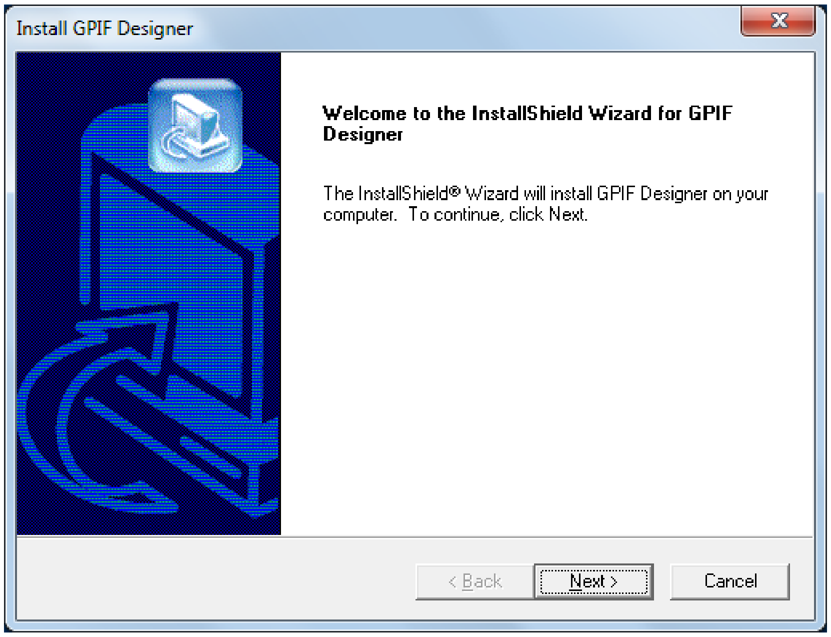
InstallShield Wizard Complete

The InstallShield Wizard has successfully installed uVision2. Click Finish to exit the wizard.

[✓] Yes, I want to view the Release Notes.

9. GPIF designer software is triggered after keil installation.This software is used to create State machine waveforms to communicate between MoBL-USB FX2LP device and devices such as FPGA, Image sensors, FIFO, and so on. If the PC already has the software installed then the installer will not trigger the installation. If the PC does not contain the software then GPIF designer welcome screen appears as shown in Figure 2-8. Click **Next**.

Figure 2-8. GPIF Designer Welcome Window

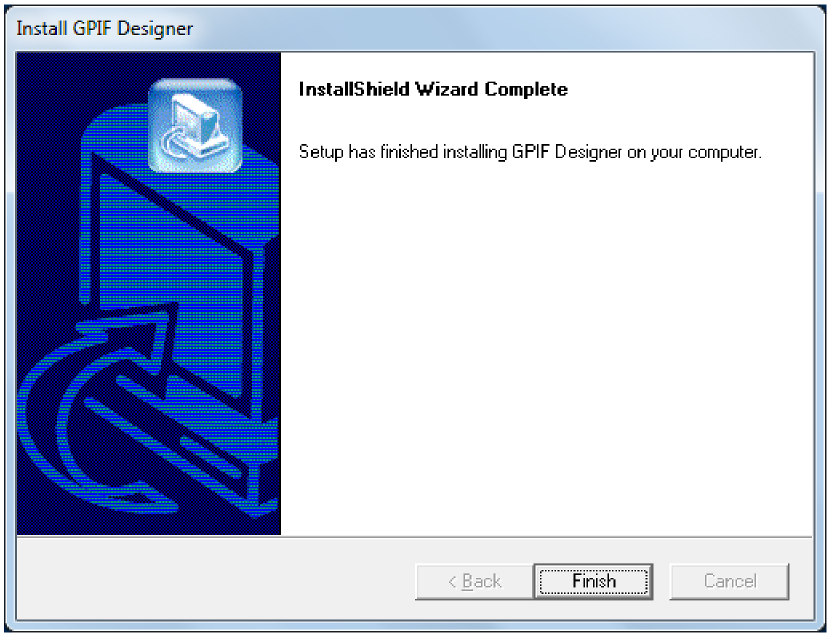

Install GPIF Designer

Welcome to the InstallShield Wizard for GPIF Designer

The InstallShield® Wizard will install GPIF Designer on your computer. To continue, click Next.

10. Click **Next** in the subsequent windows and Finish window appears as shown in Figure 2-9.

Figure 2-9. GPIF Designer Welcome Window

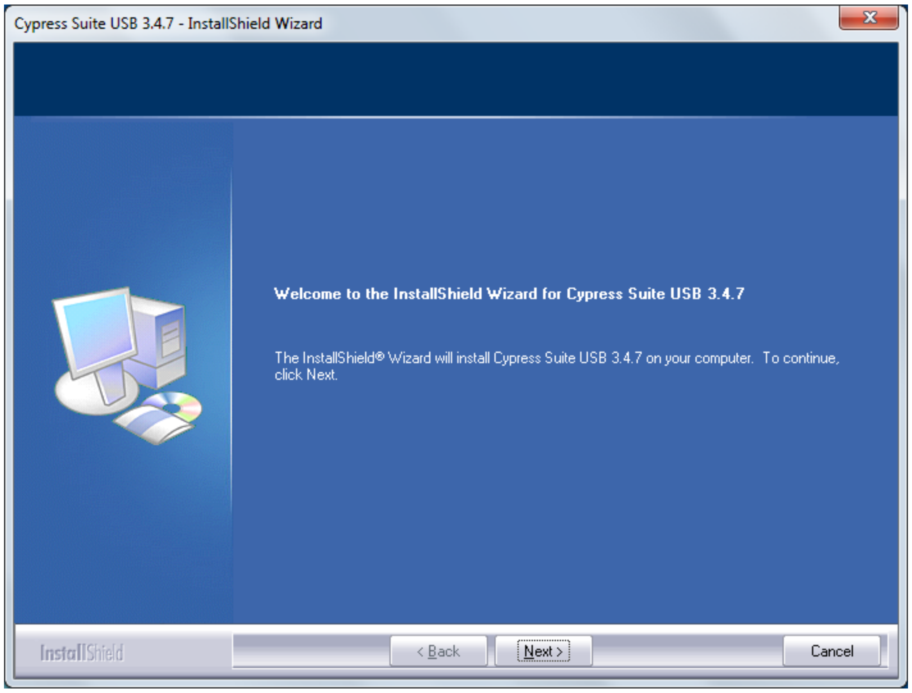

Install GPIF Designer

InstallShield Wizard Complete

Setup has finished installing GPIF Designer on your computer.

11. SuiteUSB 3.4.7 package install shield gets triggered after GPIF designer software installation. If the PC already has the software installed then the installer will not trigger the installation. If the PC does not contain the software then SuiteUSB welcome screen appears as shown in Figure 2-10. Click **Next** and accept Cypress Software license agreement as shown in Figure 2-11.

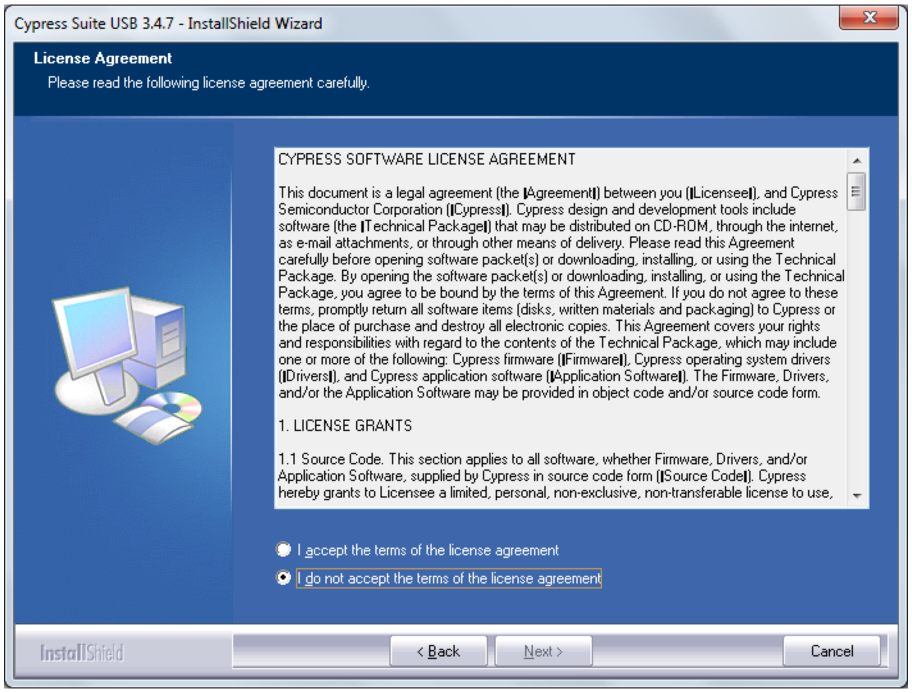
Figure 2-10. SuiteUSB Welcome Window



Welcome to the InstallShield Wizard for Cypress Suite USB 3.4.7

The InstallShield® Wizard will install Cypress Suite USB 3.4.7 on your computer. To continue, click Next.

Figure 2-11. SuiteUSB License Agreement Window



License Agreement

Please read the following license agreement carefully.

CYPRESS SOFTWARE LICENSE AGREEMENT

This document is a legal agreement (the |Agreement|) between you (|Licensee|), and Cypress Semiconductor Corporation (|Cypress|). Cypress design and development tools include software (the |Technical Package|) that may be distributed on CD-ROM, through the internet, as e-mail attachments, or through other means of delivery. Please read this Agreement carefully before opening software packet(s) or downloading, installing, or using the Technical Package. By opening the software packet(s) or downloading, installing, or using the Technical Package, you agree to be bound by the terms of this Agreement. If you do not agree to these terms, promptly return all software items (disks, written materials and packaging) to Cypress or the place of purchase and destroy all electronic copies. This Agreement covers your rights and responsibilities with regard to the contents of the Technical Package, which may include one or more of the following: Cypress firmware (|Firmware|), Cypress operating system drivers (|Drivers|), and Cypress application software (|Application Software|). The Firmware, Drivers, and/or the Application Software may be provided in object code and/or source code form.

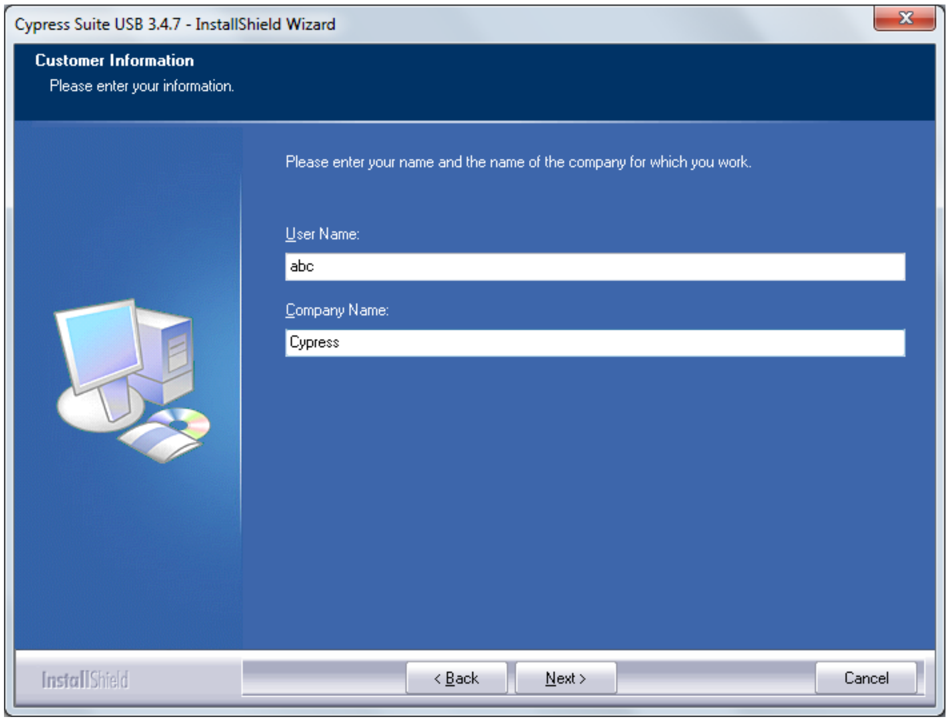
1. LICENSE GRANTS

1.1 Source Code. This section applies to all software, whether Firmware, Drivers, and/or Application Software, supplied by Cypress in source code form (|Source Code|). Cypress hereby grants to Licensee a limited, personal, non-exclusive, non-transferable license to use,

○ I accept the terms of the license agreement
● I do not accept the terms of the license agreement

12. Enter User credentials in the SuiteUSB window as shown in Figure 2-12. Click **Next**. The default directory of the SuiteUSB is **C:\Cypress\Cypress Suite USB 3.4.7**. The default directory can be changed at this stage. Click **Next** after selecting the directory. Click **Install** button in the subsequent window. The SuiteUSB package installation progress is shown in the next window. Finally the SuiteUSB **Finish** window appears. Click **Finish** button to complete the installation process of SuiteUSB.

Figure 2-12. SuiteUSB User Login Window


Cypress Suite USB 3.4.7 - InstallShield Wizard

Customer Information

Please enter your information.

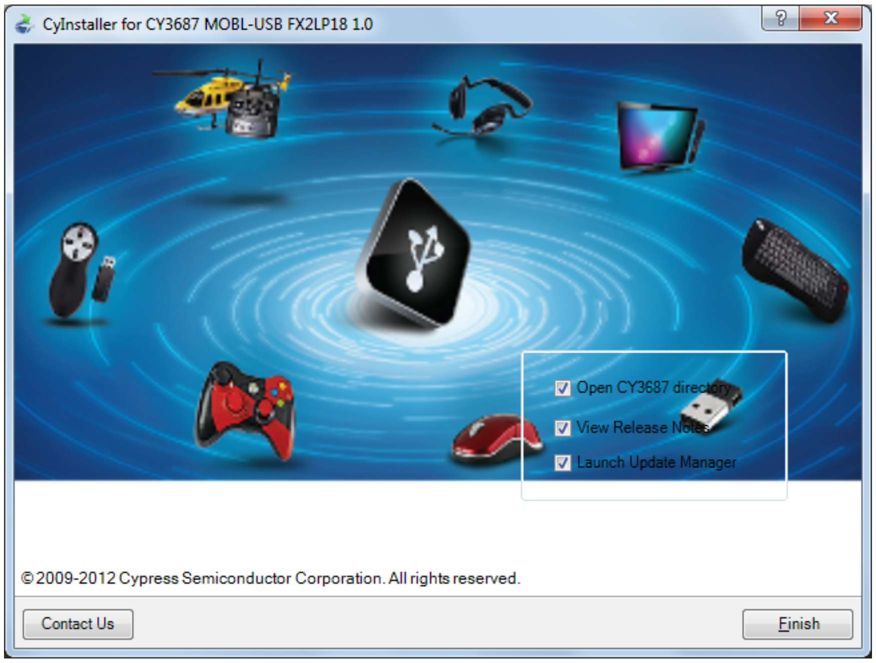
Please enter your name and the name of the company for which you work.

User Name: abc

Company Name: Cypress

13. The CY3687 MoBL-USB FX2LP18 development Kit. CY3687 **Finish** window appears after installing Kit content, Keil software, GPIF designer and SuiteUSB 3.4.7 package.

Figure 2-13. CY3687 Finish Window



☑ Open CY3687 directory
☑ View Release Notes
☑ Launch Update Manager

© 2009-2012 Cypress Semiconductor Corporation. All rights reserved.

2.2 Install Hardware

Refer to section Binding Cypress USB Driver to MoBL-USB Development Board on page 43 for hardware installation for this kit.

3. Development Board

3.1 Introduction

The Cypress Semiconductor MoBL-USB Development Board provides expansion and interface signals on six 20-pin headers. A mating prototype board allows quick construction and testing of USB designs. The board may be powered from the USB connector or an external power supply. Note that some of the signals driven by the MoBL-USB FX2LP18 device on the Advanced Development board have been replaced by VCC-IO and 1.8 V supplies.

The MoBL-USB Development Board is supplied as part of the Cypress Semiconductor MoBL-USB Development Kit, which includes an evaluation version of Cypress-customized software development tools from Keil Software Inc. The Keil 8051 assembler, C Compiler, and debugger work in concert with the development board to provide a complete code development environment. The evaluation version of the Keil tools that ships with the DVK has several restrictions that make it inappropriate for real-world development. Most significantly, it limits the compiled object size to 4 KB. The full retail version allows code of any size.

3.2 Schematic Summary

This description should be read while referring to the MoBL-USB Development Board schematic and the MoBL-USB Development Board Assembly drawing. Both drawings are attached to the end of this document and are available in PDF format in the DVK hardware directory.

U3 is the MoBL-USB 56-pin device. Although there is a large (100-pin) FX2LP on the board (U11), this chip is only used for I^2C to serial translation, it is not available as a USB device.

Power to the MoBL-USB device comes from two or three different supplies. The AVCC pins draw 3.3 V through jumper JP4. The VCCCore supply requires 1.8 V, which it receives from JP2. The I/O pins (VCCIO) can run on voltages ranging from 1.8 V to 3.3 V. JP10 provides the ability to select the input voltage to VCCIO from the 1.8 V, 3.3 V or 2.5 V (adjustable) regulator.

U6, U9, and U10 provide power to the board. All of these devices can provide up to 500 mA, so there is plenty of spare power on all of the supplies to handle any devices on the prototype board. The output voltage of U9 can be varied by changing the ratio of R29 and R34. See the LT1763 data sheet for more information.

U7 and U8 are socketed EEPROMS, used for MoBL-USB FX2LP18 initialization and 8051 general purpose access. U5 is another EEPROM, used for factory initialization. This part is required because the MoBL-USB chip starts up disconnected from USB and it requires an EEPROM load to connect to USB. JP6 prevents this part from accidental programming.

U2 and U4 are Philips PCF8574 IO expanders, which attach to the MoBL-USB FX2LP18 I2C bus and provide eight GPIO pins. U2 reads the four push-button switches S2-S5, and U11 drives the seven-segment readout U1.

U11 is used only for converting the I^2C signals to RS-232 for running the Keil debugger. It is not used for USB access. U12 converts the 3.3 V 8051 serial port signals to bipolar RS-232 levels. U13 contains the I^2C to RS-232 conversion program. U13 is not intended to be user-programmable.

Six 20-pin headers, P1-P6, provide interface signals to the plug-in prototyping board supplied in this kit, as well as serving as connection points for HP(Agilent) logic analyzer pods. P8 contains a subset of signals from P1-P6 on a connector that is pinned out for connection to a ‘straight- through’ ATA cable. Two slide switches, SW1 and SW2, control the connection and selection of the three socketed EEPROMS at U5, U7, and U8.

3.3 Jumpers

Table 3-1. Jumper Settings

Jumper	Function	Default	Notes
JP1	Current measurement point for VCCIO	IN (1-2)	This jumper may be removed and replaced with ammeter probes in series to measure current for this supply.
JP2	Current measurement point for VCCORE	IN (1-2)	This jumper may be removed and replaced with ammeter probes in series to measure current for this supply.
JP3	Removed		This jumper has been removed from the schematic
JP4	Current measurement point for AVCC	IN (1-2)	This jumper may be removed and replaced with ammeter probes in series to measure current for this supply.
JP5	Board power source	IN (1-2)	If this jumper is in place, the board is bus-powered from the USB connector (J2). If this jumper is removed, the board must be powered via JP5.1 or another 5v jumper.
JP6	SAFE_WP	IN (1-2)	Removing this jumper allows the SAFE EEPROM to be reprogrammed. Reprogramming the SAFE EEPROM is not recommended. If the SAFE EEPROM contents are lost, the EEPROMs cannot be reprogrammed via USB.
JP7	EEPROM WP	PA7 (1-2)	This jumper selects the WP input for the ‘normal’ EEPROMs. Position 1-2 ties WP to PA7. In this position, PA7 must be driven low to reprogram the EEPROM. The Cypress programming tools will drive PA7 low during programming. If PA7 is used by your application, you can remove this jumper during normal use and move it to position (2-3) during EEPROM programming.
JP8	Current measurement point or alter-nate power input for 3.3 V supply	IN (1-2)	This jumper may be removed and replaced with ammeter probes in series to measure current for this supply. This point may be used to provide an external source for the 3.3 V supply.

Table 3-1. Jumper Settings

Jumper	Function	Default	Notes
JP9	Current measurement point or alter-nate power input for 2.5V supply	IN (1-2)	This jumper may be removed and replaced with ammeter probes in series to measure current for this supply. This point may be used to provide an external source for the 2.5 V supply.
JP10	Voltage selection for VCCIO	1.8V (5-6)	Selects the input voltage for VCCIO.
JP11	Current measurement point or alter-nate power input for 1.8V supply	IN (1-2)	This jumper may be removed and replaced with ammeter probes in series to measure current for this supply. This point may be used to provide an external source for the 1.8 V supply.
JP12	WP for debug FX2LP	IN (1-2)	When this jumper is in place, the debug firmware in the 100-pin FX2LP is protected from accidental writes. Removing this jumper allows the debug firmware to be overwritten.

3.4 EEPROM Select and Enable Switches SW1 and SW2

SW1 selects between two socketed EEPROMs, one strapped to address 000 (U8), and the other strapped to address 001(U7).

SW2 enables or disables the EEPROM selected by SW1. When the SW1 EEPROMs are disabled, the ‘Safe’ EEPROM is enabled.

The MoBL-USB chip has various start-up modes, which depend on the existence of an EEPROM connected to its SCL and SDA lines. Switches SW1 and SW2 select among three EEPROMs on the board. Each of these EEPROMs has a specific purpose:

- U5 -- SAFE -- Used to select the default VID/PID for the board. Do not overwrite this EEPROM.
- U7 -- Large -- Used for firmware download. User programmable.
- U8 -- Small -- Used for VID/PID programming only. Used for reNumeration or default configura-tion. The VID/PID allows the operating system to identify your device. You must have your own VID assigned by the USB I/F. The ‘Using Cypress’ VID is not permitted.

On reset, the MoBL-USB I2C controller loads the image from one of these three EEPROMs. As this process completes, the 8051 firmware can use the I^2C controller to access the EEPROMs or both devices on the I^2C bus. The MoBL-USB bootloader accommodates two EEPROM types, in ‘Small’ and ‘Large’ versions shown in Table 3-2.

Table 3-2. Typical MoBL-USB external EEPROMS

EEPROM Type	Size	A2A1A0	Typical P/N (2.5-3.3v)	Typical P/N (1.8v)
‘Small’	128x8	000	24LC01	24AA01
	256x8	000	24LC02	24AA02
‘Large’	16Kx8	001	24LC128	24AA128

The MoBL-USB loader determines the EEPROM size by first initiating an I^{2}C transfer to address Alignment not proper between these 2 lines.

If the above transfer does not return an ACK pulse, the MoBL-USB loader initiates a second I^{2}C transfer, this time to address 10100001 (1010=EEPROM, sub-address 001). If an ACK is returned by the I^{2}C device, the MoBL-USB loader writes two EEPROM address bytes to initialize the internal EEPROM address pointer to ‘0’.

If neither transfer returns an ACK pulse, the MoBL-USB Development Board does not connect to USB. MoBL-USB requires a 0xC2 format EEPROM to connect. Three MoBL-USB startup sequences, and the associated settings for SW1 and SW2, are as follows:

1. Safe Mode:SW2 = SAFE, SW1 = either position

   This setting selects the EEPROM located in socket U5. Since the MoBL-USB chip comes out of reset disconnected from USB, an EEPROM is required to connect to the USB. The ‘Safe’ EEPROM is used for this purpose. The ‘Safe’ EEPROM contains simple firmware that connects to the USB and responds to descriptor requests with the Cypress VID and the MoBL-USB PID. ‘Safe’ mode is used to allow the development board to enumerate when no action other than a USB connect is required. For example, the ‘safe’ setting could be used if one of the other EEPROMs on the board is accidentally programmed with malfunctioning firmware. Once it is running, SW2 can be switched to the LG-SM position to allow 8051 programming or other access to the other EEPROMs. The source firmware for this EEPROM is located in the `<Installed_directory>\<Version>\Firmware\Connect` directory. The actual EEPROM image is stored in the *LP18_safe.iic* file.

2. C2 Load Small EEPROM:SW2 = LG-SM, SW1 = SMALL

   This setting selects the EEPROM located in socket U8. The I^{2}C EEPROM address pins for this socket are strapped to ‘000’. This socket only supports single-byte address EEPROMs. This EEPROM is pre-programmed at manufacturing with simple firmware which connects to USB and responds to descriptor requests with the Cypress VID (0x04B4) and the MoBL-USB Development Kit PID (0x0086). This VID/PID is associated with a driver which automatically downloads the Keil debug monitor to the development board. The source firmware for this EEPROM is located in the `<Installed_directory>\<Version>\Firmware\Connect` directory. The actual EEPROM image is stored in the *LP18_dvk.iic* file.

3. C2 Load Large EEPROM:SW2 = LG-SM, SW1 = LARGE

   This setting selects the EEPROM located in socket U8. The I^{2}C EEPROM address pins for this socket are strapped to ‘001’. This socket only supports double-byte address EEPROMs. This EEPROM is pre-programmed at manufacturing with the bulkloop example firmware (VID=0x04B4, PID=0x1004). The source firmware for this EEPROM is located in the following directory:

   `<Installed_directory>\<Version>\Firmware\bulkloop`.

   The actual EEPROM image is stored in the *bulkloop.iic* file. Note that if an EEPROM is connected to the SCL and SDA lines, but does not contain an 0xC2 formatted EEPROM, the device does not connect to USB. Therefore, it cannot be programmed. Always connect to the SAFE EEPROM if you are not using one of the programmed EEPROMs. See the MoBL-USB datasheet or MoBL-USB Technical Reference Manual for additional information on supported EEPROM formats.

3.5 Interface Connectors

Table 3-3. Logic Analyzer Pinout

Agilent 01650-63203 Pod Pins			
CLK1	3	4	D15
D14	5	6	D13
D12	7	8	D11
D10	9	10	D9
D8	11	12	D7
D6	13	14	D5
D4	15	16	D3
D2	17	18	D1
D0	19	20	GND

Six 20-pin headers P1-P6 on the MoBL-USB FX2LP18 Development Board have pins assigned to be compatible with HP (Agilent) logic analyzers, as shown in Table 3-3. The slight bulge in the middle rows of the table (pins 9 and 11) indicates the connector key. The six headers P1-P6 serve three purposes:

- They mate with the prototyping board supplied in the MoBL-USB Development Kit or the one supplied in the CY3687 MoBL-USB FX2LP18 Development Kit.
- They allow direct connection of HP (Agilent) logic analyzer pods (Agilent P/N 01650- 63203).
- They allow general purpose probing by other logic analyzers or oscilloscopes.

Table 3-3 shows the logic analyzer pod pin designations. The MoBL-USB signals on P1-P6 are arranged to fulfill the following requirements:

- High-speed MoBL-USB strobe signals (CLKOUT and IFCLK) are connected to pin 3 of each of the five connectors P1-P6, so that they may be used as the logic analyzer clock CLK1.
- CLK2 is not used. Instead, each connector brings 3.3 V power from the MoBL-USB Development Board up to the prototype board using pin 2.
- The signals are logically grouped. For example, the MoBL-USB FIFO data (which shares PORTB and PORTD pins) is on P1.

Because the 20-pin headers on the prototyping board are stackable, it is possible to build custom circuitry on the proto board, plug the board into the MoBL-USB development board, and still plug logic analyzer pods into the six connectors P1-P6.

Table 3-4 through Table 3-9 show the MoBL-USB pin designations for P1 through P6. For dual-mode pins, the power-on default signal names are shown in bold type, and the alternate pin names are shown in the outside columns.

Table 3-4. P1 Pin Designations

Alternate	Default	P1		Default	Alternate
	NC	**1**	**2**	3.3 V	
	VCCIO	**3**	**4**	PD7	FD[15]
FD[14]	PD6	**5**	**6**	PD5	FD[13]
FD[12]	PD4	**7**	**8**	PD3	FD[11]
FD[10]	PD2	**9**	**10**	PD1	FD[9]
FD[8]	PD0	**11**	**12**	PB7	FD[7]

Table 3-4. P1 Pin Designations *(continued)*

Alternate	Default	P1		Default	Alternate
FD[6]	PB6	**13**	**14**	PB5	FD[5]
FD[4]	PB4	**15**	**16**	PB3	FD[3]
FD[2]	PB2	**17**	**18**	PB1	FD[1]
FD[0]	PB0	**19**	**20**	GND	

Table 3-5. P2 Pin Designations

Alternate	Default	P2		Default	Alternate
	NC	**1**	**2**	3.3 V	
	NC	**3**	**4**	RDY1	SLWR
SLRD	RDY0	**5**	**6**	N.C.	
	NC	**7**	**8**	N.C.	
FLAGC	CTL2	**9**	**10**	CTL1	FLAGB
FLAGA	CTL0	**11**	**12**	PA7	FLAGD
PKTEND	PA6	**13**	**14**	PA5	FIFOADR1
FIFOADR0	PA4	**15**	**16**	PA3	WU2
SLOE	PA2	**17**	**18**	PA1	INT1#
INT0#	PA0	**19**	**20**	GND	

Table 3-6. P3 Pin Designations

Alternate	Default	P3		Default	Alternate
	NC	**1**	**2**	3.3 V	
	VCCIO	**3**	**4**	N.C.	
	NC	**5**	**6**	N.C.	
	NC	**7**	**8**	N.C.	
	RESET#	**9**	**10**	N.C.	
	NC	**11**	**12**	N.C.	
	NC	**13**	**14**	N.C.	
	NC	**15**	**16**	N.C.	
	NC	**17**	**18**	N.C.	
	NC	**19**	**20**	GND	

Table 3-7. P4 Pin Designations

Alternate	Default	P4		Default	Alternate
	N.C.	**1**	**2**	3.3 V	
	CLKOUT	**3**	**4**	GND	
	N.C.	**5**	**6**	NC	
	5 V	**7**	**8**	5 V	
	NC	**9**	**10**	NC	
	NC	**11**	**12**	NC	

Table 3-7. P4 Pin Designations *(continued)*

Alternate	Default	P4		Default	Alternate
	NC	**13**	**14**	NC	
	NC	**15**	**16**	NC	
	NC	**17**	**18**	NC	
	NC	**19**	**20**	GND	

Table 3-8. P5 Pin Designations

Alternate	Default	P5		Default	Alternate
	NC	**1**	**2**	3.3 V	
	IFCLK	**3**	**4**	NC	
	NC	**5**	**6**	NC	
	NC	**7**	**8**	NC	
	NC	**9**	**10**	NC	
	NC	**11**	**12**	NC	
	NC	**13**	**14**	NC	
	NC	**15**	**16**	NC	
	1.8 V	**17**	**18**	1.8 V	
	1.8 V	**19**	**20**	GND	

Table 3-9. P6 Pin Designations

Alternate	Default	P6		Default	Alternate
	NC	**1**	**2**	3.3 V	
	VCCIO	**3**	**4**	NC	
	NC	**5**	**6**	NC	
	NC	**7**	**8**	NC	
	WAKEUP#	**9**	**10**	SDA	
	SCL	**11**	**12**	NC	
	NC	**13**	**14**	NC	
	NC	**15**	**16**	NC	
	NC	**17**	**18**	NC	
	NC	**19**	**20**	GND	

3.6 ATA Connector P7

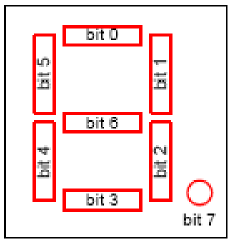
Table 3-10 shows the pinout for P7, a 40-pin connector that interfaces with a standard ATA cable. **Note** This is for ATA use only. SP1, 2, 3, and 4 should be bridged with solder to connect the appropriate pull-up or pull-down resistors required for ATA. An 80-pin cable is required for UDMA transfer modes and recommended for all transfer modes.

Table 3-10. ATA Connector Pinout

P7 (ATA)					
RESET#	PA7	1	2	GND	GND
DD7	PB7	3	4	PD0	DD8
DD6	PB6	5	6	PD1	DD9
DD5	PB5	7	8	PD2	DD10
DD4	PB4	9	10	PD3	DD11
DD3	PB3	11	12	PD4	DD12
DD2	PB2	13	14	PD5	DD13
DD1	PB1	15	16	PD6	DD14
DD0	PB0	17	18	PD7	DD15
GND	GND	19	20	N.C.	KEYPIN
DMARQ	RDY1	21	22	GND	GND
DIOW#	CTL0	23	24	GND	GND
DIOR#	CTL1	25	26	GND	GND
IORDY	RDY0	27	28	GND	CSEL
DMACK#	CTL2	29	30	GND	GND
INTRQ	PA0	31	32	N.C.	RESERVED
DA1	PA2	33	34	N.C.	PDIAG#
DA0	PA1	35	36	PA3	DA2
CS0#	PA4	37	38	PA5	CS1#
DASP#	10K Pull-up	39	40	GND	GND

3.7 I^{2}C Expanders

U2 and U4 are Philips PCF8574 I/O expanders. They connect to the I^{2}C bus SCL and SDA pins, and provide eight general-purpose input-output pins. U4 provides eight output bits, connected to the seven-segment readout U1. U2 provides eight input bits, four of which connect to push buttons S1-S4, and four of which are available for your use. U4 connects to the 7-segment readout (U1) using the following bit assignments.



U4 has the group address 0100 and is strapped to unit address 001. Therefore to write a value to the 7-segment readout, 8051 firmware sends a control byte of 01000010 (the LSB indicates a write operation), followed by the data byte.

U2 uses its I/O pins as inputs connected to S1-S4 according to the following table:

Bit	Switch
0	S1
1	S2
2	S3
3	S4

U2 has the group address 0100, and is strapped to unit address 000. Therefore to read the switch values, 8051 firmware sends a control byte of 01000001 (the LSB indicates a read operation), and then reads the data byte.

3.8 LED Indicators

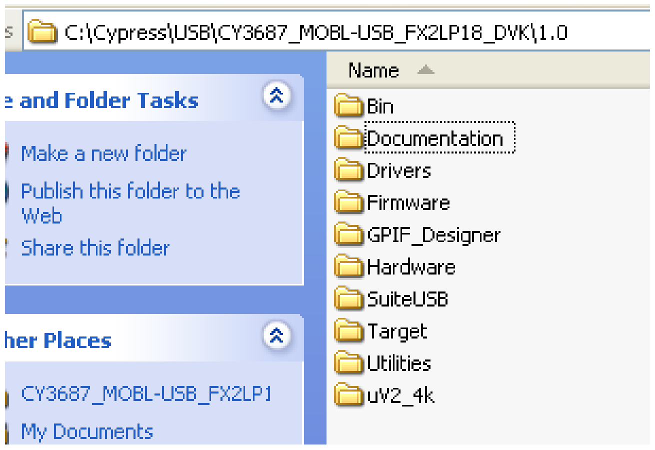
LEDs D1, D2, and D4 indicate available power on the three power rails (3.3 V, 2.5 V, and 1.8 V).

LED D3 indicates drive activity on the ATA connector. LED D5 indicates that the debugger is active.

4. Development Kit Contents

This section provides a detailed description of the structure and content of the MoBL-USB FX2LP18 as it exists on a user PC after installation. The DVK installer installs several development board related files in the Windows directory tree as shown in Figure 4-1. The default directory of CY3687 kit is C:\Cypress\USB\CY3687_MoBL-USB_FX2LP18_DVK. In further sections the default installation directory is referred to as <Installed_directory>.

Figure 4-1. CY3687 Development Kit Content Structure



Bin
Documentation
Drivers
Firmware
GPIF_Designer
Hardware
SuiteUSB
Target
Utilities
uV2_4k

4.1 Bin

This folder contains the following utilities

- **Cyscript.exe:** This utility is used to generate script files for equivalent Firmware(.hex) files
- **Hex2bix.exe:** This utility is used to convert a Firmware image compatible to RAM memory (.hex) to a EEPROM image (.iic).
- **Setenv.bat:** Batch file to set path variables for Keil compiler and Firmware examples. Click on this batch file to set the environment variables necessary before compiling the firmware examples of the kit.

4.2 Documentation

This directory contains documentation, which explains in detail about the CY3687 MoBL-USB DVK. Following is the summary of the documents in CY3687 MoBL-USB development kit as shown in Table 4-1.

Table 4-1. Documents Summary for MOBL-USB FX2LP18 Development Kit

S.No	Document	Description
1	CY3687 MoBL-USB™ FX2LP18 Quick Start Guide.pdf	The document assists user to quickly install USB drivers for MoBL-USB FX2LP18 development kit board. It also explains in detail about several components on board
2	CY3687 MoBL-USB FX2LP18 Development Kit Guide.pdf	The document provides complete details related to Kit operation. It also discusses in detail on PC applications usage, Component details on development board hardware, Signed USB drivers, etc
3	Release Notes CY3687 MoBL-USB™ FX2LP18 Development Kit.pdf	This document contains system requirement to install and to perform hands-on with CY3687 development kit and the list of documents provided with the kit.It also lists the errata items if applicable.
4	MoBL-USB™ FX2LP18 Technical Reference Manual.pdf	This document explains in detail about each IP module inside MoBL-USB FX2LP18 IC. It also lists the complete set of register definitions for the IC.
5	MoBL-USB™_FX2LP18_Datasheet.pdf	This document explains the finer details of MoBL-USB FX2LP18 IC.
6	Differences between EZ-USB FX2LP and MoBL-USB™ FX2LP18.pdf	The document explains some of pin-to-pin differences between EZ-USB FX2LP and MoBL-USB™ FX2LP18 IC's.
7	EZ-USB® FX2LP and MoBL-USB™ FX2LP18 56-Ball BGA PCB Layout Guidelines.pdf	The documents serve as a PCB design guideline for 56 pin BGA packages of EZ-USB® FX2LP and MoBL-USB™ FX2LP18

4.3 Drivers

This directory contains Microsoft certified Signed Cypress USB drivers for different Windows OS platforms like Window 2000(32-bit) and Windows XP,Vista and 7 in 32 and 64-bit OS platforms. Following is the detailed list of drivers as shown in Table 4-2.

Table 4-2. USB Drivers in MoBL-USB FX2LP18 Development Kits

S.No	USB Driver	Description
1	cyusbfx2lp18	This directory contains generic driver *cyusb.sys*. Driver information file *cyusbfx2lp18.inf* and Microsoft caltalog file(*cyusbfx2lp18.cat*) files required to enumerate MoBL-USB device.The .INF file contain default Connect mode VID/PID and firmware examples VID/PID. More details about this driver is mentioned in Chapter-6
2	CyMonfx2lp18	This directory contains generic driver *cyusb.sys* ,driver information file - *CyMonfx2lp18.inf* and Microsoft caltalog file(CyMonfx1_fx2lp.cat) files required to debug MOBL-USB firmware examples. The .INF file contains VID/PID to automatically download Keil debug monitor script file(*mon_i2c.spt*) to assist user in step-by-step debugging of firmware examples. More details about this driver is mentioned in Chapter-8

4.4 Firmware

The CY3687 MoBL-USB development kit contains several firmware examples to validate different interfaces of MoBL-USB FX2LP18 device. Following is the list of firmware examples.

Table 4-3. List of Firmware Example in MoBL-USB FX2LP18 Development Kits

S.No	Firmware Example	Description
1	hid_kb	Example firmware that emulates a HID-class keyboard using the buttons and 7-segment display on the DVK board
2	Bulkloop	Contains a bulk loopback test that exercises the MOBL-USB FX2LP18 bulk end-points. It loops back EP2OUT to EP6IN and EP4OUT to EP8IN
3	Bulkext	Contains a bulk loopback test that exercises the MOBL-USB FX2LP18 bulk end-points. The loopback is performed using the external auto pointer. Data is copied from the OUT endpoint buffer to external RAM and then to the IN endpoint buffer. It loops back EP2OUT to EP6IN and EP4OUT to EP8IN
4	Bulksrc	Contains bulk endpoint endless source/sink firmware. It can be driven using the CyConsole or CyBulk. EP2OUT always accepts a bulk OUT; EP4OUT always accept a bulk OUT; EP6IN always returns a 512-byte packet, 64 bytes at full-speed. Based on buffer availability in EP8IN, the most recent packet of EP4OUT is written to EP8IN.
5	dev_io	Contains the source files to build simple development board I/O sample. This software demonstrates how to use the buttons and LED on the MoBL-USB FX2LP18 development kit.
6	EP_Interrupts	Bulk loopback firmware that demonstrates use of endpoint interrupts using MoBL-USB FX2LP18.
7	extr_intr	Firmware that demonstrates external interrupt handling- INT0, INT1, INT4, INT5 and INT6.
8	Ibn	Contains firmware to perform bulk loopback of EP2OUT to EP6IN and EP4OUT to EP8IN using the IBN (In Bulk Nak) interrupt to initiate the transfer.
9	Pingnak	Contains firmware to perform bulk loopback of EP2OUT to EP6IN and EP4OUT to EP8IN using the PING NAK interrupt to initiate the transfer
10	iMemtest	The firmware example tests on-chip RAM
11	vend_ax	Contains the source files to build a vendor specific command sample. This example demonstrates how to implement different vendor commands

4.5 GPIF_Designer

This directory contains GPIF designer software, which allows user to create to State machine waveforms.These waveforms are useful to communicate with external devices like SRAM, FPGA..etc using a MoBL-USB FX2LP18 GPIF interface.

4.6 Hardware

The directory contains MoBL-USB hardware schematic, PCB, Layout, gerber and PCB BOM files. Following are the files in this directory.

Table 4-4. List of Hardware Files in MoBL-USB FX2LP18 Development Kits

S.No:	Files	Description
1	CY3687_MoBL-USB_FX2LP18_DVK_schematic.pdf/ CY3687_MoBL-USB_FX2LP18_DVK_Schematic.DSN	This file is Non-editable (pdf) and editable (.DSN) version of the schematic source file. The .DSN file can be viewed using Orcad software.
2	CY3687_MoBL-USB_FX2LP18_DVK_Gerber.zip	This .zip files contains PCB images of different layers of the MoBL-USB FX2LP18 development board PCB.
3	CY3687_MoBL-USB_FX2LP18_DVK_BOM.pdf/ CY3687_MoBL-USB_FX2LP18_DVK_BOM.xls	This file contains components like resistors, capacitors, jumpers, etc., used in designing MoBL-USB FX2LP18 development board
4	CY3687_MoBL-USB_FX2LP18_DVK_Board_Layout.brd /CY3687_MoBL-USB_FX2LP18_DVK_Board_Layout.pdf	These files contain the PCB layout of MoBL-USB FX2LP18 development board.The editable layout file (.brd) can be viewed using Allegro software tools.
5	PDC-9022-A-Dimension.PDF, PDC-9022-REVA.pdf, CY3681-2_ASSEMBLY.pdf, and PDC-9022-A.zip	These files are part of PROTO board daughter card designed to provide a sample prototype area for validating communication between GPIF/Slave FIFO interface and External device.

4.7 SuiteUSB

This folder contains SuiteUSB 3.4.7 installer package and a sample Cypress Software License agreement document. The software is installed as part ofCY3687 MoBL-USB FX2LP18 Kit installer and the contents are by default located at C:\Cypress\Cypress SuiteUSB 3.4.7.The package contains C++ and C# .NET application tools to communicate with MoBL-USB FX2LP18 device. Also it contains Cypress generic USB drivers (3.4.7). The package contains unsigned drivers. The signed driver packages supplied in the /Driver directory should be used to try the code examples explained in MoBL-USB Development Kit Firmware Examples chapter on page 59.

4.8 Target

This directory contains MoBL-USB FX2LP18 register definition header files, Keil debug monitor, and so on. Following are the list of files.

Table 4-5. List of Files in Target Directory

S.No	Sub-directory	File	Description
1	FW/Lp	Fw.c, periph.c ,dscr.a51,fw.uv2	This directory contains basic framework project source files used to develop the firmware examples in the MOBL-USB FX2LP18 development kit
2	Monitor	mon-int-sio1-c0.hex, mon-int-sio1-C0.spt, etc	This directory contains Keil debug monitor .hex and script files that reside in MoBL-USB FX2LP18 internal RAM. These files are used to debug firmware examples through UART port SIO-1 at 9600 baud rate.
3	Inc	fx2regs.h, lpregs.h lpregs.inc, fx2regs.inc fx2.h, lp.h syncdly.h, fx2sdly.h	These files contain MoBL-USB FX2LP18 device register definitions and basic structure definitions. Also several delay routines of fixed duration (syncdly.h/fx2sdly.h)are defined to be used in frameworks code.The MoBL-USB FX2LP18 and EZ-USB FX2LP register definitions are mostly identical.
4	Lib/Lp	EZUSB.Lib USB-JmpTb. OBJ	This folder contains I2C read/Write routines Library(EZUSB.lib) and Interrupt vector definitions for MoBL-USB FX2LP18 device (USBJmpTb.OBJ). MoBL-USB FX2LP18 and EZ-USB FX2LP I2C block and interrupt vector definitions are identical.
5	File_Transfer	64_Count.hex, 512_Count.hex, 1024_Count.hex, 1024_1248.hex, 2048_1248.hex, 4096_1248.hex and 8192_Count.hex	The files can be used to send fixed set of data over MoBL-USB FX2LP18 endpoints using File Transfer button in EZ-USB Interface Window of Cyconsole.

4.9 Utilities

This directory contains hex2bix utility source code in VC++6 environment. The project code can be used as reference to invoke different command line options supported by this utility.

4.10 uV2_4k

This directory contains Keil uVision2 Trial version IDE. The IDE has the limitation of compiling code limit of 4K. All the code examples included with the MoBL-USB FX2LP18 development kit can be compiled using this IDE.

5. MoBL-USB FX2LP18 Firmware Frameworks

The firmware frameworks simplify and accelerate USB peripheral development using the MoBL-USB chip. The MoBL-USB firmware framework library is similar to EZ-USB FX1/FX2LP device framework. The term EZ-USB framework referenced throughout this chapter also means MoBL-USB FX2LP18 device framework. The frameworks implement 8051 code for EZ-USB chip initialization, USB standard device request handling, and USB suspend power management services. The user provides a USB descriptor table and code to implement the peripheral function to complete a fully compliant USB device. The frameworks provide function hooks and example code to help with this process. The frameworks use the EZ-USB library to carry out common functions and for MoBL-USB FX2LP18 register definitions. Most of the firmware examples in the MoBL-USB FX2LP18 DVK are based on the frameworks.

5.1 Frameworks Overview

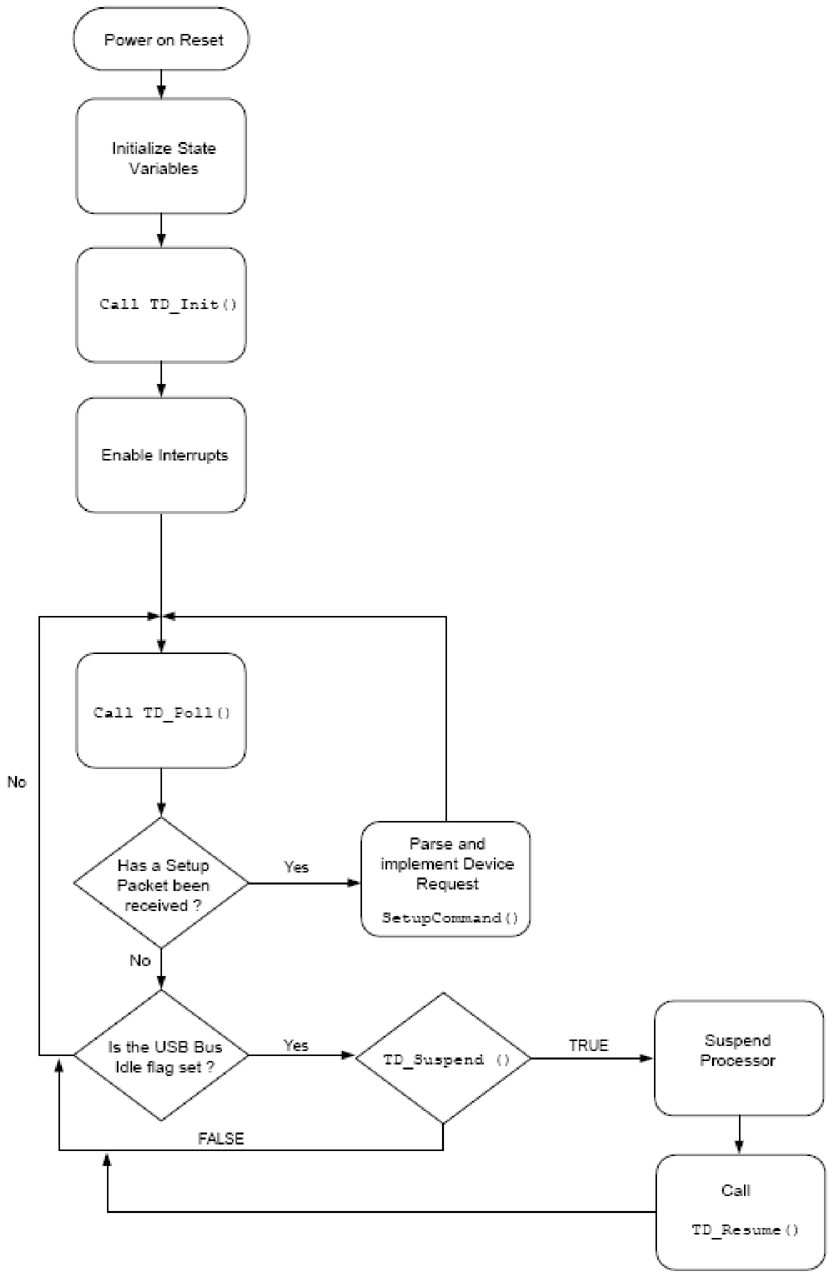
The frameworks implement the basic functionality required of a USB compliant peripheral device. By linking a minimal descriptor table, it is possible to build a fully compliant Device Framework (Chapter-9 of USB 2.0 specification at http://www.usb.org/developers/docs/).

At startup, the frameworks initialize all its internal state variables. It then calls the user initialization function TD_Init(). Upon return, the frameworks initialize the USB interface to the unconfigured state and enable interrupts. The firmware then Re-enumerates and starts the co-operative task dispatcher. The task dispatcher repeatedly performs the following tasks in the given order.

1. Calls user function TD_Poll().
2. Determines if a standard device request is pending. If so, it parses the received command and responds accordingly. The frameworks automatically handle the standard USB requests, but allow the user to override the default behavior for all requests.
3. Determines if the USB core has reported a USB suspend event. If so, it calls the user function TD_Suspend().

MoBL-USB FX2LP18 device interrupts are handled by the frameworks. It provides hooks for user code notification of USB events.

Figure 5-1. Firmware Frameworks Flow Chart



5.2 Building FrameWorks

The frameworks is written using the Keil uVision2 Compiler. It has been tested only with these tools. The source uses several Keil C extensions, so compatibility with other compilers is not guaranteed.

For your custom device firmware, you can either start with one of the firmware examples or start with the "clean" frameworks code. This code is located in the `<Installed_directory>\<version>\Target\fw` directory. The sub-directory is chip dependent. The firmware is located in the "LP" sub-directory. Before editing the firmware, create a new directory for your project and copy the various frameworks source files into it.

After starting the Windows Command Prompt, run *setenv.bat* (located in the **Bin** directory) to set up the build environment. This batch file assumes that you have installed the DVK and Keil tools in the default directories.

The following table lists and describes the main files in the frameworks:

Table 5-1. Files in Firmware Frameworks

File Name	Description
FW.C	This is the main frameworks source file. It contains main(), the task dispatcher, and the SETUP command handler. For most firmware projects, there is no need to modify this file
PERIPH.C	This source file contains initialization and task dispatch functions that are called from fw.c. This is where you customize the frameworks for your specific device. This file also contains stub interrupt service routine (ISRs) functions for all of the USB (INT2) and GPIF (INT4) interrupts
DSCR.A51	Assembly file that contains your device's custom descriptors
FX2.H/LP.H	Head file containing common EZ-USB constants, macros, data types, and library function prototypes
FX2REGS.H/ LPREGS.H	MoBL-USB FX2LP18 register declarations and bit mask constants. The EZ-USB FX2LP and MoBL-USB FX2LP18 register definitions are mostly identical.
SYNCDLY.H/ FX2SDLY.H	Contains the synchronization delay macro.
EZUSB.LIB	EZ-USB Library object code. See EZ-USB Library on page 39 for more details
USBJMPTB.OBJ	Object code that contains the ISR jump table for USB and GPIF interrupts
BUILD.BAT	Batch file for compiling/linking the firmware using the Keil command line tools
FW.UV2	Keil uVision2 project file for compiling/linking the firmware

5.3 Function Hooks

The frameworks provides function hooks to simplify the addition of user code. The functions are divided into three categories: those called by the task dispatcher, the standard device request parser, and the USB interrupt handler. The following sections contain a complete list of functions and their descriptions.

5.3.1 Task Dispatcher Functions

The following functions are called by the task dispatcher located in main().

5.3.1.1 TD_Init()

```
void TD_Init()
```

This function is called once during the initialization of the frameworks. It is called before ReNumeration and the Task Dispatcher starts. It is intended for global state variable and device initialization.

5.3.1.2 TD_Poll()

```
void TD_Poll()
```

This function is called repeatedly during device operation. It should contain a state machine that implements the user's peripheral function. High priority tasks can be completed before returning from this function. However, failure to return from this function prevents frameworks from responding to device requests and USB suspend events. If a large amount of processing time is required, it must be split up to execute in multiple calls to TD_Poll().

5.3.1.3 TD_Suspend()

```
BOOL TD_Suspend()
```

This function is called before the frameworks enter suspend mode. This function contains code that places the device in a low power state and returns TRUE. However, the user code can prevent the frameworks from entering suspend mode by returning FALSE.

5.3.1.4 TD_Resume()

```
void TD_Resume()
```

This function is called after the frameworks has resumed the processor in response to an external resume event. At this point, the device resumes full power operation.

5.3.2 Device Request Functions

These are helper functions that the device request handler (SetupCommand() in FW.C) calls. These are mainly used to override or augment the default device request handler.

5.3.2.1 DR_GetDescriptor()

```
BOOL DR_GetDescriptor()
```

This function is called before the frameworks decode and implement the GetDescriptor device request. The register array SETUPDAT contains the current eight byte setup command. It can be parsed by the user's code to determine which Get Descriptor command is issued. If TRUE is returned, the frameworks will parse and implement the command. If FALSE is returned, it will do nothing.

5.3.2.2 *DR_GetInterface()*

```
BOOL DR_GetInterface()
```

This function is called before the frameworks implement the Get Interface device request. The register array SETUPDAT contains the current eight byte setup command. If TRUE is returned, the frameworks will implement the command. If FALSE is returned, it will do nothing.

5.3.2.3 *DR_SetInterface()*

```
BOOL DR_SetInterface()
```

This function is called before the frameworks implement the Set Interface device request. The register array SETUPDAT contains the current eight byte setup command. It is the responsibility of this routine to save the new interface setting and to do any necessary device configuration. If TRUE is returned, the frameworks will implement the command. If FALSE is returned, it will do nothing.

5.3.2.4 *DR_GetConfiguration()*

```
BOOL DR_GetConfiguration()
```

This function is called before the frameworks implement the Get Configuration device request. The register array SETUPDAT contains the current eight byte setup command. If TRUE is returned, the frameworks will implement the command. If FALSE is returned, it will do nothing.

5.3.2.5 *DR_SetConfiguration()*

```
BOOL DR_SetConfiguration()
```

This function is called before the frameworks implement the Set Configuration device request. The register array SETUPDAT contains the current eight byte setup command. By default, the frameworks parses the descriptor table to determine the new configuration interface and its endpoints. It then configures the MoBL-USB FX2LP18 control registers to reflect these new endpoints. If the configuration is set to 0 then the frameworks will invalidate all of the endpoints. If TRUE is returned, the frameworks will implement the command. If FALSE is returned, it will do nothing.

5.3.2.6 *DR_GetStatus()*

```
BOOL DR_GetStatus()
```

This function is called before the frameworks implement the Get Status device request. The register array SETUPDAT contains the current eight byte setup command. If TRUE is returned, the frameworks will implement the command. If FALSE is returned, it will do nothing.

5.3.2.7 *DR_ClearFeature()*

```
BOOL DR_ClearFeature()
```

This function is called before the frameworks implement the Clear Feature device request. The register array SETUPDAT contains the current eight byte setup command. If TRUE is returned, the frameworks will implement the command. If FALSE is returned, it will do nothing.

5.3.2.8 *DR_SetFeature()*

```
BOOL DR_SetFeature()
```

This function is called before the frameworks implement the Set Feature device request. The register array SETUPDAT contains the current eight byte setup command. If TRUE is returned, the frameworks will implement the command. If FALSE is returned, it will do nothing.

5.3.2.9 *DR_VendorCmnd()*

```
void DR_VendorCmnd()
```

This function is called when the frameworks determine a vendor specific command has been issued. The register array SETUPDAT contains the current eight byte setup command. This function has no return value. The frameworks does not implement any vendor specific commands. However, the EZ-USB serial interface engine (SIE) uses vendor specific command 0xA0 to implement software uploads and downloads. Therefore, command 0xA0 will not be passed to the user's code.

5.3.3 ISR Functions

There are over 40 different USB and GPIF auto-vectored interrupts available. PERIPH.C contains stub ISR functions for all of these interrupts. This section documents the ISRs that require special handling by device firmware. For more information, refer to the Interrupts chapter in the MoBL-USB Technical Reference Manual.

5.3.3.1 *ISR_Sudav()*

```
void ISR_Sudav(void) interrupt 0
```

This function is called on receiving the Setup Data Available interrupt. This function needs to set GotSUD to TRUE so that the device request handler can process the SETUP command.

5.3.3.2 *ISR_Sof()*

```
void ISR_Sof(void) interrupt 0
```

This function is called on receiving the Start of Frame interrupt. It gets called every 1 ms at full-speed and every 125 uS at high-speed. The only action for this interrupt in the default frameworks code is to clear the interrupt.

5.3.3.3 *ISR_Ures()*

```
void ISR_Ures(void) interrupt 0
```

This function is called on receiving the USB Reset interrupt. In your custom code, place any housekeeping that must be done in response to a USB bus reset in this routine. The default frameworks code updates the configuration descriptor pointers in response to this interrupt. When a USB Reset occurs, the device is always operating in full-speed (until high-speed chirp completes). Therefore, it must return its full-speed configuration descriptor in response to a get configuration descriptor request and must return its high-speed configuration descriptor in response to a get other-speed descriptor request.

5.3.3.4 *ISR_Susp()*

```
void ISR_Susp(void) interrupt 0
```

This function is called on receiving the USB Suspend interrupt. The default frameworks code sets the global variable Sleep to TRUE in this routine. This is required for the Task Dispatcher to detect and handle the suspend event.

5.3.3.5 *ISR_Highspeed()*

```
void ISR_Highspeed(void) interrupt 0
```

This function is called on receiving the USB HISPEED interrupt. In your custom code, place any housekeeping that must be done in response to a transition to high-speed mode in this routine.

The default frameworks code updates the configuration descriptor pointers in response to this interrupt. When the device switches to high-speed mode, it must return its high-speed configuration descriptor in response to a get configuration descriptor request and must return its full-speed configuration descriptor in response to a get other-speed descriptor request.

5.4 EZ-USB Library

The EZ-USB library is an 8051 .LIB file that implements functions that are common to many firmware projects. These functions need not be modified and are therefore provided in library form. However, the kit includes the source code for the library in the event that you need to modify a function or if you just want to know how something is done.

In addition to providing common functions, the library also creates register definitions for all EZ-USB registers. The source code and the compiled .LIB file are located in the `<Installed_directory>\<Version>\Target\Lib\lp` directory.

5.4.1 Building the Library

Only the full retail version of the Keil tools can build library files. The evaluation version will not build this library. After starting the Windows Command Prompt, run setenv.bat (located in the `<Installed_directory>\<Version>\Bin` directory) to set up the build environment. This batch file assumes that you have installed the DVK and Keil tools in the default directories. To build the library, run the build.bat file from the command prompt.

Build.bat also assembles the file usbjmptb.a51 to create usbjmptb.obj. This file contains the jump table for the USB (INT2) and GPIF (INT4) autovectored interrupts. See the MoBL-USB Technical Reference Manual (TRM) in the Kit documentation for more information on autovector interrupts.

5.4.2 Library Functions

5.4.2.1 EZUSB_Delay()

```
void EZUSB_Delay(WORD ms)
```

This function performs a busy wait for a given number of milliseconds. The parameter ms determines the length of the busy wait. Upon completion of the delay the function returns.

5.4.2.2 EZUSB_Discon()

```
void EZUSB_Discon(BOOL renum)
```

This function performs a USB disconnect/reconnect. It disconnects the device, delays for 1500ms, clears any pending USB interrupts (INT2), reconnects, and then returns. The parameter renum determines if the MoBL-USB FX2LP18 renumerate bit is set in the USB control register. If renum is TRUE, the renumerate bit is set and following a return from this function the 8051 will be responsible for handling all USB device requests on endpoint 0. If renum is FALSE, the renumerate bit is not modified. If the renumerate bit is clear then the MoBL-USB FX2LP18 serial interface engine handles most of the USB device requests on endpoint 0.

5.4.2.3 EZUSB_GetStringDscr()

```
STRINGDSCR xdata * EZUSB_GetStringDscr(BYTE StrIdx)
```

This function returns a pointer to instance StrIdx of a string descriptor in the descriptor table. The instance is determined by the StrIdx parameter. If the descriptor table does not contain the given number of instances then the function returns a NULL pointer.

5.4.2.4 EZUSB_Susp()

```
void EZUSB_Susp(void)
```

This function suspends the processor in response to a USB suspend event. This function will not return until the suspend is cleared by a USB bus resume or a wake-up event on the EZUSB wake-up pin. If a suspend event is not pending, this function will return immediately.

5.4.2.5 *EZUSB_Resume()*

```
void EZUSB_Resume(void)
```

This function generates the K-state on the USB bus required for a USB device remote wake-up. This function should be called following a USB suspend. It automatically determines if the wake-up is result of a USB resume or a remote wake-up and generates the K-state accordingly.

5.4.2.6 *I^{2}C Routines*

```
void EZUSB_InitI2C(void);
BOOL EZUSB_WriteI2C_(BYTE addr, BYTE length, BYTE xdata *dat);
BOOL EZUSB_ReadI2C_(BYTE addr, BYTE length, BYTE xdata *dat);
BOOL EZUSB_WriteI2C(BYTE addr, BYTE length, BYTE xdata *dat);
BOOL EZUSB_ReadI2C(BYTE addr, BYTE length, BYTE xdata *dat);
void EZUSB_WaitForEEPROMWrite(BYTE addr);
```

These functions automate access to I^{2}C devices such as the EEPROM, 7-segment display and buttons on the DVK board. See the vend_ax and dev_io firmware examples for details on using these functions.

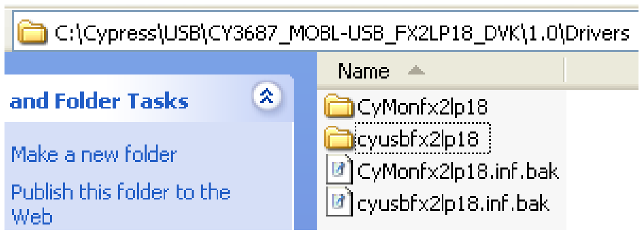
6. Cypress USB Drivers for MoBL-USB FX2LP18 Kit

This chapter explains in brief about the Cypress USB drivers provided along with the kits. The USB Signed driver package consists of several files to test different features of MoBL-USB FX2LP18 kits. Additionally SuiteUSB installer is provided which supports a collection of USB Host application tools designed in C++ and C# .NET framework. These tools are useful to communicate with any cypress USB 2.0 device.

6.1 USB Signed Driver Package for MoBL-USB FX2LP18 Devices

The MoBL-USB FX2LP18 kits contain Microsoft Certified Signed driver packages for different purposes. Following is the snapshot of the Drivers directory of the MoBL-USB FX2LP18 kit.

Figure 6-1. Driver Packages in MoBL-USB FX2LP18 Kits

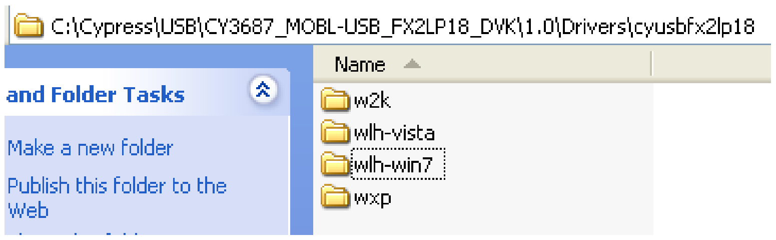


C:\Cypress\USB\CY3687_MOBL-USB_FX2LP18_DVK\1.0\Drivers

CyMonfx2lp18
cyusbfx2lp18
CyMonfx2lp18.inf.bak
cyusbfx2lp18.inf.bak

Note: All these drivers support Windows 2000 (32-bit) and Windows XP, Vista and 7 in both 32-bit as well as 64-bit configurations. The following Figure 6-2 summarizes the list of folders in CyLoad drivers specific to each OS.

Figure 6-2. Driver Sub-directory Summary

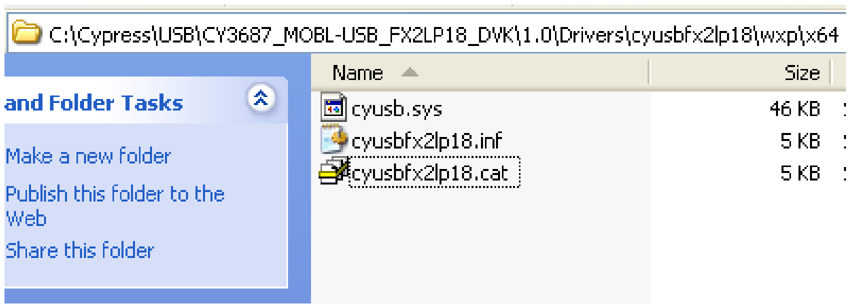


C:\Cypress\USB\CY3687_MOBL-USB_FX2LP18_DVK\1.0\Drivers\cyusbfx2lp18

w2k
wlh-vista
wlh-win7
wxp

These folders contain Signed driver files for relevant Windows OS platform. Following Figure 6-3 shows a sample snapshot of one of the Signed driver package-cyusbfx2lp18 sub-directory for Windows 64-bit OS platform.

Figure 6-3. Sample Signed Driver Package


C:\Cypress\USB\CY3687_MOBL-USB_FX2LP18_DVK\1.0\Drivers\cyusbfx2lp18\wxp\x64

Name	Size
cyusb.sys	46 KB
cyusbfx2lp18.inf	5 KB
cyusbfx2lp18.cat	5 KB

The basic set of files in a signed package is

- Cyxxx.cat - These are Windows Catalog files which contain Digital Signature. This file indicates that this driver-cyusb.sys passed Microsoft driver Certification process (WHQL) process.
- Cyusb.sys - This is the cypress provided generic USB driver for all USB 2.0 products.The driver version 3.4.5 was used in the driver certification process.
- Cyxx.INF - The file contains information about the .cat file and .sys file entries. The corresponding destination directories for driver files or scripts are also defined to allow copying of these files to Windows system folders. The INF file automates the process of driver loading and registering the entry in device manager of Windows.

The .INF files provided in the package **should not be modified**. These are provided for end user to test different features of MoBL-USB FX2LP18 device. If the .INF file is edited the Microsoft Digital signature is no longer valid. Also the default VID/PID mentioned in these INF files should not be used for any other purpose except for testing the basic features of the device. Attempt to bind the modified .INF file along with the existing **.cat** and **.sys** file would result in a Warning window. The Signed driver loading conditions are stringent in Windows Vista and 7 64-bit configurations. To temporarily avoid driver loading problems with edited .INF file press F8 during re-boot of Windows machine and select **“Disable Driver signature Enforcement”**. The 64-bit OS will still provide warnings but allow the edited INF file loading. This setting will not propagate to next re-boot and needs to performed again during next re-boot.

Note: To completely avoid these warnings users should remove all the instances of default VID/PID in the .INF file and replace with their own VID/PID combination. The modified .INF along with cyusb.sys needs to be re-submitted to Microsoft for driver certification. For more details on WHQL re-submission process for modified Cypress USB driver package refer to Application note titled “AN52970 - Windows Hardware Quality Labs (WHQL) Signing Procedure for Customer Modified Cypress USB Driver Files”.Depending on the functionality of each of these .INF packages they are classified into two major categories:

- Drivers for firmware examples and default MoBL-USB FX2LP configuration
- Drivers for keil monitor automatic download using script files

6.2 Drivers for Firmware Examples and Default Connect Mode MoBL-USB FX2LP18 Configuration

The MOBL-USB FX2LP18 development kit uses several VID/PID for tasks such as default connect mode enumeration (0x04B4/0x8614), firmware example re-enumeration (0x04B4/0x1004), etc. In the list of signed driver packages cyusbfx2lp18.inf contains all the VID/PID relevant for these tasks. Following is the summary of VID/PID used in this .INF file.

Table 6-1. List of VID/PID used in MoBL-USB FX2LP18 Kits

S.No	VID/PID	Functional description
1	0x04B4/0x8614	MOBL-USB FX2LP18 Connect Mode. After Reset MoBL-USB FX2LP18 device uses LP18_dvk.iic image is U5 Safe EEPROM. The image enables DPTR register and enumerate with this VID/PID
2	0x04B4/0x1004	Firmware examples in MoBL-USB FX2LP18 kit use this combination except for hid_kb example that uses 0x04B4/1005. This example requires Windows HID Class driver and does not require Cypress USB driver
3	0x04B4/0x1003	This is used by Cystream firmware example available in SuiteUSB 3.4.7 supplied along with this it. Download *CYStream.hex* from C:\Cypress\Cypress Suite USB 3.4.7\Firmware\CyStreamer using Cyconsole or CyControlCenter utility and test it using Streamer applications available in SuiteUSB.The procedure to download a firmware .hex file is explained in Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 of this document.

6.2.1 Binding Cypress USB Driver to MoBL-USB Development Board

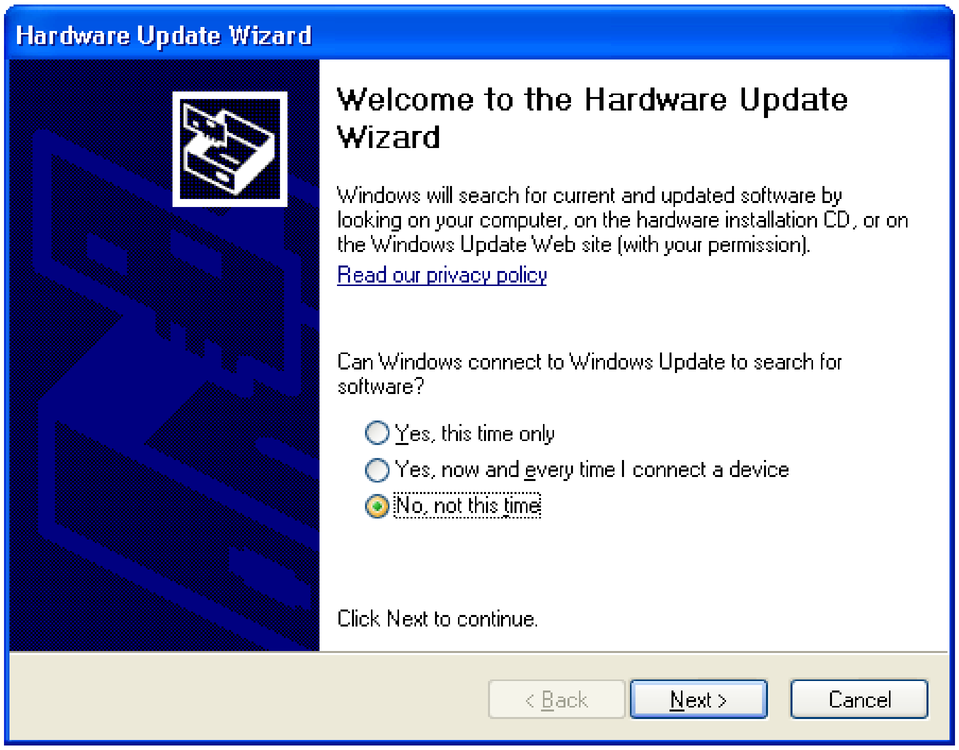
The MoBL-USB FX2LP18 development board supplied with the kit is used to bind Signed Cypress USB driver. Following are the steps to bind the driver.

1. Disconnect the USB A-to-B cable between J2 connector and PC USB Host port if connected previously.
2. Verify the following default jumper settings for the MoBL-USB FX2LP18 development board.

   FX2LP18: Short on JP1, JP2, JP4, JP5, JP6, JP7 (1-2), JP8, JP9 and JP10 (5-6), JP11 and JP12.

   The functionality of each of these jumpers is explained in Development Board chapter on page 17.
3. Verify if SW2 is switched to the side marked as **“SAFE” and SW1** can be either position.
4. Re-connect the USB A-to-B cable between J2 connector and PC USB Host port.
5. The MoBL-USB FX2LP18(0x04B4/0x8614) board enumerates with default Connect mode -U5 safe EEPROM VID/PID.
6. The Windows hardware wizard window pops up for user to update the corresponding driver path as shown in the following Figure 6-4.

Figure 6-4. Windows Hardware Wizard for Driver Update



Welcome to the Hardware Update Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

Read our privacy policy

Can Windows connect to Windows Update to search for software?

Yes, this time only

Yes, now and every time I connect a device

[✓ No, not this time]

Click Next to continue.

7. Select **No, not this time** and click **Next**. Select **Install from a specific location** and click **Next**.
8. In the subsequent window select Don't search. I will choose driver to install and select Next.In the list of Hardware devices select Universal Serial Bus Controllers and click on Have Disk button. A new Window pops up for locating USB driver. Click Browse button and point to respective directory of Windows PC Host PC to which the MoBL-USB FX2LP18 development board is connected. Following are the driver paths for different Windows OS platforms with respect to the path <Installed_directory>\<Version>\Drivers\cyusbfx2lp18
   a. **Windows2000:** w2k\x86
   b. **Windows XP (32-bit):** wxp\x86
   c. **Windows XP(64-bit):** wxp\x64
9. In Windows Vista and 7 OS platforms the Hardware wizard Window options will differ. If the hardware wizard window does not pop up then type **devmgmt.msc** directly in the vacant box in Windows Start. Locate the Unknown Device marked in Yellow. Right click on the Unknown Device and verify in the **Details >Hardware ID** if the VID/PID are matching with default connect mode VID/PID for MoBL-USB FX2LP18 device. Then right click again on the Unknown Device and select Update Driver Software. The Windows OS Hardware wizard window will now pop up. If the hardware wizard window pops up automatically then the entire process mentioned in step-9 can be avoided.
10. Select **Browse my computer for driver Software**. In the next window adjacent to **Browse for the driver software on your computer** click **Browse** button and select the following directory paths for Windows Vista and 7 OS
   a. **Windows-Vista(32-bit):** wlh-vista\x86
   b. **Windows-Vista(64-bit):** wlh-vista\x64
   c. **Windows-7(32-bit):** wlh-win7\x86

    d. **Windows-7(64-bit):** wlh-win7\x64

11. Open device manager as mentioned in step-9 and expand the list of USB Controllers.
12. Observe the MoBL-USB devices in the list of USB device controllers in device manager as mentioned in step-9. The MoBL-USB FX2LP18 device enumerates as **“Cypress MOBL-USB FX2LP18 Connect Mode(3.4.5.000)”**.

This completes the entire binding process for the MoBL-USB FX2LP18 device. The process is similar for any USB device with its own proprietary drivers.

6.3 Drivers for Keil Monitor Automatic Download using Script Files

The firmware examples provided with the MoBL-USB FX2LP18 kit can be manually downloaded using Cyconsole or CyControlcenter. If the user needs to automate the firmware downloading process then script files can be used i.e: Whenever the MoBL-USB board with relevant VID/PID w.r.t script file is detected then windows OS automatically downloads the actual firmware inside the script and the MoBL-USB device re-enumerates with new VID/PID defined in the firmware. The Keil debug monitor script file (mon_i2c.spt) does not enumerate after download but waits for the user to initiate debug session in keil uVision2 IDE.Following are the relevant driver files provided under \Drivers directory for keil monitor debugging

- **CyMonfx2lp18:** This directory contains keil debug monitor script file (mon_i2c.spt), Microsoft catalog file(cymon.cat), Driver Information file(CyMonfx2lp18.inf) and generic USB driver(cyusb.sys). The keil debug monitor after downloading to MoBL-USB FX2LP18 RAM memory will enable user to debug MoBL-USB FX2LP18 firmware examples using step-by-step debugging using UART port at 9600 baud rate. The method to debug firmware examples using this driver package is explained in detail in section Debugging Using Keil Monitor Program on page 90

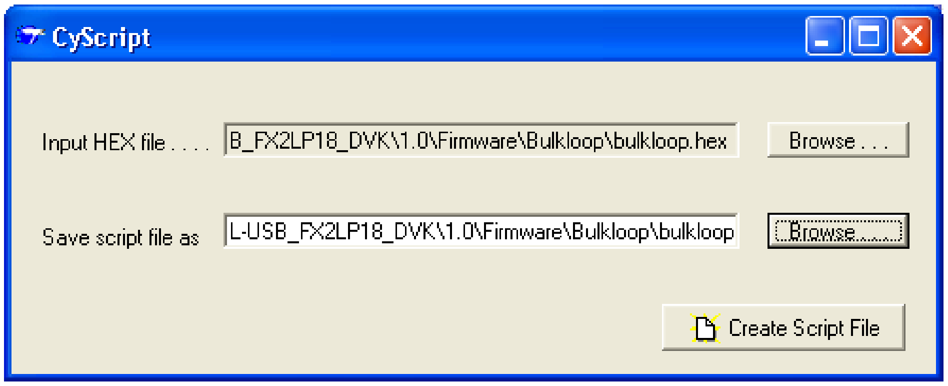
6.3.1 How to Generate and Play Script Files (.spt)

The script files are generated for a specific firmware. Select Bulkloop firmware example to experiment for this purpose. Choose Bulkloop.hex under <Installed_directory>\<version>\Firmware\Bulkloop in the MoBL-USB FX2LP18 kit contents. The script file can be generated using three tools available with MoBL-USB FX2LP18 development kit.

6.3.1.1 Script File Generation using Cyscript Tool

Open this tool located at <Installed_directory>\<Version>\Bin after installing MoBL-USB FX2LP18 kit contents. Click on Browse button adjacent to Input HEX file and select the path where the .hex file is located. Also choose the directory and filename of your choice for the script file and click **Create Script File** button as shown in Figure 6-5.

Figure 6-5. Script File Creation using CyScript Tool



Input HEX file . . . . B_FX2LP18_DVK\1.0\Firmware\Bulkloop\bulkloop.hex

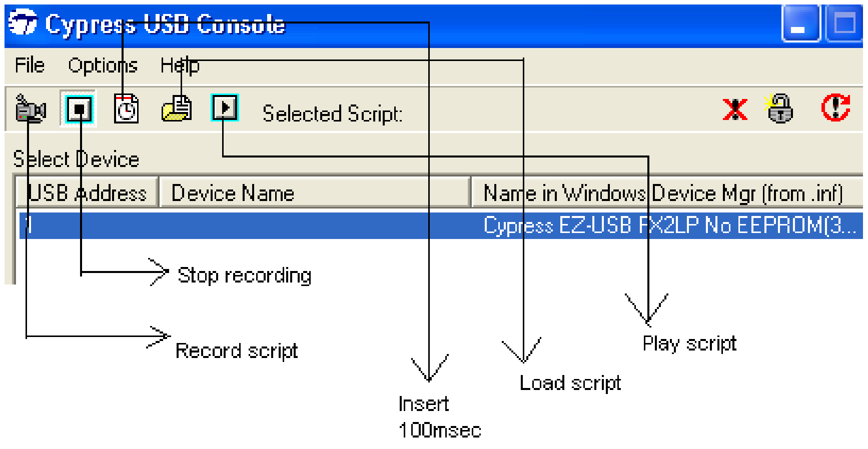
Save script file as L-USB_FX2LP18_DVK\1.0\Firmware\Bulkloop\bulkloop

Verify if the **xx.spt** file is located in the destination directory.This tool can only generate a script but cannot play the script. Using **Cyconsole** or **Controlcenter** the generated script file can be loaded and played to download the firmware inside the script.

6.3.1.2 *Script File Generation and Play using CyConsole*

Open Cyconsole tool located at `C:\Cypress\Cypress Suite USB 3.4.7\CyConsole`. Click on `C:\Cypress\Cypress Suite USB 3.4.7\CyConsole`. Alternately the tool can be accessed at Windows **Start->All programs->Cypress->Cypress SuiteUSB 3.4.7**. Follow the below process to generate and play the script.

Figure 6-6. Script Generation and Download Option in Cyconsole



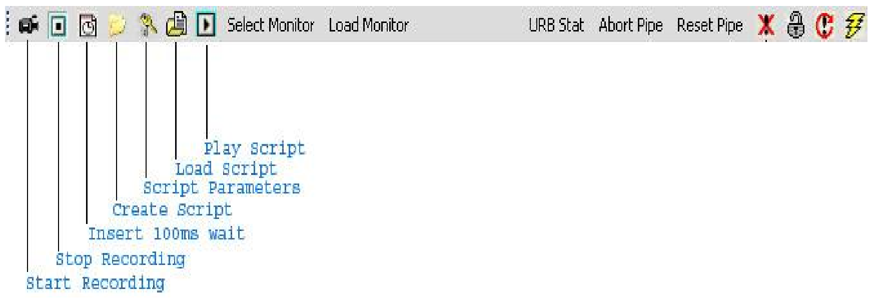
1. Connect a USB A-to-B cable between Windows PC USB Host port and J2 connector on MoBL-USB FX2LP18 development board.
2. The board should have switch settings SW2 = SAFE, SW1 = either position. The MoBL-USB FX2LP18 device enumerates with default Connect mode EEPROM (U5) VID/PID.
3. Select **Record Script** button.In the options menu select EZ-USB Interface. Select Download in the new pop up window and browse to **Bulkloop.hex** at <Installed_directory>\<Version>\Firm-ware\Bulkloop
4. Observe the **Bulkloop.hex** file getting downloaded successfully to MoBL-USB FX2LP18 internal RAM memory. After firmware download click **Stop recording** button and save the .spt file gener-ated into the local directory of choice.

5. Select Load script button and choose the xxx.spt file generated in step-4.
6. Click on Play script button. The entire firmware gets downloaded and the MoBL-USB FX2LP18 re-enumerates again with VID/PID -0x04B4/0x1004 defined in the firmware.
7. Open device manager by clicking on Windows start >Run. Type **devmgmt.msc** in the Run box. In windows Vista/7 the **devmgmt.msc** can be typed directly into the vacant box at Windows **Start**. If *cyusbfx2lp18.inf* file was binded to MoBL-USB FX2LP18 board similar to method defined in section Binding Cypress USB Driver to MoBL-USB Development Board on page 43 then MOBL-USB FX2LP18 device enumerates as **“Cypress EZ-USB Example Device(3.4.5.000)”**.The EZ-USB(FX1/FX2LP) and MoBL-USB(FX2LP18) has common set of examples using the same VID/PID.The common file has string mentioned above and this does not effect the functionality of the code example.

6.3.1.3 *Script Generation and Play using CyControlCenter*

Open this tool located at **C:\Cypress\Cypress Suite USB 3.4.7\CyUSB.NET\examples\Control Center\bin\Release** after installing MOBL-USB FX2LP18 kit contents. Click on **C:\Cypress\Cypress Suite USB 3.4.7\CyConsole**. Alternately the tool can be accessed at Windows **Start>All programs>Cypress>Cypress SuiteUSB 3.4.7**. Follow the below process to generate and play the script.

Figure 6-7. CyControlCenter Script Button Options



1. Connect a USB A-to-B cable between Windows PC USB Host port and MoBL-USB FX2LP18 development board J2 connector. The board should have SW2 switched to the side marked as SAFE EEPROM and SW1 can be in either position.
2. The MoBL-USB FX2LP18 enumerates with default connect mode EEPROM VID/PID(0x04B4/0x8614).
3. Click on **Create Script** button and select Bulkloop.hex at <Installed_directory>\<Version>\Firmware\Bulkloop.Save the file as Bulkloop.spt to any local directory of choice.
4. Select **Load script** button and choose the Bulkloop.spt file generated in step-3.
5. Click on **Play script**. The entire firmware gets downloaded and the MoBL-USB FX2LP18 device and it re-enumerates again with VID/PID -0x04B4/0x1004 defined in the firmware.
6. Open device manager by clicking on Windows **Start>Run**. Type devmgmt.msc in the Run box. In Windows Vista and 7 OS platforms type devmgmt.msc directly in the vacant box near Windows **Start** button. If cyusbfx2lp18.inf was binded to MOBL-USB FX2LP18 previously as mentioned in section Binding Cypress USB Driver to MoBL-USB Development Board chapter on page 43 then MoBL-USB device enumerates as **“Cypress EZ-USB Example Device(3.4.5.000)”**. The EZ-USB(FX1/FX2LP) and MoBL-USB(FX2LP18) has common set of examples using the same VID/PID. The common file has string mentioned above and this does not effect the functionality of the firmware example.

6.3.2 Keil Debug Monitor Download using Script and CyMonfx2lp18 Driver Package

The keil debug monitor is used to debug the firmware examples based on MoBL-USB FX2LP18 kit. The keil monitor program is embedded in script file (mon_i2c.spt). The procedure to generate the script file for a .hex file was explained in section. Using Sample keil monitor .hex programs located at <Installed_directory>\<Version>\Target\Monitor the relevant script file can be generated. Alternatively the current script file mon_i2c.spt can be used to debug the firmware example. The detailed list of steps to debug a sample firmware example dev_io was explained in section Debugging Using Keil Monitor Program on page 90.

6.4 SuiteUSB Driver Packages

Along with the MoBL-USB FX2LP18 kit contents SuiteUSB package is also provided. After MoBL-USB Kit installation the SuiteUSB package contents can be located by default at C:\Cypress\Cypress Suite USB 3.4.7. The Cypress generic USB drivers are located in the Driver\bin folder. The cypress generic USB drivers (ver -3.4.7) are located in this directory for different Windows OS platforms. These are unsigned drivers. For testing the applications provided with SuiteUSB the Signed driver package provided under <Installed_directory>\<Version>\Drivers\cyusbfx2lp18 can be used. The relevant VID/PID necessary to verify the functionality of each of these SuiteUSB applications was mentioned in Table 7-1 on page 50.

7. USB PC Host Utilities and SuiteUSB Applications

This chapter explains in brief about the USB PC Host utilities provided with CY3687 MoBL-USB FX2LP18 kit. Additionally SuiteUSB installer is provided which supports a collection of USB Host application tools designed using C++ and C# .NET software design tools. These host applications are used to communicate with Cypress USB 2.0 devices like MoBL-USB FX2LP18 and EZ-USB(FX1/FX2LP).

7.1 USB Applications in MoBL-USB FX2LP18 Development Kit

The MoBL-USB FX2LP18 development kit contains two utilities under \Bin directory. They are

1. **CyScript.exe:** This utlity is used to generate script file (.spt) for the corresponding RAM image file (.hex).The procedure to generate script file is explained in section How to Generate and Play Script Files (.spt) on page 45.
2. **hex2bix.exe:** The utlity is used to convert RAM images (.hex) files to equivalent EEPROM images(.iic).For detailed options of hex2bix utlity refer to application note titled "AN45197 - Using the Hex2bix Conversion Utility".

7.2 SuiteUSB Applications

SuiteUSB tools can be used to communicate with any cypress USB 2.0 device. The USB driver package provided are generic Cypress USB drivers (cyusb.sys and cyusbinf). Suite USB 3.4.7 installer executable is provided in the CD/DVD under **SuiteUSB** folder with respect to CD/DVD root directory. The Kit installer automatically triggers the SuiteUSB package installation. The default directory of installation is C:\Cypress\Cypress Suite USB 3.4.7. Following are the list of C++ tools and their equivalent C# .NET application tools provided in SuiteUSB.If there are no relevant tools in C# .NET the corresponding column is marked as NA (Not applicable)

.

Table 7-1. List of SuiteUSB C++ and C# Applications

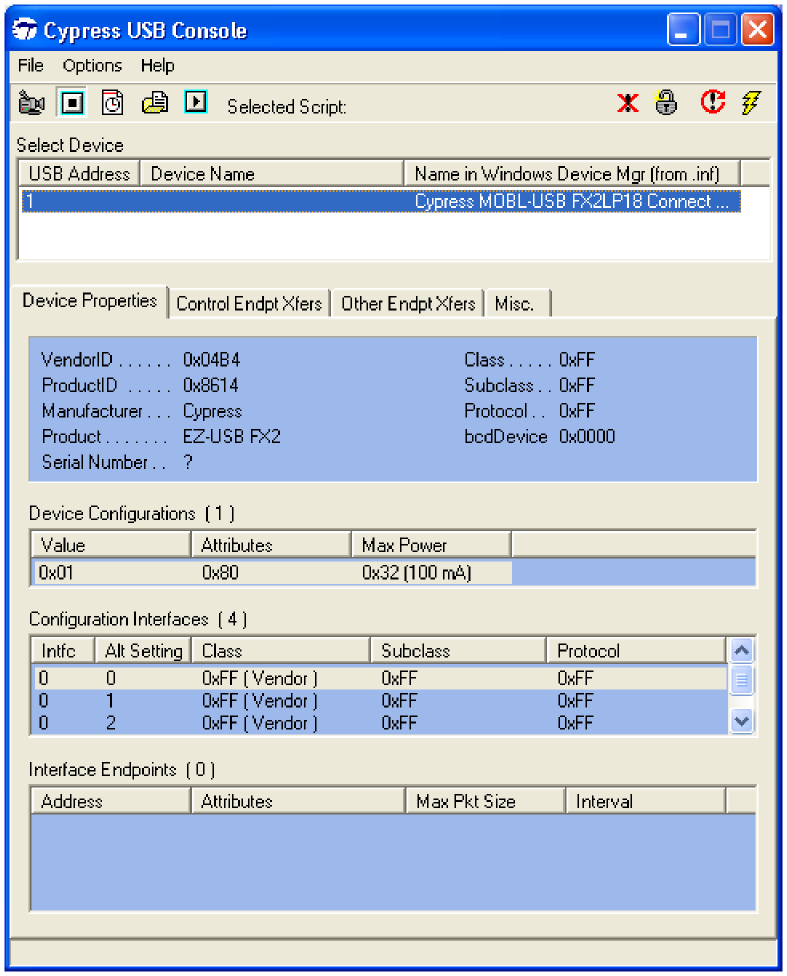
S.No	C++ Application	C#.NET Application	Description
1	Cyconsole	CyControlCenter	Both these applications are used for general purpose tasks like firmware download to MoBL-USB FX2LP18 RAM, Large EEPROM(U8). Additionally Script recording and download options are available.
2	Streamer	Streamer	Both these applications continuously Stream Bulk and Isochronous data in OUT and IN direction. The OUT and IN endpoint act as Sink and source of data i.e. the received data on OUT endpoint is discarded and the relevant endpoint re-armed again and a constant size of data is sent over IN endpoint. Using Cystream firmware located at C:\Cypress\Cypress Suite USB 3.4.7\Firmware\CyStreamer these applications can be verified.
3	cybulk	Bulkloop	Both these utilities perform the same functionality of looping back USB packet data received on Bulk OUT endpoint to a Bulk IN endpoint. Using **Bulkloop** firmware example provided with the kit these applications can be tested. Refer MoBL-USB Development Kit Firmware Examples chapter on page 59 for more details
4	cydesc	-NA-	The utlity provides the USB descriptor information of Cypress USB 2.0 devices connected to Windows PC.
5	FxEEPROM	-NA-	The utility gets activated only with default FX2LP Fuse ROM VID/PID and hence cannot be used for firmware download to MoBL-USB FX2LP18 development board.

The C++ applications use CyAPI.lib to communicate with the Cypress USB device.The C# .NET framework applications use CyUSB.dll to communicate with the hardware.

7.2.1 Cyconsole Utility

The Cyconsole performs tasks like firmware download to MoBL-USB FX2LP18 RAM, Small EEPROM (U8) and Large EEPROM(U7). Additionally it can be used perform script generation, loading, etc.

Figure 7-1. CyConsole Main Window Snapshot



USB Address	Device Name	Name in Windows Device Mgr (from .inf)
1		Cypress MOBL-USB FX2LP18 Connect ...

VendorID	0x04B4	Class	0xFF
ProductID	0x8614	Subclass . .	0xFF
Manufacturer . . .	Cypress	Protocol . .	0xFF
Product	EZ-USB FX2	bcdDevice	0x0000
Serial Number . .	?		

Device Configurations (1)

Value	Attributes	Max Power
0x01	0x80	0x32 (100 mA)

Configuration Interfaces (4)

Intfc	Alt Setting	Class	Subclass	Protocol
0	0	0xFF (Vendor)	0xFF	0xFF
0	1	0xFF (Vendor)	0xFF	0xFF
0	2	0xFF (Vendor)	0xFF	0xFF

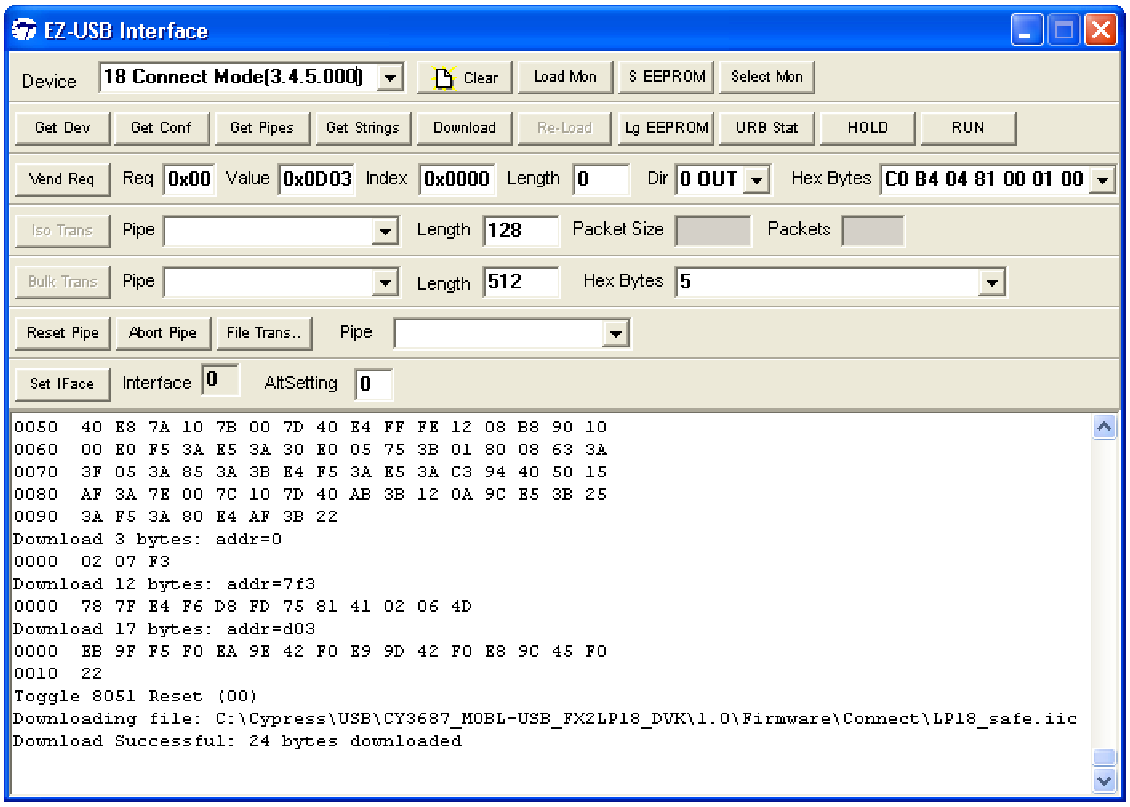
Interface Endpoints (0)

Address	Attributes	Max Pkt Size	Interval

As shown in the above figure it displays the connected Cypress USB 2.0 device (e.g MoBL-USB FX2LP18 in this case) and its attributes like USB class and list of endpoints it supports.

Prior to firmware download the **Record Script** button on top left corner is used to record the entire download process including the firmware binary embedded inside it. After firmware download is complete the **Stop Recording** button is clicked to save the entire download into a script file - **xxx.spt**.To verify the script it can be loaded using Load script button and played using **Play Script**. Due to firmware embedded inside the script getting downloaded the MoBL-USB FX2LP18 development board will re-enumerate again with new VID/PID defined in the firmware. The process of script download was mentioned in detailed in section How to Generate and Play Script Files (.spt) on page 45. Click on **Options->EZ-USB Interface** and the window shown below in Figure 7-2 pops up.

Figure 7-2. MoBL-USB Interface Window



Device: 18 Connect Mode(3.4.5.000)

Req 0x00 Value 0x0D03 Index 0x0000 Length 0 Dir 0 OUT Hex Bytes C0 B4 04 81 00 01 00

Length 128

Length 512 Hex Bytes 5

Interface 0 AltSetting 0

```
0050   40 E8 7A 10 7B 00 7D 40 E4 FF FE 12 08 B8 90 10
0060   00 E0 F5 3A E5 3A 30 E0 05 75 3B 01 80 08 63 3A
0070   3F 05 3A 85 3A 3B E4 F5 3A E5 3A C3 94 40 50 15
0080   AF 3A 7E 00 7C 10 7D 40 AB 3B 12 0A 9C E5 3B 25
0090   3A F5 3A 80 E4 AF 3B 22
Download 3 bytes: addr=0
0000   02 07 F3
Download 12 bytes: addr=7f3
0000   78 7F E4 F6 D8 FD 75 81 41 02 06 4D
Download 17 bytes: addr=d03
0000   EB 9F F5 F0 EA 9E 42 F0 E9 9D 42 F0 E8 9C 45 F0
0010   22
Toggle 8051 Reset (00)
Downloading file: C:\Cypress\USB\CY3687_MOBL-USB_FX2LP18_DVK\1.0\Firmware\Connect\LP18_safe.iic
Download Successful: 24 bytes downloaded
```

The functionality of most frequently used buttons are briefly mentioned below

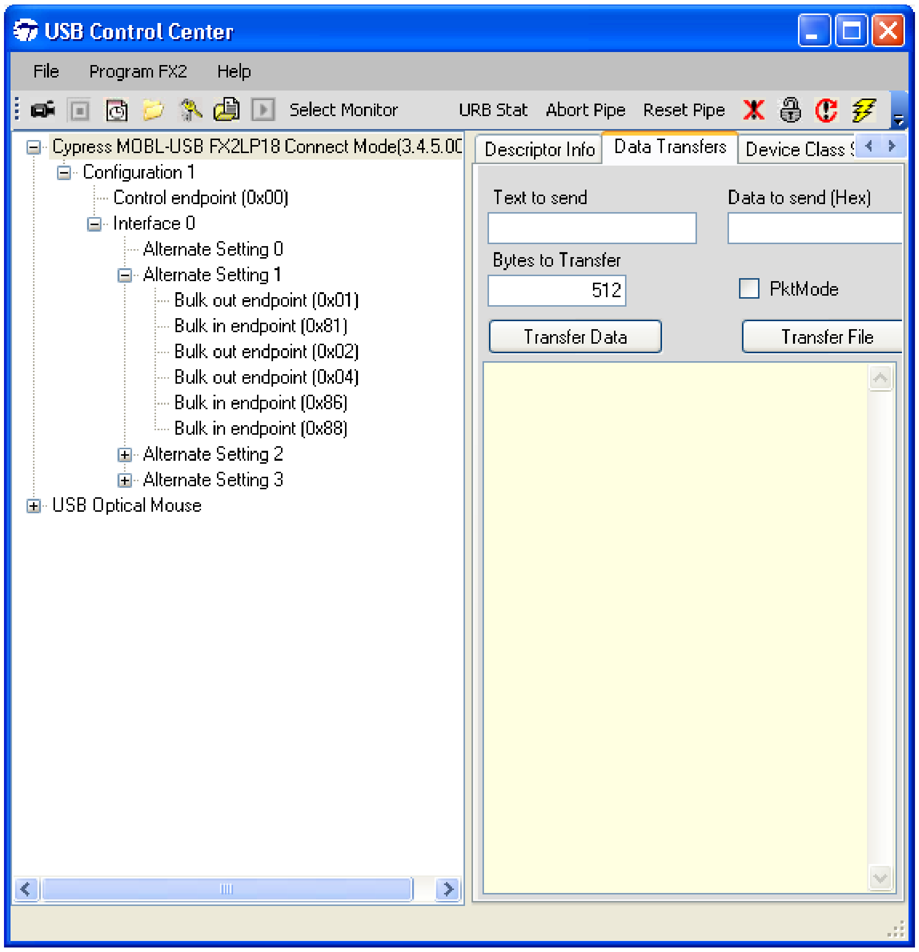
- **Download:** This button is used to download the firmware(.hex format) file to MoBL-USB FX2LP18 RAM memory
- **Lg EEPROM:** This button is used to store the entire firmware (.iic) in Large EEPROM U7 - 24AA128. The EEPROM needs to be selected before firmware download using SW1 and SW2 switches.
- **S EEPROM:** This button is used to store the entire firmware (.iic) in small EEPROM U8 - 24AA01.The image contains a new VID/PID used tfor keil Monitor debugging. The EEPROM needs to be selected before firmware download using SW1 and SW2 switches.
- **Select Mon:** Keil Monitor program provided with the MoBL-USB FX2LP18 kit to help user debug the firmware through UART port. After Kit software installation the Keil Monitor programs are located at **<Installed_directory>\<Version>\Target\Monitor**.
- **Load Mon:** This button is used to download Keil Monitor program to either internal RAM memory. After monitor download the MoBL-USB FX2LP18 firmware is debugged through UART port. A sample demonstration of Keil monitor usage is provided in section Debugging Using Keil Monitor Program on page 90
- **Vend Req:** This button is used to send different vendor commands to MoBL-USB FX2LP18 device. The usage of this button is explained in detail in section extr_intr Firmware Example on page 83
- **Iso Trans:** This button is used transfer data over Isochronous IN/OUT endpoints. After Cystream firmware example(CYStream.hex) at C:\Cypress\Cypress Suite USB 3.4.7\Firmware\CyStreamer this button is used to send data over Isochronous IN/OUT endpoints

- **BulkTrans:** This button is used transfer data over Bulk IN/OUT endpoints. After Bulkloop firmware example is downloaded (Bulkloop.hex) this button is used to send data over Bulk IN/OUT endpoints using Bulk Trans button. A sample demonstration is provided under section Bulkloop Example on page 71. For more details on Cyconsole refer to CyConsole.chm/CyConsole.pdf at C:\Cypress\Cypress Suite USB 3.4.7\CyConsole. A sample demonstration of this utility while using firmware examples is provided in MoBL-USB Development Kit Firmware Examples chapter on page 59 of this document.
- **FileTrans:** This button is used to download raw packet data to MoBL-USB FX2LP18 device.The sample files that could be used to transfer are available at <Installed_directory>\<Version>\Target\File_Transfer.

7.2.2 CyControlCenter Utility

The CyControlCenter performs functions similar to CyConsole application. Following are major functions of CyControlCenter

Figure 7-3. CyControlCenter Snapshot for MoBL-USB FX2LP18 Device



Cypress MOBL-USB FX2LP18 Connect Mode(3.4.5.00
Configuration 1
Control endpoint (0x00)
Interface 0
Alternate Setting 0
Alternate Setting 1
Bulk out endpoint (0x01)
Bulk in endpoint (0x81)
Bulk out endpoint (0x02)
Bulk out endpoint (0x04)
Bulk in endpoint (0x86)
Bulk in endpoint (0x88)
Alternate Setting 2
Alternate Setting 3
USB Optical Mouse

Bytes to Transfer: 512

- **Firmware Download:** After connecting MoBL-USB FX2LP18 development board in SAFE EEPROM mode (switch SW2 on board to SAFE) the firmware can be download to RAM, small EEPROM or the large EEPROM.The procedure to download is explained in detail in MoBL-USB Development Kit Firmware Examples chapter on page 59 for each of the firmware examples provided with the kit.

- **Script file generation and automatic firmware download:** This utility can be used to generate script file for a relevant firmware .hex file and later use the script file to automatically download the firmware using the script. The process of automatic firmware download using scripts is explained in section How to Generate and Play Script Files (.spt) on page 45.
- **Data transfers:** Using the tool the USB packet data can be transferred over an endpoint. The procedure is explained in detail in CyControlCenter.pdf located at C:\Cypress\Cypress Suite USB 3.4.7\CyUSB.NET\

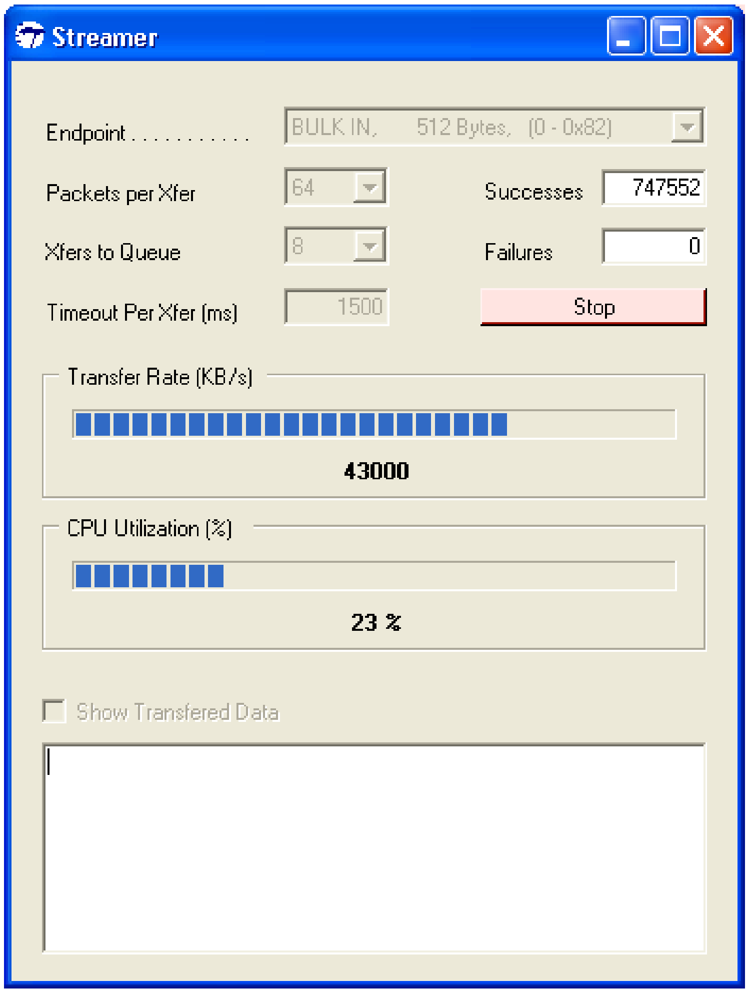
7.2.3 Streamer Utility

The Streamer utilities are available in both C++ and C#.NET framework versions. These utilities are used to test the Bulk and Isochronous data transfer throughput in both IN and OUT directions.

- Download **cystream.hex** located at C:\Cypress\Cypress Suite USB 3.4.7\Firmware\CyStreamer using CyConsole or CyControlCenter.The procedure to download firmware to RAM memory is explained in MoBL-USB Development Kit Firmware Examples chapter on page 59
- Open anyone of the Streamer application at the following locations:

1. **Streamer using C++ CYAPI.lib:** For 32-bit Windows OS platforms the utility is located at **C:\Cypress\Cypress Suite USB 3.4.7\CyAPI\examples\Streamer\x86\Release**. For 64-bit OS platforms refer to **C:\Cypress\Cypress Suite USB 3.4.7\CyAPI\examples\Streamer\x64\Release**

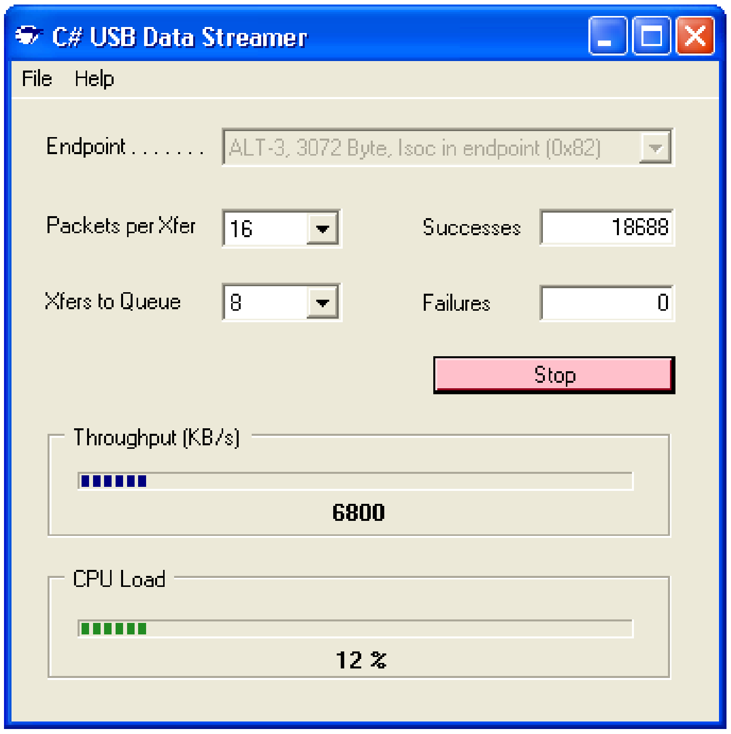
Figure 7-4. Streamer(C++) throughput on Bulk OUT Endpoint



Select the relevant Bulk or Isochronous In/OUT endpoint .Vary The **Packets per Xfer** and **Xfers to Queue** parameters and verify the throughput for different Bulk and Isochronous endpoints across different Alternate interfaces.

2.**Streamer using C# .NET CYUSB.dll:** The throughput can also be measured using this utility available at **C:\Cypress\Cypress Suite USB 3.4.7\CyUSB.NET\examples\Streamer\bin\Release**.

Figure 7-5. Streamer throughput on Iso IN Endpoint


C# USB Data Streamer

Endpoint .... ALT-3, 3072 Byte, Isoc in endpoint (0x82)

Packets per Xfer 16

Successes 18688

Xfers to Queue 8

Failures 0

Throughput (KB/s) 6800

CPU Load 12 %

Select the relevant Bulk or Isochronous In/OUT endpoint. Vary The **Packets per Xfer** and **Xfers to Queue** parameters and verify the throughput for different Bulk and Isochronous endpoints across different Alternate interfaces.

Note: The maximum data allowed per transfer is 64KB for Bulk/Isochronous transfers.

7.2.4 Bulkloop Application

The Bulkloop firmware example can be tested using either cybulk (C++) application or Bulkloop.exe (C# .NET application).

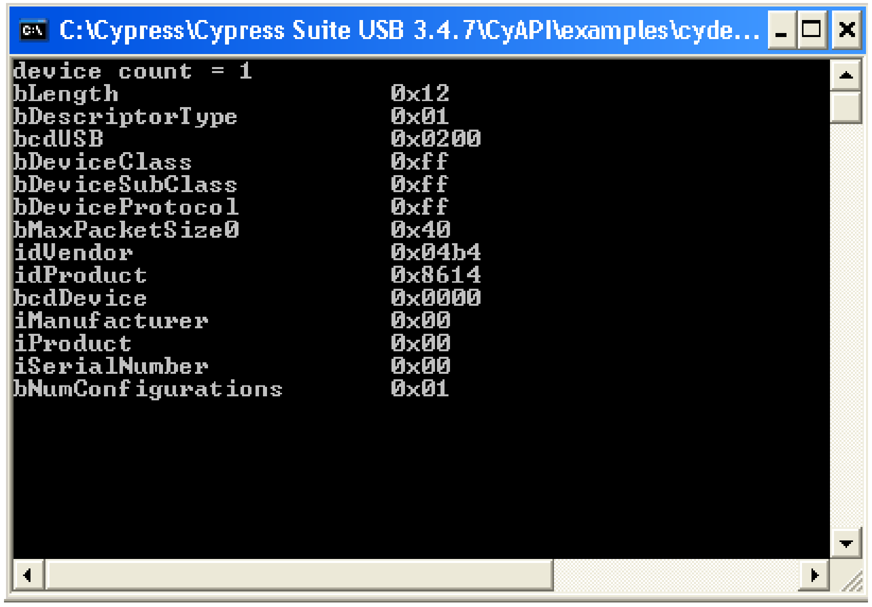
- CyBulk application for 32-bit Windows OS is located at C:\Cypress\Cypress Suite USB 3.4.7\CyAPI\examples\cybulk\x86\Release
- CyBulk application for 64-bit Windows OS is located at C:\Cypress\Cypress Suite USB 3.4.7\CyAPI\examples\cybulk\x64\Release
- Bulkloop.exe is located at C:\Cypress\Cypress Suite USB 3.4.7\CyUSB.NET\bin

The procedure to test these applications using **Bulkloop** firmware example is explained in MoBL-USB Development Kit Firmware Examples chapter on page 59.

7.2.5 Cydesc Utility

The utlity is used to view the USB device descriptor of Cypress USB 2.0 devices. Following figure shows the MoBL-USB FX2LP18 device default Connect Mode device descriptor.

Figure 7-6. Cydesc Display of MoBL-USB FX2LP18 Device Descriptor



```
device count = 1
bLength                    0x12
bDescriptorType            0x01
bcdUSB                     0x0200
bDeviceClass               0xff
bDeviceSubClass            0xff
bDeviceProtocol            0xff
bMaxPacketSize0            0x40
idVendor                   0x04b4
idProduct                  0x8614
bcdDevice                  0x0000
iManufacturer              0x00
iProduct                   0x00
iSerialNumber              0x00
bNumConfigurations         0x01
```

8. MoBL-USB Development Kit Firmware Examples

This chapter explains in detail about firmware example and how to test each firmware example provided with the kit. The MoBL-USB FX2LP18 contain common set of firmware examples to demonstrate the MoBL-USB FX2LP18 capabilities along with the various components on board (Seven segment LED, Push button, RAM memory, EEPROM, and so on). It also explains how to debug a firmware example using Keil uVision2 IDE.Following are the firmware examples provided with the kit along with their brief description.

Table 8-1. List of Firmware Example in MoBL-USB FX2LP18 Development Kit

S.No	Firmware Example	Description
1	hid_kb	Example firmware that emulates a HID-class keyboard using the buttons and 7-segment display on the DVK board
2	Bulkloop	Contains a bulk loopback test that exercises the MoBL-USB FX2LP18 bulk endpoints. It loops back EP2OUT to EP6IN and EP4OUT to EP8IN
3	Bulkext	Contains a bulk loopback test that exercises the MoBL-USB FX2LP18 bulk endpoints. The loopback is performed using the external auto pointer. Data is copied from the OUT endpoint buffer to external RAM and then to the IN endpoint buffer. It loops back EP2OUT to EP6IN and EP4OUT to EP8IN
4	Bulksrc	Contains bulk endpoint endless source/sink firmware. It can be driven using the CyConsole or CyBulk. EP2OUT always accepts a bulk OUT; EP4OUT always accept a bulk OUT; EP6IN always returns a 512-byte packet, 64 bytes at full-speed. Based on buffer availability in EP8IN, the most recent packet of EP4OUT is written to EP8IN.
5	dev_io	Contains the source files to build simple development board I/O sample. This software demonstrates how to use the buttons and LED on the MoBL-USB FX2LP18 development kit.
6	EP_Interrupts	Bulk loopback firmware that demonstrates use of endpoint interrupts using MoBL-USB FX2LP18.
7	extr_intr	Firmware that demonstrates external interrupt handling- INT0, INT1, INT4, INT5 and INT6.
8	Ibn	Contains firmware to perform bulk loopback of EP2OUT to EP6IN and EP4OUT to EP8IN using the IBN (In Bulk Nak) interrupt to initiate the transfer.
9	Pingnak	Contains firmware to perform bulk loopback of EP2OUT to EP6IN and EP4OUT to EP8IN using the PING NAK interrupt to initiate the transfer
10	iMemtest	The firmware example tests on-chip RAM
11	vend_ax	Contains the source files to build a vendor specific command sample. This example demonstrates how to implement different vendor commands

Note: All the above Firmware examples except hid_kb use common VId/PID-0x04B4/0x1004. The hid_kb project uses VId/PID-0x04B4/0x1005.

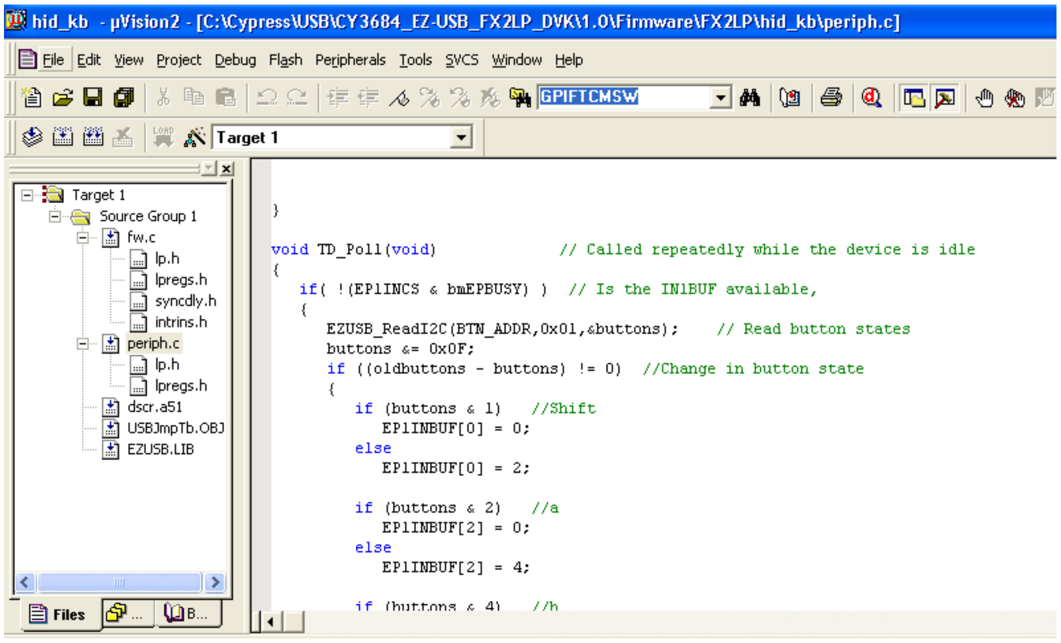
8.1 Method to Verify the Code Example Functionality

The code examples provided with the kit can be verified using MoBL-USB FX2LP18 development board provided with the kit. There are different types of firmware download mechanism for MoBL-USB FX2LP18 devices. After firmware was successfully downloaded the MoBL-USB FX2LP18 device re-enumerates and prompts for a relevant windows USB driver. The user needs to bind to the appropriate cyusbxxx.inf and cyusb.sys files provided with the kit. Finally when the driver binding is complete relevant PC tools can be used to test the firmware functionality. To demonstrate each of these code examples the entire process is divided into various stages for each example.

8.2 hid_kb Code Example

This example describes the implementation of a 4-button Virtual HID Class keyboard using the MoBL-USB FX2LP18 development kit board. Open hid_kb.Uv2 project file in Keil uVision2 IDE. Figure 8-1 provides the project snapshot in the IDE.

Figure 8-1. hid_kb Project View in keil uVision2 IDE


hid_kb - µVision2 - [C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Firmware\FX2LP\hid_kb\periph.c]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
GPIFTCMSW
Target 1
Target 1
Source Group 1
fw.c
lp.h
lpregs.h
syncdly.h
intrins.h
periph.c
lp.h
lpregs.h
dscr.a51
USBJmpTb.OBJ
EZUSB.LIB
Files
B...

```
}

void TD_Poll(void)              // Called repeatedly while the device is idle
{
   if( !(EP1INCS & bmEPBUSY) )  // Is the IN1BUF available,
   {
      EZUSB_ReadI2C(BTN_ADDR,0x01,&buttons);    // Read button states
      buttons &= 0x0F;
      if ((oldbuttons - buttons) != 0)  //Change in button state
      {
         if (buttons & 1)   //Shift
            EP1INBUF[0] = 0;
         else
            EP1INBUF[0] = 2;

         if (buttons & 2)   //a
            EP1INBUF[2] = 0;
         else
            EP1INBUF[2] = 4;

         if (buttons & 4)   //b
```

The firmware example detects if any of the four push buttons are pressed (on the development board) and sends relevant fixed data to the Host PC. For the HID class devices such as keyboard, Mouse, the USB bandwidth requirements is typically 64 KB/sec. Most of the HID devices are either low-speed or full-speed devices. Due to this low data rate requirements of the device, only the end-point EP1(64 byte buffer) is selected for both IN and OUT interrupt transfers. The high-speed data endpoints EP2, EP4, EP6, and EP8 are disabled as shown in the following code snippet:

```
EP1OUTCFG = 0xB0; // valid, interrupt OUT, 64 bytes, Single buffered
EP1INCFG = 0XB0; // valid, interrupt IN, 64 bytes, Single buffered
EP2CFG = EP4CFG = EP6CFG = EP8CFG = 0; // disable unused endpoints
```

For a typical HID device, the data related to events like button press, key strokes, mouse clicks are transferred to Host in the form of Input Reports using an Interrupt IN endpoint. Similarly, the Reports can be requested by Host PC using control endpoint or an Interrupt OUT endpoint. The firmware sets EP1IN and EP1OUT as Interrupt endpoints for data transfers. The following table summarizes the mapping of Push buttons on the MoBL-USB FX2LP18 development board to keyboard buttons.

Table 8-2. Function Mapping of Development Board Buttons

MoBL-USB FX2LP18 Development Board Push Button	Function
f1	Shift
f2	Send ‘a’
f3	Send ‘b’
f4	Send ‘c’

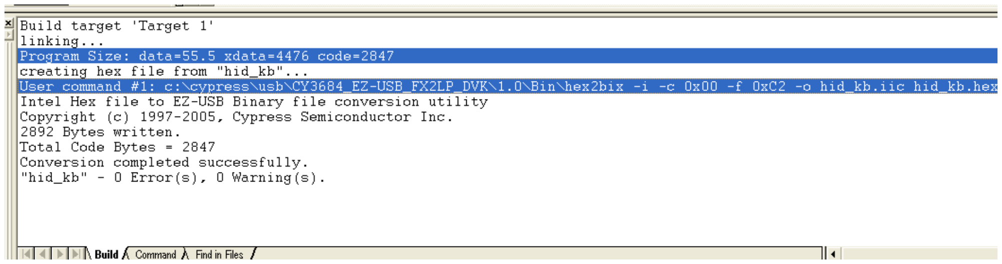
The function TD_poll () in the firmware (periph.c) is where the periodic checking of a new push button event is done. Following is the code snippet of this function.

```
if( !(EP1INCS & bmEPBUSY) ) // Is the EP1INBUF
                            //available,
{
EZUSB_ReadI2C(BTN_ADDR,0x01,&buttons); // Read button states
buttons &= 0x0F;
if ((oldbuttons - buttons) != 0) //Change in button state
{
if (buttons & 1) //Shift
EP1INBUF[0] = 0;
else
EP1INBUF[0] = 2;
if (buttons & 2) //a
EP1INBUF[2] = 0;
else
EP1INBUF[2] = 4;
if (buttons & 4) //b
EP1INBUF[3] = 0;
else
EP1INBUF[3] = 5;
if (buttons & 8) //c
EP1INBUF[4] = 0;
else
EP1INBUF[4] = 6;
EP1INBUF[1] = 0;
EP1INBC = 5;
}
oldbuttons = buttons;
}
```

8.2.1 Building Firmware Example Code for MoBL-USB FX2LP18 Internal RAM and External EEPROM.

- Click on **Build Target button** at the top right corner of the IDE. Figure 8-2 of the **Build** window of the Keil IDE shows the successful compilation of the entire project.

Figure 8-2. Build Window Snapshot of hid_kb Project

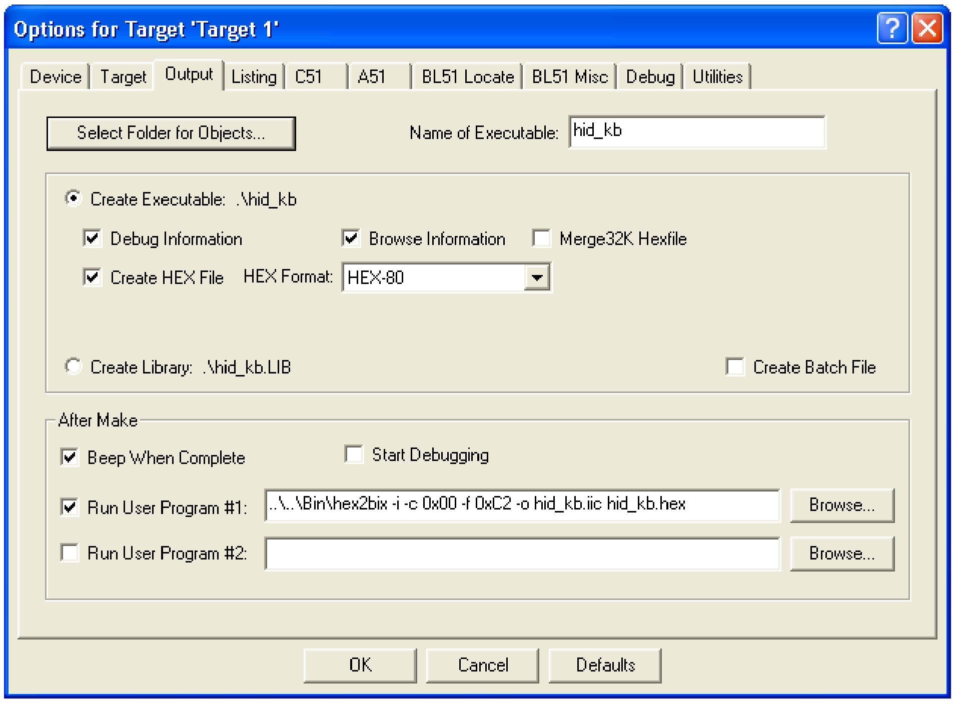


```
Build target 'Target 1'
linking...
Program Size: data=55.5 xdata=4476 code=2847
creating hex file from "hid_kb"...
User command #1: c:\cypress\usb\CY3684_EZ-USB_FX2LP_DVK\1.0\Bin\hex2bix -i -c 0x00 -f 0xC2 -o hid_kb.iic hid_kb.hex
Intel Hex file to EZ-USB Binary file conversion utility
Copyright (c) 1997-2005, Cypress Semiconductor Inc.
2892 Bytes written.
Total Code Bytes = 2847
Conversion completed successfully.
"hid_kb" - 0 Error(s), 0 Warning(s).
```

Note: Observe that the total Code bytes of **hid_kb** project is less than the 4k code limit of Keil uVision2 IDE provided along with the kit.

- **Firmware for MoBL-USB FX2LP18 RAM memory:** The output of the **Build Target** is **hid_kb.hex**. It is relevant file for downloading to MoBL-USB FX2LP18 RAM memory.
- **Firmware for external I2C EEPROM:** To generate EEPROM compatible firmware Image the Keil IDE invokes **hex2bix.exe** utility to convert the output file **hid_kb.hex** into **hid_kb.iic**. Right click on **Target1** in Project Window and select **Options for Target** '**Target1**'. This will pop-up keil settings for this project.Select **Output** tab and observe at the bottom of IDE the hex2bix utility is invoked as shown in Figure 8-3.

Figure 8-3. hid_kb Project Output Image Settings



Options for Target 'Target 1'

Device | Target | Output | Listing | C51 | A51 | BL51 Locate | BL51 Misc | Debug | Utilities

Select Folder for Objects... Name of Executable: hid_kb

(●) Create Executable: .\hid_kb
[✓] Debug Information [✓] Browse Information [ ] Merge32K Hexfile
[✓] Create HEX File HEX Format: HEX-80

( ) Create Library: .\hid_kb.LIB [ ] Create Batch File

After Make
[✓] Beep When Complete [ ] Start Debugging
[✓] Run User Program #1: ..\..\Bin\hex2bix -i -c 0x00 -f 0xC2 -o hid_kb.iic hid_kb.hex
[ ] Run User Program #2:

Under **Run User program#1** section observe the hex2bix utility is invoked in the following manner:

```
..\..\Bin\hex2bix -i -c 0x00 -f 0xC2 -o hid_kb.iic hid_kb.hex
```

Refer to application note, “Using the hex2bix Conversion Utility - AN45197” to know more details on the hex2bix utility.

8.2.2 Method to Download Firmware Image to MoBL-USB Internal RAM Memory

1. On the MoBL-USB FX2LP18 development board select switch **SW2** to **SAFE** side.

2. Connect USB A-to-B cable from J2 connector on board to a Windows PC USB Host controller port.
3. The MoBL-USB FX2LP18 development board should by default bind to *cyusbfx2lp18.inf* in the ***\ Drivers*** folder at <Installed_directory>\<version>\ for corresponding OS. Refer to section Binding Cypress USB Driver to MoBL-USB Development Board on page 43 on how to bind this driver to MoBL-USB FX2LP18 development board. If the binding process is already performed ignore this step.
4. Open Cyconsole PC application from Windows **Start->All Programs->Cypress->Cypress SuiteUSB 3.4.7-->CyConsole.** Observe MoBL-USB FX2LP18 listed as **"Cypress MOBL-USB FX2LP18 Connect Mode(3.4.5.000)"**.
5. Click on **Options->EZ-USB Interface**. The EZ-USB Interface window pops up on top of CyConsole Window. To download **hid_kb.hex** to MoBL-USB internal RAM memory click on Download button and browse to the image path a `<Installed_directory>\<Version>\Firmware\hid_kb`
6. After download the image does not require a cypress USB driver for testing the 4-button virtual keyboard functionality.The complete functionality is handled by Microsoft Windows OS native HID drivers.

8.2.3 Method to Download Firmware Image to External I2C EEPROM

1. On the MoBL-USB FX2LP18 board select **SW2-SAFE and SW1** can be in either position to select U5. and connect USB A-to-B cable from J2 connector on board to a Windows PC USB Host controller port.The MoBL-USB FX2LP18 device is enumerates with connect mode VID/PID(0x04B4/0x8614).

2. Before programming the EEPROM image file (.iic) Select **SW2-LG-SM and SW1-LARGE EEPROM** as switch settings to select large EEPROM U7 on board.

3. Open Cyconsole PC application from Windows **Start->All Programs->Cypress->Cypress SuiteUSB 3.4.7-->CyConsole** as shown in the following Figure 8-13. Observe MOBL-USB FX2LP18 listed as **"Cypress MOBL-USB FX2LP18 Connect Mode(3.4.5.000)"**.

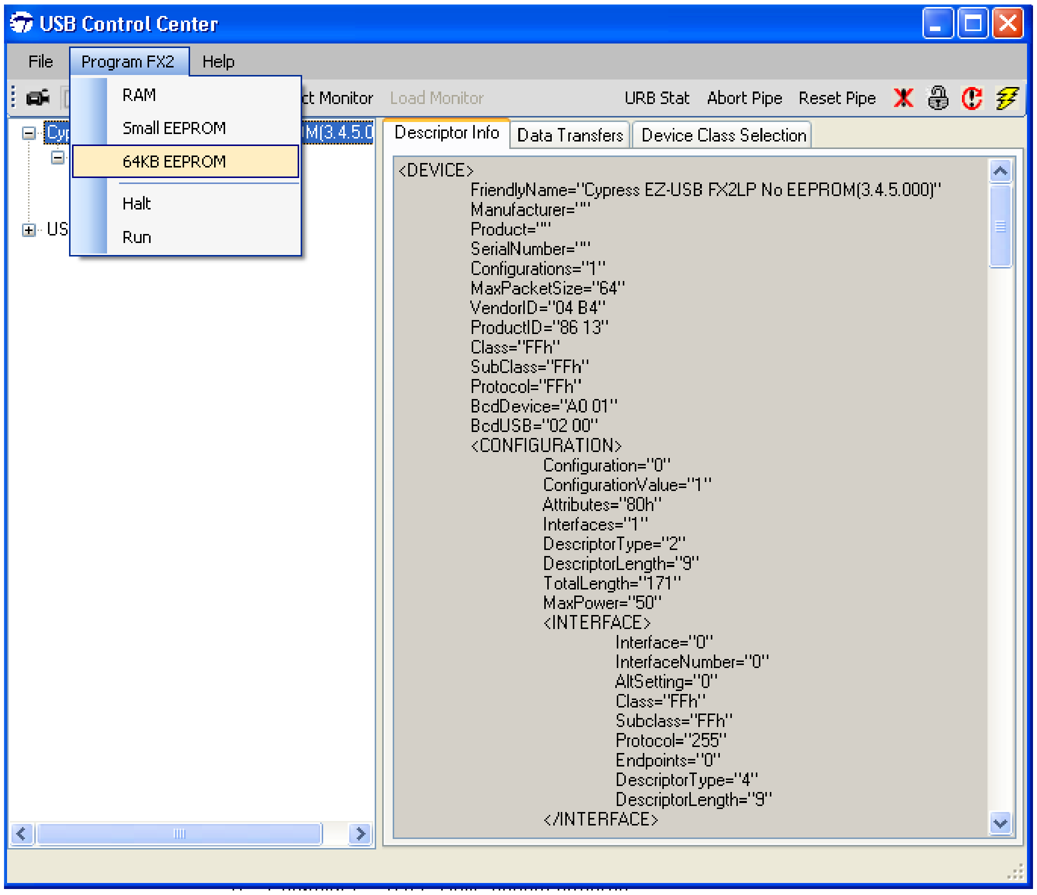
4. Click on **Lg EEPROM** button on EZ-USB Interface window and browse to project folder and select the image hid_kb.iic at <Installed_directory>\<Version>\Firmware\hid_kb.The EZ-USB interface window shows successful completion of image download to large EEPROM U7-24AA128

5. Press RESET button S6 again and this will eventually prompt the MoBL-USB FX2LP18 device to boot from Large EEPROM Image-**hid_kd.iic**.

6. After download the image does not require a cypress USB driver for testing the 4-button virtual keyboard functionality.The complete functionality is handled by Microsoft Windows OS native HID drivers.

7. The firmware image can also be downloaded using CyControlCenter. Open CyControlCenter PC application from Windows **Start->All programs->Cypress->Cypress Suite USB 3.4.7 -->Control Center**. Switch SW2 on board to NO EEPROM and press RESET button. Observe MOBL-USB FX2LP18 listed as **"Cypress MOBL-USB FX2LP18 Connect Mode(3.4.5.000)"**. Before **hid_kb.iic** file download select **SW1-LARGE EEPROM and SW2-LG-SM** options on board. Select Program FX2 '64KB EEPROM as shown in the following Figure 8-4. Browse and select hid_kb.iic file. The application automatically downloads the entire image to Large EEPROM-U7 on board.

Figure 8-4. CyControlCenter dIsplay To Download Image to Large EEPROM



```
<DEVICE>
        FriendlyName="Cypress EZ-USB FX2LP No EEPROM(3.4.5.000)"
        Manufacturer=""
        Product=""
        SerialNumber=""
        Configurations="1"
        MaxPacketSize="64"
        VendorID="04 B4"
        ProductID="86 13"
        Class="FFh"
        SubClass="FFh"
        Protocol="FFh"
        BcdDevice="A0 01"
        BcdUSB="02 00"
        <CONFIGURATION>
                Configuration="0"
                ConfigurationValue="1"
                Attributes="80h"
                Interfaces="1"
                DescriptorType="2"
                DescriptorLength="9"
                TotalLength="171"
                MaxPower="50"
                <INTERFACE>
                        Interface="0"
                        InterfaceNumber="0"
                        AltSetting="0"
                        Class="FFh"
                        Subclass="FFh"
                        Protocol="255"
                        Endpoints="0"
                        DescriptorType="4"
                        DescriptorLength="9"
                </INTERFACE>
```

Note: To download Image to small EEPROM follow similar steps 1-7.The only changes are onboard EEPROM settings in step-2 **(SW2-LG-SM and SW1-SMALL EEPROM)** which selects U8 EEPROM.The MOBL-USB FX2LP18 EEPROM accept only Images with 0xC2 as first Byte.This EEPROM is usually programmed during manufacturing with LP18_dvk.iic.This file is used for Keil monitor debugging. Additionally Cyconsole EZ-USB Interface Window select **S EEPROM** instead of **Lg EEPROM** before download and in CyControlCenter menu Select **Program FX2->small EEPROM** instead of 64 KB EEPROM.

8.2.4 Binding Cypress USB Driver for the Downloaded Firmware Image

The **hib_kb** project contains firmware for a HID class keyboard device (Interface class: HID =03 and subclass=00) and uses Microsoft native HID driver instead of Cypress generic USB driver.

8.2.5 Testing the hid_kb Firmware Example Functionality

The MoBL-USB FX2LP18 development board enumerates as Human Interface Device(HID). Open Device Manager Window by typing **devmgmt.msc** in Windows **Start->Run box**. In Windows Vista and 7 OS platforms type **devmgmt.msc** directly in the vacant box near **Start** button. The device will be shown as part of the HID devices list. Open a new notepad in windows and point the mouse to the text area of the notepad. Press button F2, F3, F4 sequentially and observe letters 'a', 'b', 'c' getting printed in the notepad. Press them simultaneously with F1 and observe alphabets 'A', 'B' and 'C' on the notepad.

8.3 IBN Firmware Example

8.3.1 Description

This example illustrates the configuration of MoBL-USB FX2LP18 to accept bulk data from the host and loop it back to the host using an IN-BULK-NAK interrupt. Click on **ibn.Uv2** project file at <Installed_directory>\<Version>\Firmware\ibn. In TD_init() function of **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 have to be configured in this function. This is done by the following statements:

```
EP2CFG = 0xA2;
SYNCDELAY;
EP4CFG = 0xA0;
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 two clock cycles needed for a MOVX instruction. Therefore, the SYNCDELAY, already defined, is added. The TRM provides the list of registers that need this delay function when writing to them. The OUT endpoints, after they are configured, need to be ARMed to accept packets from the host. Because the endpoints are double-buffered, you must arm the endpoint twice. Arming is essentially freeing up the buffers and making them available to the host to receive packets.

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 lines above 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 In-Bulk-NAK flags of all endpoints and any pending In-Bulk-NAK interrupts and enables the In-Bulk-NAK 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. This is done by reading the endpoint 2 empty bit in the endpoint status register (EP2468STAT). If endpoint 2 has data (that is sent from the host), then check if the host has requested data on EP6. This is done by reading the EP6 In-Bulk-Flag bit in the IbnFlag variable. If the host has requested for 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
```

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 have been enabled the pointers increment automatically, and the statement

```
EXTAUTODAT2 = EXTAUTODAT1;
```

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”, that is, make the FIFO buffers available to the host for reading data from the 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 EP6 In-Bulk-NAK Flag bit in the IbnFlag variable is cleared. The EP6 In-Bulk-NAK interrupt request is cleared by setting the corresponding bit in the IBNIRQ register. Finally, the EP6 In-Bulk-NAK 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 MoBL-USB FX2LP 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 MoBL-USB FX2LP18 device 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 IN-BULK-NAK (IBN) interrupt provides this notification. The IBN interrupt fires whenever a Bulk endpoint NAKs an IN request. The IBNIE/IBNIRQ registers contain individual enable and request bits per endpoint, and the NAKIE/NAKIRQ registers each contain a single-bit, IBN, that is the ORd combination of the individual bits in IBNIE/IBNIRQ, respectively. The MoBL-USB FX2LP18 device firmware framework provides hooks for all the interrupts that it implements. The example project uses ISR_Ibn interrupt service routine to handle In-Bulk-NAK(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
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;
}
```

8.3.2 Building Firmware Example Code for MoBL-USB FX2LP18 Internal RAM and EEPROM

Click on **Build Target** button at the top right corner of the IDE. The firmware example builds successfully since the total code bytes of **IBN** firmware example is less than the 4k code limit Keil uVision2 IDE provided along with the kit. The output of the **Build Target** is ibn.hex and ibn.iic files.

8.3.3 Method to Download Firmware Image to MoBL-USB FX2LP18 device Internal RAM and External EEPROM

Refer to section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 and Method to Download Firmware Image to External I2C EEPROM on page 63 and follow similar procedure to download **ibn.hex** to either RAM memory or **ibn.iic** to large EEPROM using Cyconsole/CyControlCenter.After download the firmware re-enumerates with PC using its internal VID/PID-0x04B4/0x1004.

8.3.4 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 Cypress generic USB driver cyusb.sys and driver information file *cyusbfx2lp18.inf* which contains the relevant VID/PID of this example. Follow the procedure outlined in section Binding Cypress USB Driver to MoBL-USB Development Board on page 43 to manually bind the driver using Windows Hardware Wizard.

8.3.5 Testing the IBN Firmware Functionality

Following are the detailed steps to test the functionality

1. After the board has enumerated, use CyConsole to send 512 bytes from EP2 to EP6.The data received should be same as the data sent. 512 bytes of user-defined data can be sent from the host to Endpoint 2 using CyConsole. For example, select Endpoint 2 OUT in the pipe window near **Bulk Trans** button of **EZ-USB interface** window, enter the length as 512 and HexBytes as 5, and then press the **Bulk Trans** button.
2. This data can be read back from Endpoint 6 using CyConsole For example, select Endpoint 6 IN in the pipe, enter the length as 512, and then press the **Bulk Trans** button. Similarly, loopback

using endpoint 4 and 8 can also be tested. Since EP2 and EP4 are double-buffered, they can only contain 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 In-Bulk-NAK interrupt is asserted and data is transferred from EP2 to EP6 or from EP4 to EP8. This data appears on the **EZ-USB Interface** Window.
4. The above can be tested 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.

8.4 Pingnak Firmware Example

8.4.1 Description

This project illustrates the configuration of MoBL-USB FX2LP18 device to accept bulk data from the host and loop it back to the host and the use of PING-NAK interrupt. Click on **pingnak.Uv2** located at <Installed_directory>\<Version>\Firmware\pingnak and observe the code. Four endpoints are configured in **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 have to be configured in this function. This is done by the following statements:

```
EP2CFG = 0xA2;
SYNCDELAY;
EP4CFG = 0xA0;
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 two clock cycles needed for a MOVX instruction. Therefore, the SYNCDELAY, already defined, is added. The MoBL-USB Technical Reference Manual at <Installed_directory>\<Version>\Documentation provides the list of registers that need this delay function when writing to them. The OUT endpoints, after they are configured, need to be ARMed to accept packets from the host. Because the endpoints are double-buffered, you must arm the endpoint twice. Arming is essentially freeing up the buffers and making them available to the host to receive packets. 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;
```

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. To implement this, endpoint 2 is first checked to see if it has data. This is done by reading the endpoint 2 empty bit in the endpoint status register (EP2468STAT). If endpoint 2 has data (that is sent from the host), the capability of endpoint 6 to receive the data is checked. This is done 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, 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 above statement is executed the auto pointer is incremented. The above statement is executed repeatedly to transfer each byte from endpoint 2 to 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', that is, 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
```

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 an OUT data to an endpoint, and the previous data transfer was responded by 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. MoBL-USB FX2LP18 implements PING-NAK interrupt as EP0PING, EP1PING, etc., one for each endpoint. The EPxPING interrupt is asserted when the host PINGs an endpoint and the MoBL-USB FX2LP18 responds with a NAK because the particular endpoint buffer memory is not available. The MoBL-USB FX2LP18 firmware framework provides hooks for all the interrupts that it implements. The example project uses ISR_Ep2pingnak and ISR_Ep4pingnak interrupt service routines to handle EP2PING and EP4PING interrupts respectively.

```
void ISR_Ep2pingnak(void) interrupt 0
{
SYNCDELAY; // Re-arm endpoint 2
EP2BCL = 0x80;
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 re-arming 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 to service the EP4PING interrupt in ISR_Ep4pingnak.

8.4.2 Building Firmware Example Code for MoBL-USB FX2LP18 Internal RAM and EEPROM

Click on **Build Target** button at the top right corner of the IDE. The total Code bytes of **pingnak** firmware example is less than 4k code limit Keil uVision2 IDE provided along with the kit. The output of the **Build Target** is **pingnak.hex** and **pingnak.iic** files.

8.4.3 Method to Download Firmware Image to MoBL-USB FX2LP18 Internal RAM and External EEPROM

Refer to section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 and Method to Download Firmware Image to External I2C EEPROM on page 63 and follow similar procedure to download pingnak.hex to RAM memory or **pingnak.iic** to Large EEPROM using CyConsole or CyControlCenter.Both images are located at <Installed_directory>\<Version>\Firmware\pingnak.After downloading, the firmware re-enumerates with PC using its internal VID/PID-0x04B4/0x1004.

8.4.4 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 Cypress generic USB driver cyusb.sys and driver information file cyusbfx2lp18.inf which contains the relevant VID/PID of this example. Follow the procedure outlined in section Binding Cypress USB Driver to MoBL-USB Development Board on page 43 to manually bind the driver using Windows Hardware Wizard. If the binding process is performed for anyone of the firmware example the process can be skipped for this example.

8.4.5 Testing the pingnak Firmware Functionality

Following are the steps to test the pingnak firmware:

1. After the board has re-enumerated, use CyConsole to send 512 bytes from EP2 to EP6.The data received should be same as the data sent. 512 bytes of user-defined data can be sent from the host to Endpoint 2 using CyConsole. For example, select Endpoint 2 OUT in the pipe window near **Bulk Trans** button of **EZ-USB interface** window, enter the length as 512 and HexBytes as 5, and then press the **Bulk Trans** button.
2. This data can be read back from Endpoint 6 using CyConsole. For example, select Endpoint 6 IN in the pipe, enter the length as 512, and then press the Bulk Trans button. 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.
3. The ISRs that handle the PING-NAK interrupt. (ISR_Ep2pingnak and ISR_Ep4pingnak) discards 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. The above can be tested 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.

8.5 Bulkloop Example

8.5.1 Description

This project illustrates the configuration of MoBL-USB FX2LP18 to accept bulk data from the host and loop it back to the host. Click on **bulkloop.Uv2** at <Installed_directory>\<Version>\Firmware\Bulkloop and observe the source code. Four endpoints are configured in **TD_init()** function of **bulkloop.c** to handle bulk transfer: two OUT endpoints and two IN endpoints. The four endpoints defined in the descriptor file have to be configured in this function. This is done by the following statements

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

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

The above 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. 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. This is done by reading the endpoint 2 empty bit in the endpoint status register (EP2468STAT). If endpoint 2 has data (that is sent from the host), the capability of endpoint 6 to receive the data is checked. This is done 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, 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 above statement is executed the auto pointer is incremented. The above statement is executed repeatedly to transfer each byte from endpoint 2 to 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', that is, make the FIFO buffers available to the host for reading data from 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', that is, make the FIFO buffers available to the host for reading data from endpoint 6.

This is accomplished by the following statements:

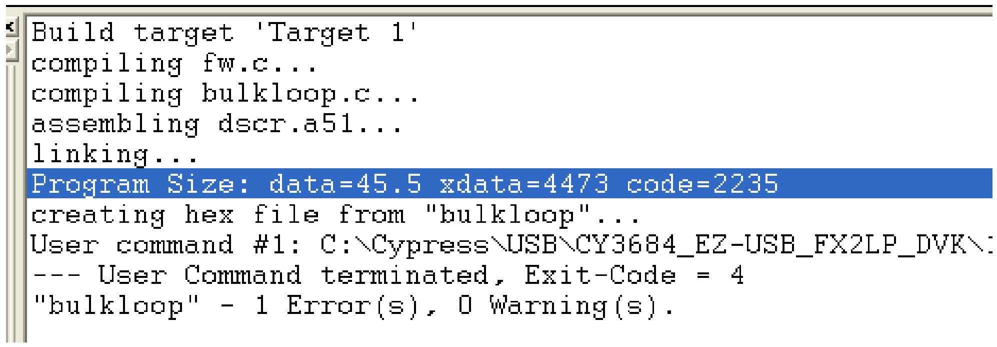
```
EP6BCH = EP2BCH;
SYNCDELAY;
EP6BCL = EP2BCL; // commit EP6IN by specifying the number of bytes the
host can read //from EP6
SYNCDELAY;
EP2BCL = 0x80; // re (arm) EP2OUT
```

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

8.5.2 Building Bulkloop Firmware Example Code for MoBL-USB FX2LP18 Internal and EEPROM

- Click on **Build Target button** at the top right corner of the IDE. Following snapshot of the **Build** window of the Keil IDE shows the successful compilation of the entire project.

  Figure 8-5. Output Window Snapshot of Bulkloop Project Build



```
Build target 'Target 1'
compiling fw.c...
compiling bulkloop.c...
assembling dscr.a51...
linking...
Program Size: data=45.5 xdata=4473 code=2235
creating hex file from "bulkloop"...
User command #1: C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\I
--- User Command terminated, Exit-Code = 4
"bulkloop" - 1 Error(s), 0 Warning(s).
```

  Note: Observe in the above Figure 8-5 the total Code bytes of Bulkloop project is less than the 4k code limit Keil uVision2 IDE provided along with the kit.

- **Firmware output for MoBL-USB FX2LP18 RAM memory:** The output of the **Build Target** is **bulkloop.hex** relevant for downloading to MoBL-USB FX2LP18 RAM memory.
- **Firmware output for external EEPROM:** To generate EEPROM compatible firmware Image the Keil IDE invokes **hex2bix.exe** utility to convert the output file bulkloop.hex into **bulkloop.iic**. Right click on **Target1** in Project Window and select Options for **Target** '**Target1**'.This will pop-up keil settings for this project. Select Output tab and observe at the bottom of IDE the hex2bix utility is invoked under **Run User program#1** section and observe the **hex2bix** utility is invoked in the following manner

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

Refer to application note, “Using the hex2bix Conversion Utility - AN45197” to know more details on the hex2bix utility.

8.5.3 Method to Download Bulkloop Firmware Image to Internal RAM or EEPROM

Refer to section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 and Method to Download Firmware Image to External I2C EEPROM on page 63 and follow similar procedure to download **bulkloop.hex** to RAM memory and **bulkloop.iic** to Large EEPROM using Cyconsole/CyControlCenter.The bulkloop.hex and bulkloop.iic files located at <Installed_directory>\<Version>\Firmware\Bulkloop must be chosen accordingly for MoBL-USB FX2LP18. After downloading, the firmware re-enumerates with PC using its internal VID/PID-0x04B4/0x1004.

8.5.4 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 Cypress generic USB driver cyusb.sys and driver information file *cyusbfx2lp18.inf*, which contains the relevant VID/PID of this example. Follow the procedure outlined in section Binding Cypress USB Driver to MoBL-USB Development Board on page 43 to manually bind the driver using Windows Hardware Wizard. If the binding process is performed for anyone of the previous firmware examples the process can be skipped for this example.

8.5.5 Testing the Bulkloop Firmware Functionality

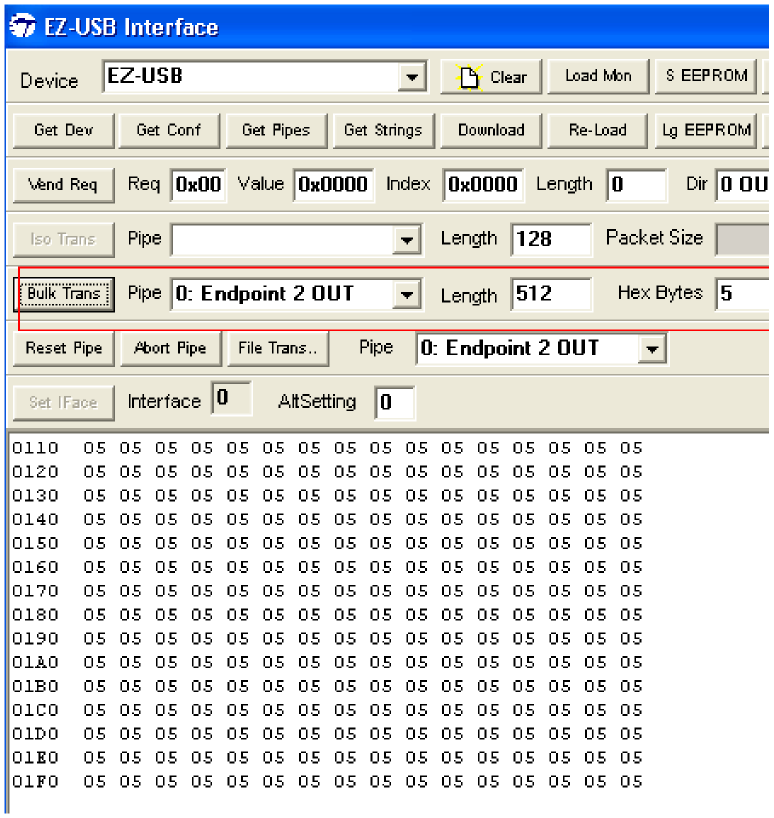
The Bulkloop firmware functionality can be tested using the following applications available in SuiteUSB package.

- Cyconsole
- CyBulk
- Bulkloop

8.5.5.1 Test using Cyconsole PC Application

1. Open Cyconsole PC application from Windows **Start->All Programs->Cypress->Cypress SuiteUSB 3.4.7 -->CyConsole**. Click on **Options->EZ-USB Interface**.This will pop-up EZ-USB Interface Window. Select EP2 OUT as Bulk OUT data transfer pipe adjacent to **BulkTrans** button and enter length as 512 byte with 0x5 as the actual data. Click on **BulkTrans** button. The following Figure 8-6 summarizes the entire operation.

Figure 8-6. EP2 OUT Data Transfer using CyConsole



Device EZ-USB

Req 0x00 Value 0x0000 Index 0x0000 Length 0 Dir 0 OU

Pipe Length 128 Packet Size

Pipe 0: Endpoint 2 OUT Length 512 Hex Bytes 5

Pipe 0: Endpoint 2 OUT

Interface 0 AltSetting 0

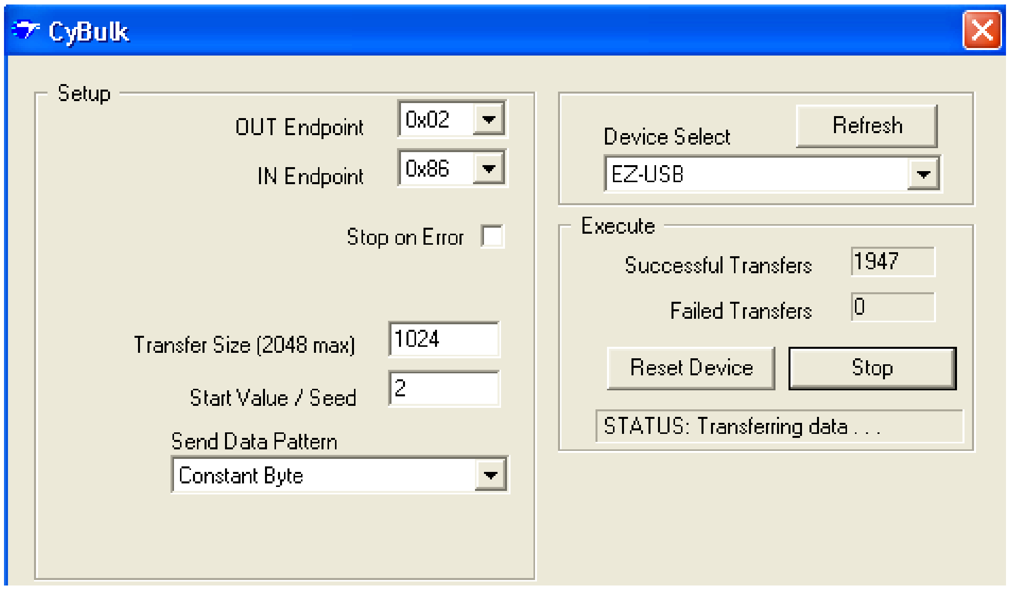
```
0110  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
0120  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
0130  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
0140  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
0150  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
0160  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
0170  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
0180  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
0190  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01A0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01B0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01C0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01D0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01E0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01F0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
```

2. Select EP6 IN as pipe near **BulkTrans** button and enter length as 512 byte. Click on **BulkTrans** button. The data sent on EP2 is loopbacked through EP6. The following figure summarizes the entire operation. The same sequence can be repeated for EP4-OUT and EP8-IN pair.

8.5.5.2 Test using Cybulk Application

The Bulkloop firmware can be tested using this C++ application. For 32-bit Windows OS the CyBulk can be accessed at **C:\Cypress\Cypress Suite USB 3.4.7\CyAPI\examples\cybulk\x86\Release**. The 64-bit version of Cybulk application is located at **C:\Cypress\Cypress Suite USB 3.4.7\CyAPI\examples\cybulk\x64\Release**. Select MoBL-USB FX2LP18 device in the drop down menu and also select anyone the Bulk Endpoint pairs- EP2/EP6 or EP4/EP8. The following Figure 8-7 summarizes the entire operation. Different data patterns of Bulk USB packets can be chosen under **Send Data pattern** and maximum transfer size upto 2048 bytes.

Figure 8-7. Bulkloop using CyBulk Application

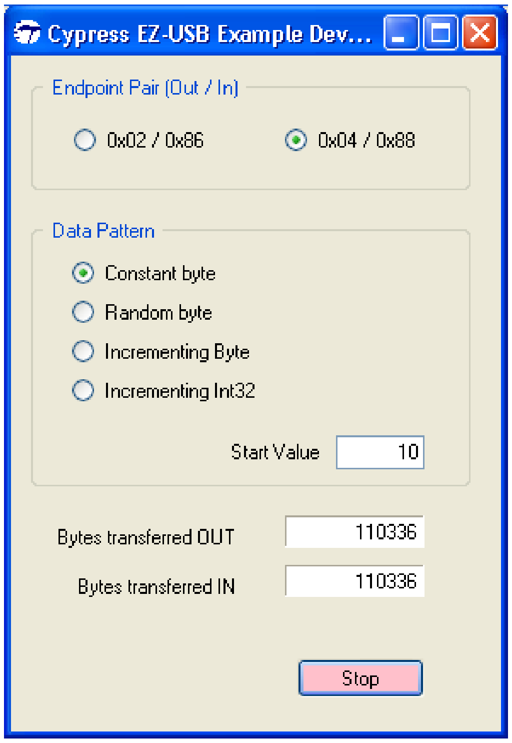


OUT Endpoint 0x02
IN Endpoint 0x86
Stop on Error ☐
Transfer Size (2048 max) 1024
Start Value / Seed 2
Send Data Pattern Constant Byte
Device Select EZ-USB
Successful Transfers 1947
Failed Transfers 0
STATUS: Transferring data . . .

8.5.5.3 *Testing Bulkloop Example using Bulkloop C# .NET Application*

The Bulkloop firmware can be tested using this Bulkloop C# .NET application. The application can be accessed at **Windows Start->All programs->Cypress->Cypress SuiteUSB 3.4.7->Bulkloop**. Select the Bulkloop OUT/IN endpoint pairs EP2/EP6 or EP4/EP8. Click **Start** and observe the no of successful Bulk IN/OUT Transfers

Figure 8-8. Bulkloop using Bulkloop C# Application



Endpoint Pair (Out / In): 0x02 / 0x86 [✓ 0x04 / 0x88]
Data Pattern: [✓ Constant byte] Random byte Incrementing Byte Incrementing Int32
Start Value 10
Bytes transferred OUT 110336
Bytes transferred IN 110336

8.6 Bulksrc Firmware Example

8.6.1 Description

This project illustrates the configuration of MoBL-USB FX2LP18 device to accept bulk data from the host and loop it back to the host. Click on **bulksrc.Uv2** located at <Installed_directory>\<Version>\Firmware\Bulksrc and observe the code. Five endpoints are configured in **TD_init()** function of **bulksrc.c** to handle bulk transfer: Two OUT (EP2/EP4) endpoints and two IN (EP6/EP8) endpoint are double buffered pairs. The fifth endpoint is EP1 which acts as both Bulk IN and OUT endpoint with 64 bytes buffer. These are defined in the descriptor file (**dscr.a51**).The Endpoints are configured in this TD_init function. This is done by the following statements:

```
  EP1OUTCFG = 0xA0;
  EP1INCFG = 0xA0;
  SYNCDELAY;                    // see TRM section 15.14
  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.

```
// 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 above lines arm the two OUT endpoints by skipping two packets of data making the buffers available to receive OUT data.

The IN endpoint EP6 is armed with fixed pattern of data starting with 0x2 irrespective of data sent on 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 TD_poll() function if there is packet content in EP2 then it is re-armed again discarding the current data.

```
// if there is some data in EP2 OUT, re-arm it
   if(!(EP2468STAT & bmEP2EMPTY))
   {
      SYNCDELAY;                //
      EP2BCL = 0x80;
   }
```

Endpoint EP6 is armed again with incremental pattern of data starting with 0x2.

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

The contents received EP4 OUT endpoint are copied to a temporary buffer myBuffer[] and re-armed again.

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

      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 temporary buffer are transferred to a AUTO pointer and finally copied to EP8 IN buffer as shown below.

```
 // if there is room in EP8IN, then copy the contents of the temporary buf-
fer 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
    }
```

8.6.2 Building Bulksrc Firmware Example Code for MoBL-USB FX2LP18 Internal RAM Memory and EEPROM

Click on **Build Target** button at the top right corner of the IDE. The total Code bytes of Bulksrc firmware example is less than 4k code limit Keil uVision2 IDE provided along with the kit. The output of the **Build Target** is **bulksrc.hex** and **bulksrc.iic** files

8.6.3 Method to Download Bulksrc Firmware Image to MoBL-USB FX2LP18 Internal RAM and EEPROM

Refer to section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 and Method to Download Firmware Image to External I2C EEPROM on page 63 and follow similar procedure to download bulksrc.hex to RAM memory and **bulksrc.iic** to Large EEPROM using Cyconsole/CyControlCenter.The **bulksrc.hex** and **bulksrc.iic** files are located at <Installed_directory>\<Version>\Firmware\Bulksrc. After downloading, the firmware re-enumerates with PC using its internal VID/PID-0x04B4/0x1004

8.6.4 Binding Cypress USB Driver for the Downloaded Firmware Image

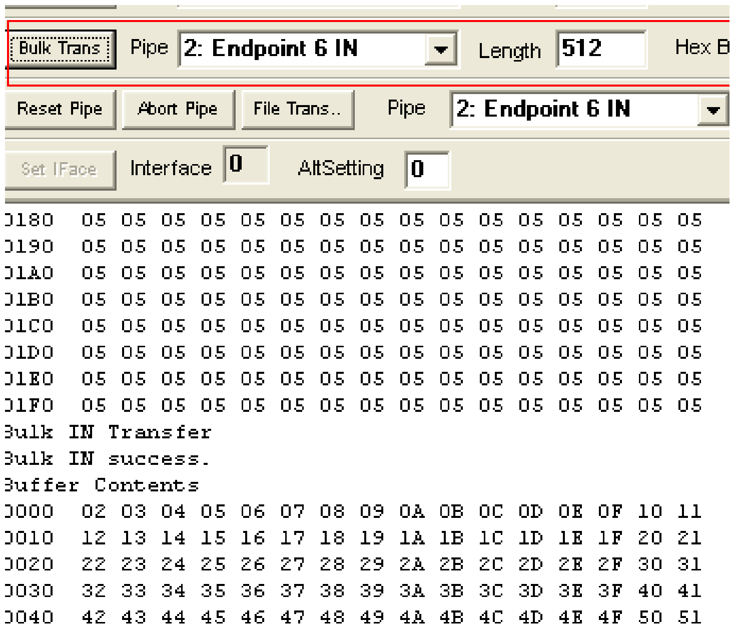
The **Bulksrc** firmware uses vendor class (0xFF) with VID/PID-0x04B4/1004. This example should bind with Cypress generic USB driver cyusb.sys and driver information file *cyusbfx2lp18.inf*, which contains the relevant VID/PID of this example. Follow the procedure outlined in section Binding Cypress USB Driver to MoBL-USB Development Board on page 43 to manually bind the driver using Windows Hardware Wizard. If the binding process is performed for anyone of the previous firmware examples the process can be skipped for this example.

8.6.5 Testing the Bulksrc Firmware Functionality

The Bulksrc firmware functionality can be tested using Cyconsole Utility. Following are the steps

- Open Cyconsole PC application from Windows **Start->All Programs->Cypress->Cypress SuiteUSB 3.4.7->Cyconsole**.
- Click on **Options->EZ-USB Interface**. This will pop-up EZ-USB Interface Window. Select EP2 OUT as pipe and enter length as 512 bytes. Enter sample data as 0x5.Observe the constant pattern 0x5 displayed on the EZ-USB window. Select EP6 IN endpoint with 512 byte length and click on **BulkTrans** button and observe the data with incremental pattern starting with 0x2.

Figure 8-9. Bulk IN Data Transfer on EP6 Endpoint



```
0180  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
0190  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01A0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01B0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01C0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01D0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01E0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
01F0  05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
Bulk IN Transfer
Bulk IN success.
Buffer Contents
0000  02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11
0010  12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21
0020  22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31
0030  32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41
0040  42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51
```

- Select EP4 and EP8 pairs and repeat the same procedure as mentioned above. Observe that the data transferred on EP4 is exactly looped back to EP8.Internally the loopback is performed via a temporary buffer (myBuffer [512]).

8.7 Bulkext Firwmare Example

8.7.1 Description

This example is exactly similar to Bulkloop example. Click on **Bulkext.uv2** project located at <Installed_directory>\<Version>\Firmware\Bulkext and open **bulkext.c** in Keil IDE Project window. The only difference between **Bulkloop** and **Bulkext** example is the source and destination buffer memory address of Bulk endpoint pairs -EP2/EP6 and EP4/EP8. In **Bulkloop** the endpoint FIFO are directly used as source and destination buffers. These are internal RAM buffers residing in MoBL-USB FX2LP18 device. In TD_poll() function the endpoint EP2 data buffer destination is defined as external RAM memory address-0x2800 which is inturn defined as source buffer to EP6 IN endpoint and the data is copied through AUTOPTR mechanism as shown in the following code.

```
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
        // Source is EP2OUT
        APTR1H = MSB( &EP2FIFOBUF );
        APTR1L = LSB( &EP2FIFOBUF );
        // Destination is external RAM (at 0x2800)
        AUTOPTRH2 = 0x28;
        AUTOPTRL2 = 0x00;

        count = (EP2BCH << 8) + EP2BCL;
```

```
            for( i = 0x0000; i < count; i++ )
            {
               EXTAUTODAT2 = EXTAUTODAT1;
            }

            // Source is external RAM
            APTR1H = 0x28;
            APTR1L = 0x00;

            // Destination is EP6IN
            AUTOPTRH2 = MSB( &EP6FIFOBUF );
            AUTOPTRL2 = LSB( &EP6FIFOBUF );

            count = (EP2BCH << 8) + EP2BCL;

            for( i = 0x0000; i < count; i++ )
            {
               EXTAUTODAT2 = EXTAUTODAT1;
            }

            EP6BCH = EP2BCH;
            SYNCDELAY;
            EP6BCL = EP2BCL;        // arm EP6IN
            SYNCDELAY;
            EP2BCL = 0x80;          // re(arm) EP2OUT
         }
      }
```

Similarly endpoint EP4 destination and EP8 source buffer is commonly defined as the external RAM memory-0x2A00.

8.7.2 Building Bulkext fIrmware Example Code for MoBL-USB FX2LP Internal RAM Memory and EEPROM

Click on **Build Target** button at the top right corner of the IDE. The total Code bytes of **Bulkext** firmware example is less than 4k code limit Keil uVision2 IDE provided along with the kit. The output of the **Build Target** is **bulkext.hex** and **bulkext.iic** files.

8.7.3 Method to Download Firmware Image to MoBL-USB FX2LP18 Internal RAM and EEPROM

Refer to section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 and Method to Download Firmware Image to External I2C EEPROM on page 63 and follow similar procedure to download **bulkext.hex** to RAM memory and **bulkext.iic** to Large EEPROM using Cyconsole/CyControlCenter. The **bulkext.hex** and **bulkext.iic** files are located at <Installed_directory>\<Version>\Firmware\Bulkext. After downloading, the firmware re-enumerates with PC using its internal VID/PID-0x04B4/0x1004.

8.7.4 Binding Cypress USB Driver for the Downloaded Firmware Image

The **Bulkext** firmware uses vendor class (0xFF) with VID/PID-0x04B4/1004.This example should bind with Cypress generic USB driver cyusb.sys and driver information file *cyusbfx2lp18.inf* which contains the relevant VID/PID of this example. Follow the procedure outlined in section Binding

Cypress USB Driver to MoBL-USB Development Board on page 43 to manually bind the driver using Windows Hardware Wizard. If the binding process is performed for anyone of the previous firmware examples the process can be skipped for this example.

8.7.5 Testing the Bulkext Firmware Functionality

The example firmware should be tested in a similar manner as **Bulkloop** example using Cyconsole or CyControlCenter.

8.8 EP_Interrupts Example

8.8.1 Description

The **EP_interrupts** example works in a similar manner as Bulkloop on MoBL-USB FX2LP18. The major differences include addition of a 64-byte EP1 as Bulk OUT/IN endpoint to the existing list of 4 endpoints- EP2, EP4, EP6, and EP8.The endpoints are re-armed using their respective interrupt service routines. Following are the interrupts for each of 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()

8.8.2 Building EP_Interrupts Firmware Example Code for MoBL-USB FX2LP Internal RAM and EEPROM

Click on **Build Target** button at the top right corner of the IDE. The total Code bytes of **EP_Interrupts** firmware example is less than 4k code limit Keil uVision2 IDE provided along with the kit. The output of the **Build Target** is **EP_Interrupts.hex** and **EP_Interrupts.iic** files

8.8.3 Method to Program EP_Interrupts Firmware Image to MoBL-USB FX2LP18 Internal RAM and EEPROM

Refer to section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 and Method to Download Firmware Image to External I2C EEPROM on page 63 and follow similar procedure to download **EP_Interrupts.hex** to RAM memory and **EP_Interrupts.iic** to Large EEPROM using Cyconsole/CyControlCenter.The **EP_Interrupts.hex** and **EP_Interrupts.iic** files are located at <Installed_directory>\<Version>\Firmware\EP_Interrupts. After downloading, the firmware re-enumerates with PC using its internal VID/PID-0x04B4/0x1004.

8.8.4 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 Cypress generic USB driver cyusb.sys and driver information file *cyusbfx2lp18.inf*, which contains the relevant VID/PID of this example. Follow the procedure outlined in section Binding Cypress USB Driver to MoBL-USB Development Board on page 43 to manually bind the driver using Windows Hardware Wizard. If the binding process is performed for anyone of the previous firmware examples the process can be skipped for this example.

8.8.5 Testing the EP_Interrupts Firmware Functionality

The code example should be tested in a similar manner as **Bulkloop** example.The Bulk data transfers on EP1 are tested with length of 64 bytes and 512 bytes for the EP2, EP4, EP6 and EP8. The process is similar to the one outlined in section Testing the Bulkloop Firmware Functionality on page 74.

8.9 iMemtest Firmware Example

This example does a data integrity check by writing and reading back the data on different memories inside MoBL-USB FX2LP18 device such as GPIF waveform memory(0xE400), Endpoint buffer memories(0xE740, 0xF000, and so on) and avoid the range where the firmware is located. If the written data and read data match then “good” is displayed on Seven segment display-U9 and if there are errors at any specific memory location the corresponding location is displayed. The example is compiled using the Keil IDE similar to previous examples and corresponding images for RAM (**imemtest.hex**) and EEPROM (**imemtest.iic**) can be generated. Both the images are located at <Installed_directory>\<version\iMemtest. After downloading the images for RAM(iMemtest.hex) or EEPROM(iMemtest.iic) using the process outlined in section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 observe the seven segment display for either “GOOD” string displayed or the exact location of memory write/read failure.

8.10 Dev_IO Firmware Example

This example is used to test the connectivity of seven segment display and the Push button switches (f2,f3) w.r.t MoBL-USB FX2LP18 device. The Seven segment display (U9) and Push buttons are connected to Philips PCF8574 I/O expanders (U8 and U10). The example is compiled using the Keil IDE similar to previous examples and corresponding images for RAM (**Dev_IO.hex**) and EEPROM (**Dev_IO.iic**) can be generated. Both the images are located at <Installed_directory>\<version\dev_io. After downloading the images using the process outlined in section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 and Method to Download Firmware Image to External I2C EEPROM on page 63 press F2 push button and observe decrement values in the range 0xF-0x0.Similarly pressing F3 will increment the values in the range 0x0-0xF starting from the current value. Observe the Seven segment displaying the values for each button press.

8.11 extr_intr Firmware Example

This example is used to demonstrate the use of external interrupts INT0 and INT1. The relevant interrupt service routines (ISR) for each of these external interrupts were provided in **isr.c**. The following Table 8-3 lists the registers and associated pins for each of these interrupts:

Table 8-3. External Interrupts and Register Definitions in MoBL-USB FX2LP18 Device

Interrupt	**Interrupt Enable**	**Interrupt Pin**	**Priority Control**	**Natural Priority**	**Interrupt Request Flag**	**Interrupt Type**	**Interrupt type Controlling Bit**
INT0	IE.0	PA.0	IP.0	1	TCON.1	Level or Edge sensitive, active low	[TCON.0]
INT1	IE.2	PA.1	IP.2	3	TCON.3	Level or Edge, sensitive active low	[TCON.2]

Refer to the chapter 4 'Interrupts' in the MoBL-USB Technical Reference Manual

Notes:

1. IE, EIE, IP, EIP, TCON, EXIF, and EICON are all SFRs. For a description of these SFRs, refer to the MoBL-USB Technical Reference Manual
2. Active low interrupts are falling edge triggered and active high interrupts are rising edge triggered. In the example, the below register configurations are done in "extr_int.c" for setting up the interrupts:

```
//INT0 and INT1
PORTACFG = 0x03; // PA0 and PA1 are pins for INT0 and INT1 respectively.
TCON |= 0x05; // INT0 and INT1 are configured as Edge triggered inter-
rupts.
```

The Interrupt service routines for each of these external interrupts are defined in "isr.c". These routines clear the interrupt and toggle the relevant Port pin and anyone of the LED D2-D5.

```
void ISR_EXTR0(void) interrupt 0

{

  TCON &= 0xFD;         // Clear INT0  TCON.1 Flag

  // Yes .  You do not need to clear this flag as it is

  // cleared by hardware

  IOB ^= 0x01;    // Toggle pin 19-P1 header of PortB

}
```

The example is compiled using the Keil IDE similar to previous examples and corresponding images for RAM (extr_intr.hex) and EEPROM (extr_intr.iic) can be generated. Both the images are located at <Installed_directory>\<version\extr_intr. Download the images using the process outlined in section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 and Method to Download Firmware Image to External I2C EEPROM on page 63.

8.11.1 Testing the Example

The function generator can be set to generate a square wave of known frequency(Use low frequency like 100hz-1Khz).Connect the input to 20 pin header P2(19th pin) which is PA.0 for triggering INT0 interrupt. Similarly feed the square waveform to P2(18th pin) which is PA.1.This triggers INT1 interrupt. The corresponding ISR for each of these interrupts get triggered and PORTB pins on P1 header P1-pin 19(PB.0 for INT0) and P1-pin18(PB1 for INT1) generate square wave form. The square waveform can be observed using an oscilloscope.

8.12 Vend_ax Example

This example demonstrates the use of different vendor commands. Vendor commands are used to accomplish unique tasks like MoBL-USB reset, RAM download, setting different frequency for I2C interface, etc. The vendor commands are defined in vend_ax.c source file of the example. Open the project by clicking on vend_ax.uv2.

Located at <Installed_directory>\<version\Firmware/vend_ax and observe vendor commands implemented in the C routine - DR_VendorCmnd (void) .Following are the vendor commands defined in "vend_ax.c" file:

Table 8-4. Vendor Command Definitions in vend_ax Example

S.No	Vendor Command/Macro Definition	Function
1	0xA2/VR_EEPROM	Downloads data to a small EEPROM
2	0xA3/ VR_RAM	Downloads data to internal RAM memory
3	0xA6/ VR_GET_CHIP_REV	The command retrieves the current revision of MOBL-USB FX2LP18 IC
4	0xA8/VR_RENUM	The MoBL-USB FX2LP18 device disconnects and re-connect again.
5	0xA9/VR_DB_FX	The Commands selects double byte addressed large EEPROM-U7 and the contents can be uploaded or downloaded to EEPROM
6	0xAA/VR_I2C_100	Sets the I2C interface to 100 kHz
7	0xAB/VR_I2C_400	Sets the I2C interface to 400 kHz

The example is compiled using the Keil IDE similar to previous examples and corresponding images for RAM (vend_ax.hex) and EEPROM (vend_ax.iic) can be generated. Both the images are located at <Installed_directory>\<version>\vend_ax.Using CyConsole/CyControlCenter the images can be downloaded as outlined in section Method to Download Firmware Image to MoBL-USB Internal RAM Memory on page 62 and Method to Download Firmware Image to External I2C EEPROM on page 63.

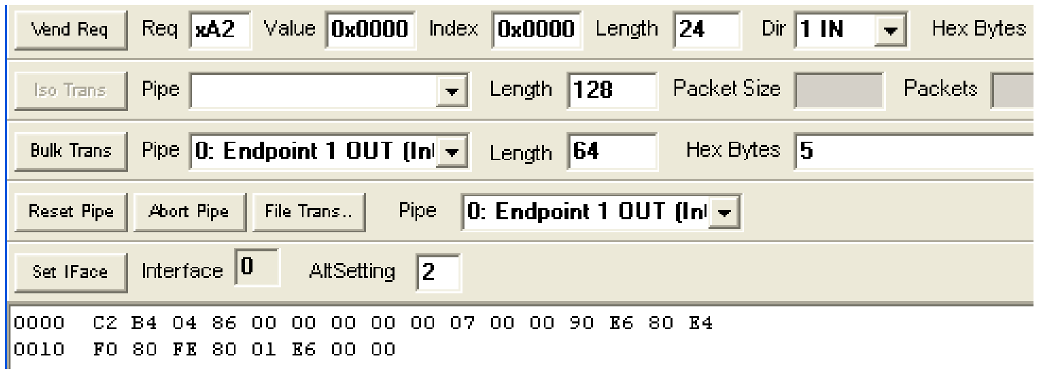
8.12.1 Testing the vend_ax Example

1. **0xA2 command-Read/Write to EEPROM**
   As mentioned in table this command is used to read and write contents to small EEPROM. Short pins 2-3 of JP7 to enable Writes to small EEPROM-U8.Before Writing to EEPROM using 0xA2 command set SW2-LG-SM and SW1-SMALL EEPROM..

   a. Test using CyConsole

   To read the contents of small EEPROM select Req=0xA2 wValue=0 wIndex=0 Length = 24 bytes (data to read) and direction IN for reading the data on control endpoint. Click on **Vend Req**. Following figure summarizes the entire operation

Figure 8-10. A2 Vendor Command Read Operation using Cyconsole



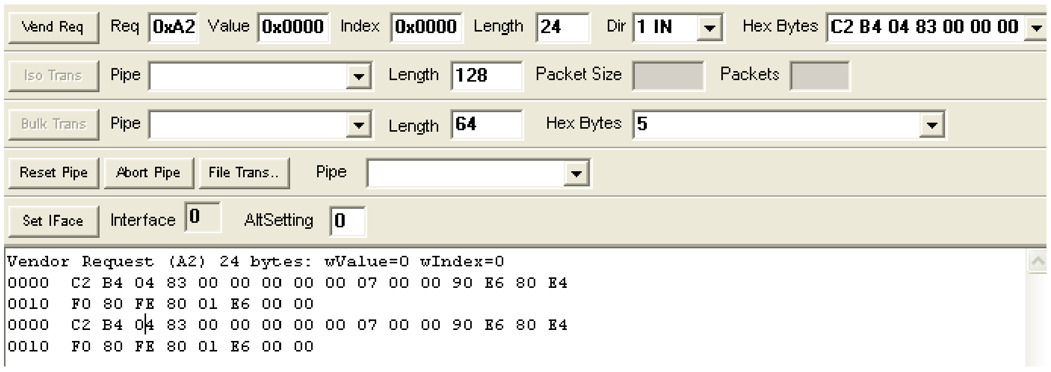
Req xA2 Value 0x0000 Index 0x0000 Length 24 Dir 1 IN Hex Bytes
Pipe Length 128 Packet Size Packets
Pipe 0: Endpoint 1 OUT (Int Length 64 Hex Bytes 5
Pipe 0: Endpoint 1 OUT (Int
Interface 0 AltSetting 2

```
0000  C2 B4 04 86 00 00 00 00 00 07 00 00 90 E6 80 E4
0010  F0 80 FE 80 01 E6 00 00
```

Observe the contents read from the small EEPROM

To write the contents to small EEPROM select Req=0xA2 wValue=0 wIndex=0 Length = 24 bytes (data to read) and direction OUT for sending data on control endpoint. The small EEPROM content first valid byte is always 0xC2 and additional bytes contain new VID/PID information.Type the **"c2 b4 04 83 00 00 00 00 00 07 00 00 90 e6 80 e4 f0 80 fe 80 01 e6 00 00"** in Hex Bytes box. Click on **Vend Req**. Following figure summarizes the entire operation. To verify the written data read the data in IN direction and observe the same data as shown in following Figure 8-11.

Figure 8-11. A2 Vendor Command Write Operation using Cyconsole


Vend Req | Req 0xA2 | Value 0x0000 | Index 0x0000 | Length 24 | Dir 1 IN | Hex Bytes C2 B4 04 83 00 00 00

Iso Trans | Pipe | Length 128 | Packet Size | Packets

Bulk Trans | Pipe | Length 64 | Hex Bytes 5

Reset Pipe | Abort Pipe | File Trans.. | Pipe

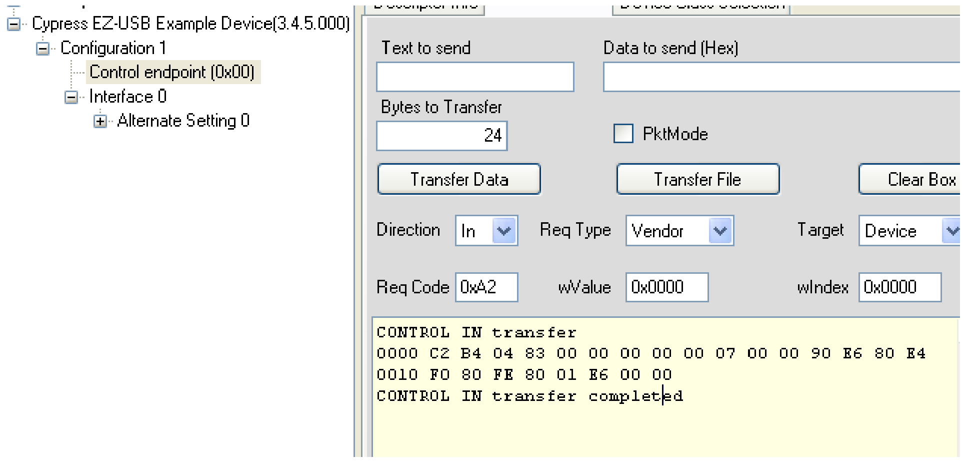
Set IFace | Interface 0 | AltSetting 0

```
Vendor Request (A2) 24 bytes: wValue=0 wIndex=0
0000  C2 B4 04 83 00 00 00 00 00 07 00 00 90 E6 80 E4
0010  F0 80 FE 80 01 E6 00 00
0000  C2 B4 04 83 00 00 00 00 00 07 00 00 90 E6 80 E4
0010  F0 80 FE 80 01 E6 00 00
```

b. **Test using CyControlCenter**

To read the contents of small EEPROM select Direction=In Req Type=vendor Target=Device Bytes to Transfer=24 bytes (data to read) and Req Code=0xA2 for reading the data on control endpoint. Click on **Transfer Data** button. Figure 8-12 summarizes the entire operation.

Figure 8-12. A2 Vendor Command Read Operation using CyControlCenter



Bytes to Transfer: 24

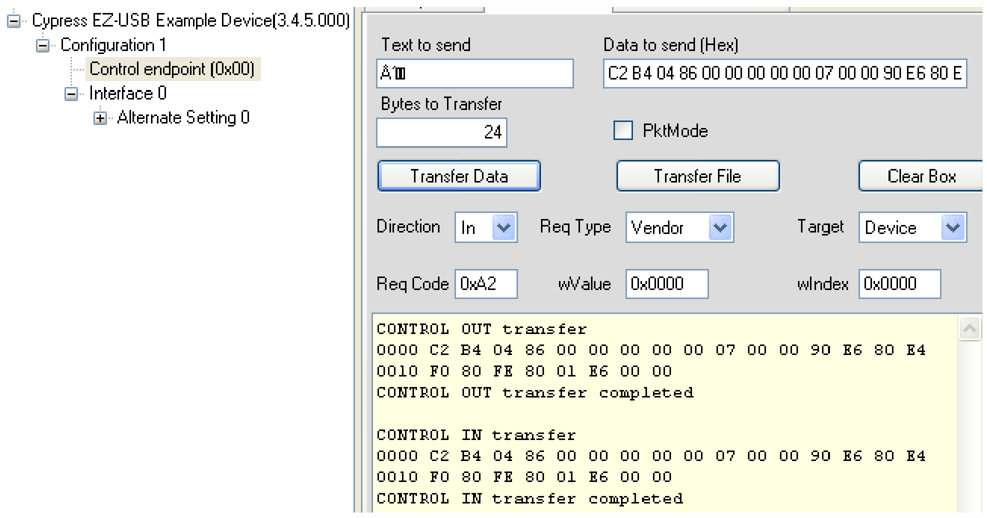
Direction: In | Req Type: Vendor | Target: Device

Req Code: 0xA2 | wValue: 0x0000 | wIndex: 0x0000

```
CONTROL IN transfer
0000 C2 B4 04 83 00 00 00 00 00 07 00 00 90 E6 80 E4
0010 F0 80 FE 80 01 E6 00 00
CONTROL IN transfer completed
```

To write the contents to small EEPROM select Direction=OUT Req Type=vendor Target=Device Bytes to Transfer=24 bytes (data to read) and Req Code=0xA2 and enter data to send-**c2 b4 04 84 00 00 00 00 00 07 00 00 90 e6 80 e4 f0 80 fe 80 01 e6 00 00 in Data to send** box. Click on **Transfer Data** button and observe the EEPROM getting programmed. To verify set Direction =In and click Transfer Data button Figure 8-13 summarizes the entire operation. Figure 8-13 summarizes the entire operation

Figure 8-13. A2 Vendor Command Write Operation using CyControlCenter



Cypress EZ-USB Example Device(3.4.5.000)
Configuration 1
Control endpoint (0x00)
Interface 0
Alternate Setting 0

Text to send: A⅏
Data to send (Hex): C2 B4 04 86 00 00 00 00 00 07 00 00 90 E6 80 E
Bytes to Transfer: 24
PktMode
Transfer Data | Transfer File | Clear Box
Direction: In | Req Type: Vendor | Target: Device
Req Code: 0xA2 | wValue: 0x0000 | wIndex: 0x0000

```
CONTROL OUT transfer
0000 C2 B4 04 86 00 00 00 00 00 07 00 00 90 E6 80 E4
0010 F0 80 FE 80 01 E6 00 00
CONTROL OUT transfer completed

CONTROL IN transfer
0000 C2 B4 04 86 00 00 00 00 00 07 00 00 90 E6 80 E4
0010 F0 80 FE 80 01 E6 00 00
CONTROL IN transfer completed
```

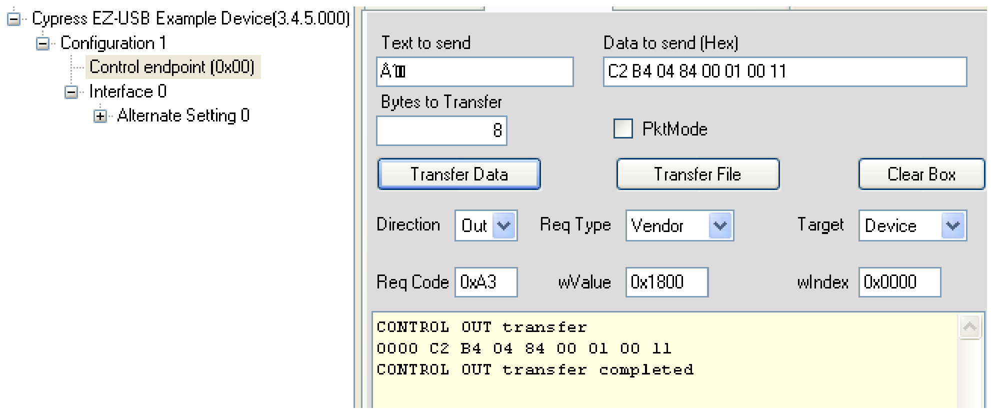
2. **0xA3 command-Download data to RAM memory**

   This command is used to download data to MoBL-USB FX2LP18 internal (0x0000-0x3FFFF) RAM.

   a. **Test using CyControlCenter**

   To write the contents to RAM memory select Direction=OUT Req Type=vendor Target=Device Bytes to Transfer=8 bytes (data to read) wValue=0x1800(RAM address), Req Code=0xA3 and enter data to send. For e.g: "**C2 B4 04 84 00 01 00 11" in Data to send** box. Click on **Transfer Data** button and observe the RAM memory getting programmed. Figure 8-14 summarizes the entire operation.

Figure 8-14. A3 Vendor Command Read Operation using CyControlCenter



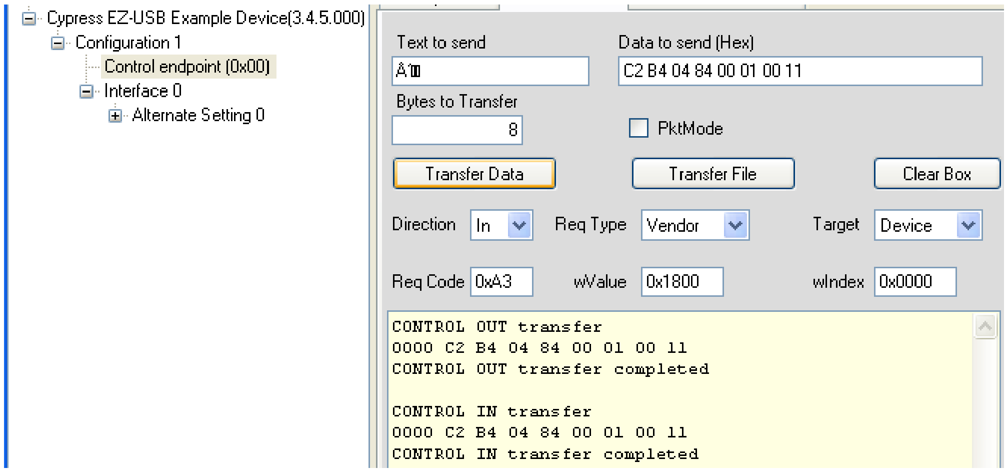
Cypress EZ-USB Example Device(3.4.5.000)
Configuration 1
Control endpoint (0x00)
Interface 0
Alternate Setting 0

Text to send: A⅏
Data to send (Hex): C2 B4 04 84 00 01 00 11
Bytes to Transfer: 8
PktMode
Transfer Data | Transfer File | Clear Box
Direction: Out | Req Type: Vendor | Target: Device
Req Code: 0xA3 | wValue: 0x1800 | wIndex: 0x0000

```
CONTROL OUT transfer
0000 C2 B4 04 84 00 01 00 11
CONTROL OUT transfer completed
```

To read the contents from RAM memory select Direction=IN Req Type=vendor Target=Device Bytes to Transfer=8 bytes (data to read) and Req Code=0xA3. Click on **Transfer Data** button and observe the RAM memory written previously matches with the read data. Figure 8-15 summa-rizes the entire operation.

Figure 8-15. A3 vendor command read operation using CyControlCenter



Text to send: A'
Data to send (Hex): C2 B4 04 84 00 01 00 11
Bytes to Transfer: 8
PktMode: ☐
Direction: In
Req Type: Vendor
Target: Device
Req Code: 0xA3
wValue: 0x1800
wIndex: 0x0000

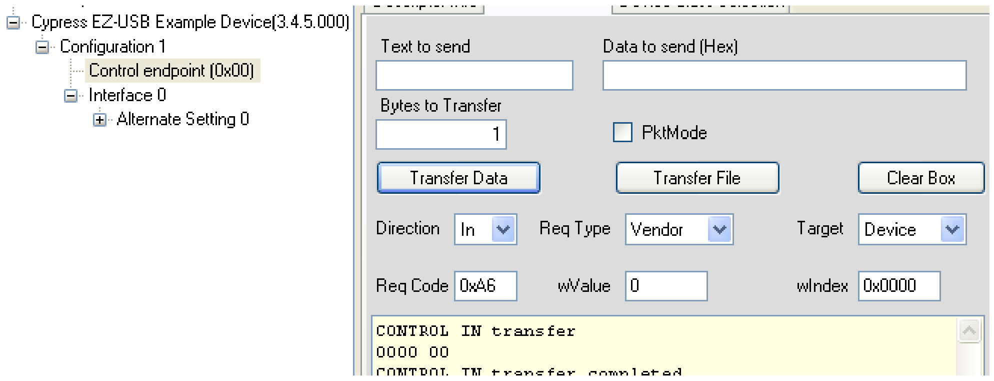
```
CONTROL OUT transfer
0000 C2 B4 04 84 00 01 00 11
CONTROL OUT transfer completed

CONTROL IN transfer
0000 C2 B4 04 84 00 01 00 11
CONTROL IN transfer completed
```

3. **0xA6 command-Get Chip Revision**

   To retrieve the current revision of the MoBL-USB FX2LP18) device this command is used. Figure 8-16 of CyControlCenter summarizes the entire operation.

Figure 8-16. A6 Vendor Command using CyControlCenter



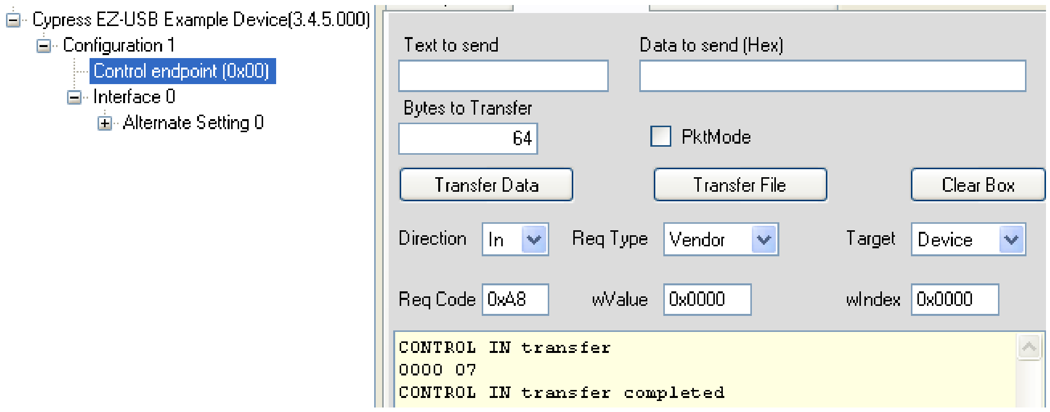
Text to send:
Data to send (Hex):
Bytes to Transfer: 1
PktMode: ☐
Direction: In
Req Type: Vendor
Target: Device
Req Code: 0xA6
wValue: 0
wIndex: 0x0000

```
CONTROL IN transfer
0000 00
CONTROL IN transfer completed
```

4. **0xA8 command-Disconnect and re-connect MoBL-USB FX2LP18**

   This command is used to disconnect and re-connect the MoBL-USB FX2LP18 IC using CPUCS register. The MoBL-USB FX2LP18 re-enumerates again. Observe Cypress device disappearing from CyControlCenter window and re-appearing back in the same window. Figure 8-17 summarizes the command trigger using CyControlCenter.

Figure 8-17. A8 Vendor Command Operation using CyControlCenter



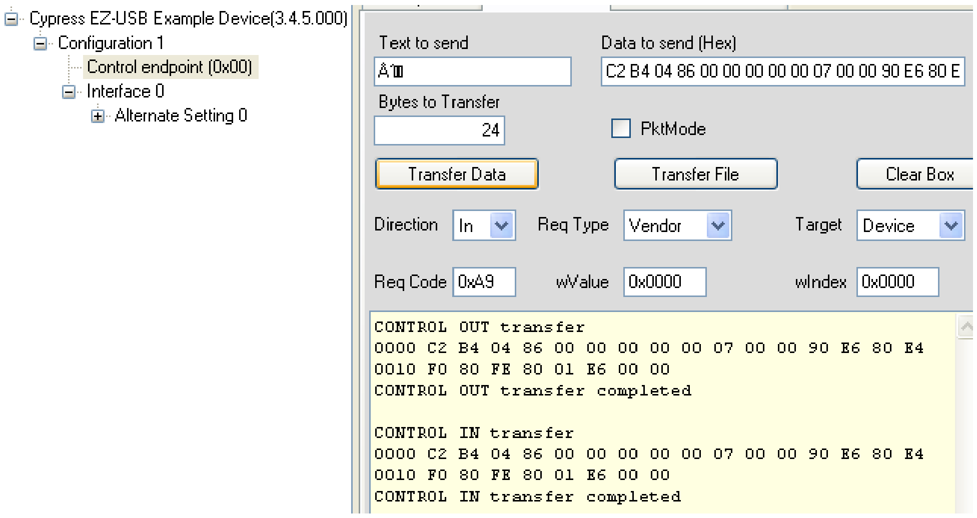
Text to send:
Data to send (Hex):
Bytes to Transfer: 64
PktMode: ☐
Direction: In
Req Type: Vendor
Target: Device
Req Code: 0xA8
wValue: 0x0000
wIndex: 0x0000

```
CONTROL IN transfer
0000 07
CONTROL IN transfer completed
```

5. **0xA9 command- Read/Write Large EEPROM**

   The Large EEPROM U7 on MoBL-USB FX2LP18 can be Written and Read using 0XA9 command. A sample firmware Image LP18_dvk.iic located at <Installed_directory>\<Version>\Firmware\Connect is used as sample data. Before sending 0xA9 command switch SW2-LG-SM and SW1-LARGE EEPROM settings should be set.Short 2-3 pins of JP7 to enable writes on EEPROM.Copy the content of LP18_dvk.iic to **Data to Send** box.Select OUT direction, Req Code =0xA9 Req Type=vendor. Click **Transfer Data button**. Similarly read back the data in IN direction. Following Figure 8-18 summarizes the entire operation.

   Figure 8-18. A9 Vendor Command Operation using CyControlCenter



Text to send: A
Data to send (Hex): C2 B4 04 86 00 00 00 00 00 07 00 00 90 E6 80 E
Bytes to Transfer: 24
PktMode: ☐
Direction: In
Req Type: Vendor
Target: Device
Req Code: 0xA9
wValue: 0x0000
wIndex: 0x0000

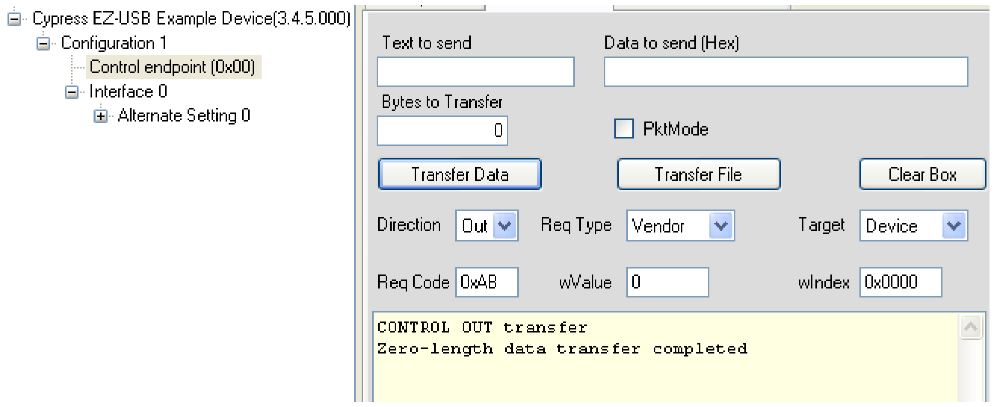
```
CONTROL OUT transfer
0000 C2 B4 04 86 00 00 00 00 00 07 00 00 90 E6 80 E4
0010 F0 80 FE 80 01 E6 00 00
CONTROL OUT transfer completed

CONTROL IN transfer
0000 C2 B4 04 86 00 00 00 00 00 07 00 00 90 E6 80 E4
0010 F0 80 FE 80 01 E6 00 00
CONTROL IN transfer completed
```

6. **0xAA/0xAB-Setting I2C interface frequency**

   Using this command the I2C interface frequency can be set to 100 kHz or 400 kHz. Figure 8-19 summarizes the command trigger using CyControlCenter.

Figure 8-19. AA/AB Vendor Command Operation using CyControlCenter



Bytes to Transfer: 0
Direction: Out
Req Type: Vendor
Target: Device
Req Code: 0xAB
wValue: 0
wIndex: 0x0000

```
CONTROL OUT transfer
Zero-length data transfer completed
```

8.13 Debugging Using Keil Monitor Program

The Keil uVision2 IDE supplied with the kit enables user to debug the firmware examples provided with the kit. Using Keil debug monitor program and UART port-SIO-1 on CY3687 MoBL-USB FX2LP18 development board the firmware examples are debugged. Following is the procedure to debug the firmware using Keil IDE.

1. The MoBL-USB FX2LP18 development board contains small EEPROM (U8) which is by default programmed with 0xC2 image .It contains relevant VID/PID (0x04B4/0x0086) for Keil debug monitor download. If the image does not exist due to programming any of the images defined in previous sections follow the process outlined in section Method to Download Firmware Image to External I2C EEPROM on page 63 to download LP18_dvk.iic. The image is located at <Installed_directory>\<Version>\Firmware\EEPROM Images.
2. Switch SW2-LG-SM SW2-SMALL EEPROM before downloading the image to small EEPROM-U8.After download press RESET. Connect a UART cable between SIO-1 and Windows PC.
3. Follow the process outlined in section Binding Cypress USB Driver to MoBL-USB Development Board on page 43 to bind CyMonfx2lp18 driver package at <Installed_directory>\<Version>\Drivers\CyMonfx2lp18.The driver files for relevant Windows OS version of the Host PC can be chosen respect to the above path as follows:
   a. Windows2000: w2k\x86
   b. Windows XP (32-bit): wxp\x86
   c. Windows XP (64-bit): wxp\x64
   d. Windows-Vista (32-bit): wlh-vista\x86
   e. Windows-Vista (64-bit): wlh-vista\x64
   f. Windows-7(32-bit): wlh-win7\x86
   g. Windows-7(64-bit): wlh-win7\x64
4. Observe the green BKPT/monitor light D5 glowing GREEN on the development board. Observe in the device manager the MoBL-USB FX2LP18 devices are listed as: **“Cypress MoBL-USB FX2LP18 Board Keil monitor(3.4.5.000)”**
5. The MoBL-USB device re-enumerates with VID/PID - 0x04B4/0x0086
6. The Keil debug monitor (.hex) is already recorded in a script file -mon_i2c.spt. The procedure to generate a script file for corresponding .hex file was explained in section 6.3.1. Sample monitor .hex files are located at <Installed_directory>\<Version>\Target\Monitor. Following are sample monitor files in the MoBL-USB FX2LP18 development kit.

a. mon-int-sio1.hex: This keil debug monitor file resides in internal MoBL-USB RAM memory and communicates through SIO-1 UART port.
b. mon-int-sio1.spt: This keil debug monitor script file resides in internal MoBL-USB RAM memory and communicates through SIO-1 UART port.

Copy latest version Keil debug Monitor DLL (ver 2.40) located at <Installed_directory>\<version>\Target\Monitor\mon51 to C:\Keil\C51\BIN.The existing DLL can be saved elsewhere if required. The latest Mon51.dll supports debugging on this CY3687 MoBL-USB FX2LP18 development board.

7. Open Keil uVision2 IDE by selecting Start > Programs > Keil uVision2. Open the dev_io project file at <Installed_directory>\<Version>\Firmware\dev_io, as shown in Figure 8-20 and Figure 8-21.

Figure 8-20. Opening Project File using Keil uVision2 IDE

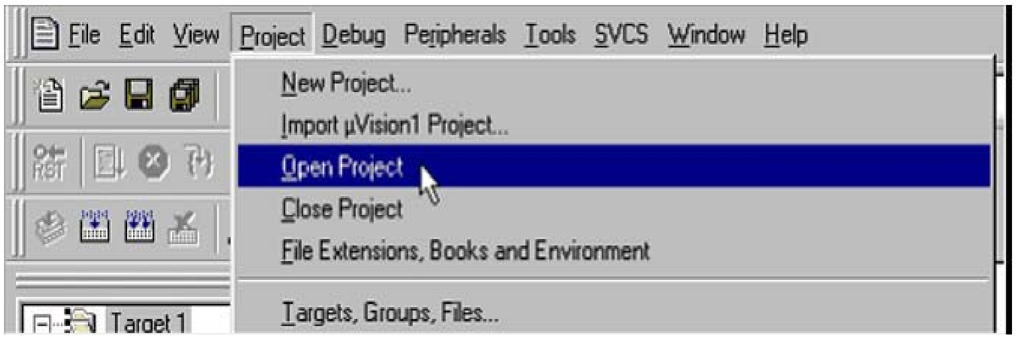
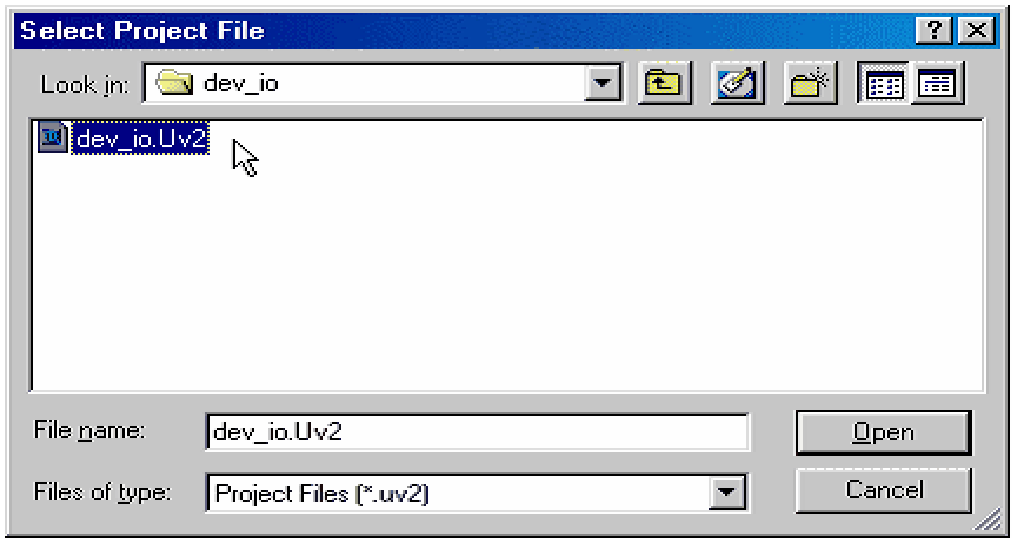


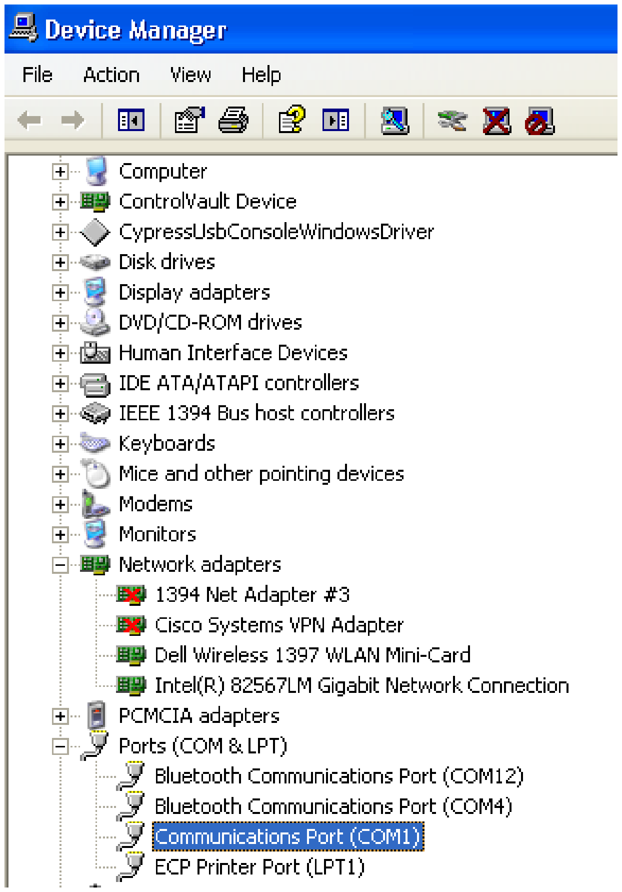
Figure 8-21. Selecting Project File using Keil uVision2 IDE



8. Make sure you are using the correct serial port and that the baud rate is set correctly. To do this, select **Project > Options for Target 'Target 1' > Debug > Settings**.

Note: Your PC may have a single serial port. In this case, use relevant COM port by checking under Ports (COM and LPT) in Device Manager (Type **devmgmt.msc** in Windows **Start->Run**) as shown in Figure 8-22. Check the box labeled Serial Interrupt.

Figure 8-22. Serial Ports List in Device Manager Window


Device Manager
File Action View Help
Computer
ControlVault Device
CypressUsbConsoleWindowsDriver
Disk drives
Display adapters
DVD/CD-ROM drives
Human Interface Devices
IDE ATA/ATAPI controllers
IEEE 1394 Bus host controllers
Keyboards
Mice and other pointing devices
Modems
Monitors
Network adapters
1394 Net Adapter #3
Cisco Systems VPN Adapter
Dell Wireless 1397 WLAN Mini-Card
Intel(R) 82567LM Gigabit Network Connection
PCMCIA adapters
Ports (COM & LPT)
Bluetooth Communications Port (COM12)
Bluetooth Communications Port (COM4)
Communications Port (COM1)
ECP Printer Port (LPT1)

9. Click on **Project->Options for Target 'Target1'** in Keil uVision2 IDE and select Debug Tab in the new pop-up window as shown in Figure 8-23 and Figure 8-24.

Figure 8-23. Project Options in Keil uVision2 IDE

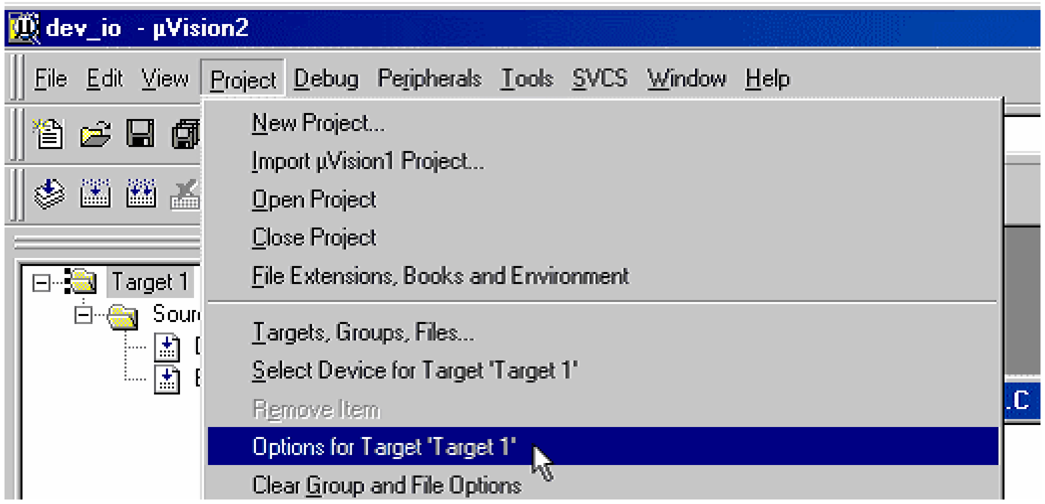
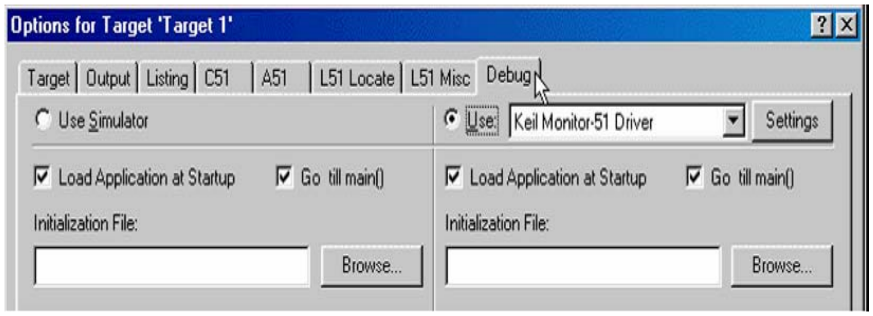

dev_io - µVision2
File Edit View Project Debug Peripherals Tools SVCS Window Help
New Project...
Import µVision1 Project...
Open Project
Close Project
File Extensions, Books and Environment
Targets, Groups, Files...
Select Device for Target 'Target 1'
Remove Item
Options for Target 'Target 1'
Clear Group and File Options
Target 1
Sour

Figure 8-24. Debug Tab window in Project options



Options for Target 'Target 1'

Target | Output | Listing | C51 | A51 | L51 Locate | L51 Misc | Debug

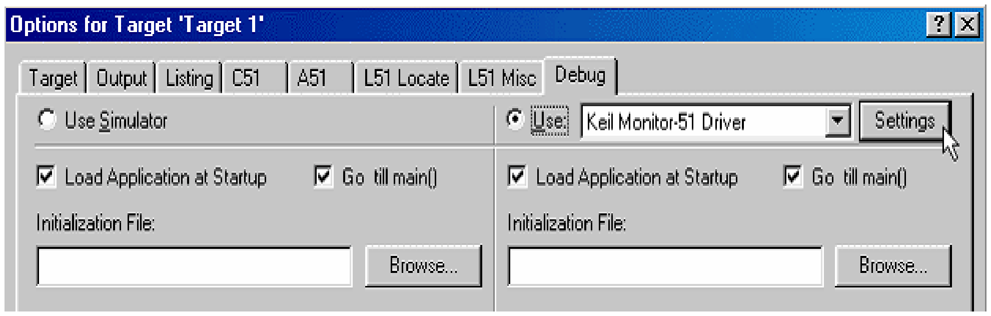
( ) Use Simulator — (•) Use: Keil Monitor-51 Driver — Settings

[✓] Load Application at Startup [✓] Go till main() — [✓] Load Application at Startup [✓] Go till main()

Initialization File: Browse... — Initialization File: Browse...

10. Select Settings under **Keil Monitor-51 Driver** and select relevant COM port for the UART cable connected to SIO-1 port as shown in Figure 8-25 and Figure 8-26.

Figure 8-25. Settings Button for Keil Debug Monitor



Options for Target 'Target 1'

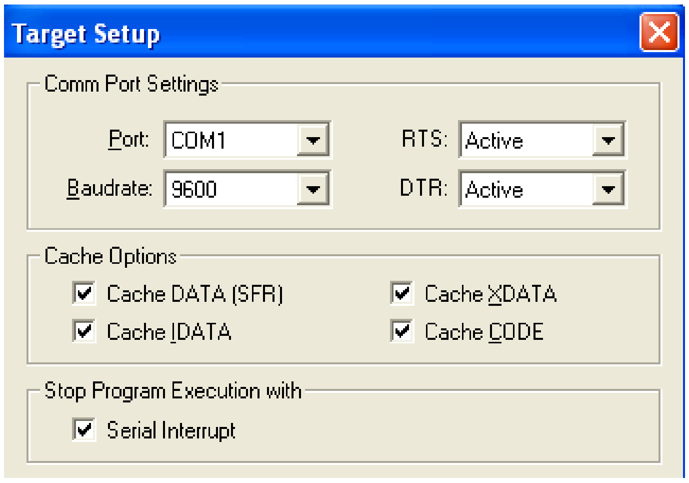
Target | Output | Listing | C51 | A51 | L51 Locate | L51 Misc | Debug

( ) Use Simulator — (•) Use: Keil Monitor-51 Driver — Settings

[✓] Load Application at Startup [✓] Go till main() — [✓] Load Application at Startup [✓] Go till main()

Initialization File: Browse... — Initialization File: Browse...

Figure 8-26. UART Settings for Keil Debug Monitor



Target Setup

Comm Port Settings

Port: COM1 — RTS: Active

Baudrate: 9600 — DTR: Active

Cache Options

[✓] Cache DATA (SFR) [✓] Cache XDATA

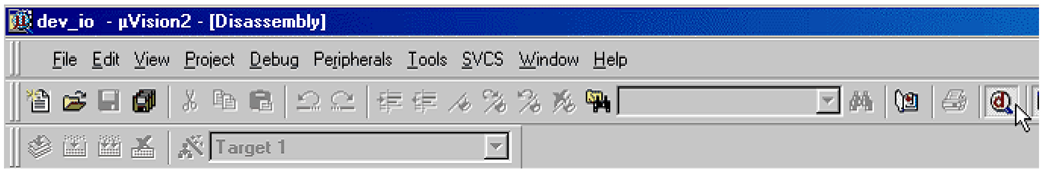
[✓] Cache IDATA [✓] Cache CODE

Stop Program Execution with

[✓] Serial Interrupt

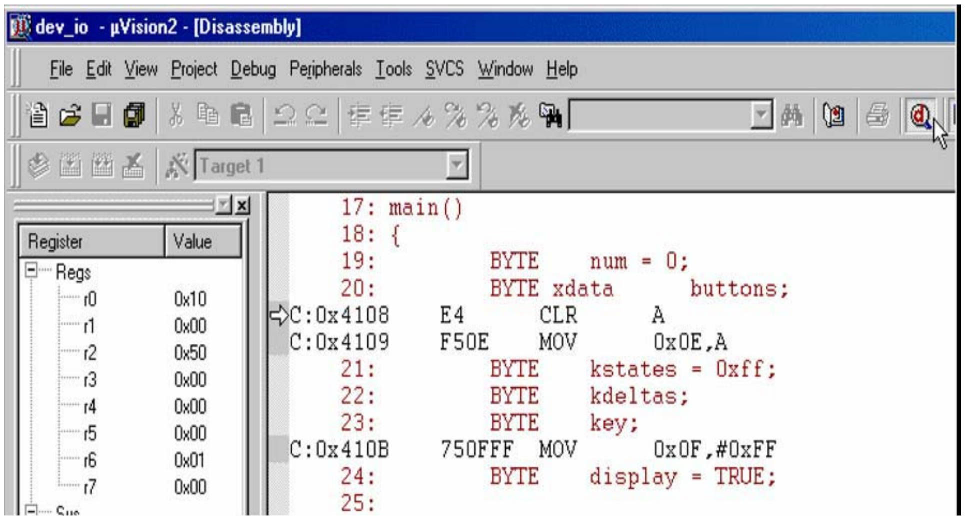
11. Click **OK** to close **Target set up** Window and then close the **Options for Target "Target 1"** window.
12. Select the **Start/Stop Debug Session** button on the Keil IDE as shown in the Figure 8-27.

Figure 8-27. Debug Session Trigger in Keil uVision2 IDE


dev_io - µVision2 - [Disassembly]
File Edit View Project Debug Peripherals Tools SVCS Window Help
Target 1

13. The IDE switches to Debug mode is seen; a yellow arrow indicates the Program Counter location in the Disassembly window of dev_io project.

Figure 8-28. Disassembly View of dev_io.c file in Keil uVision2 IDE

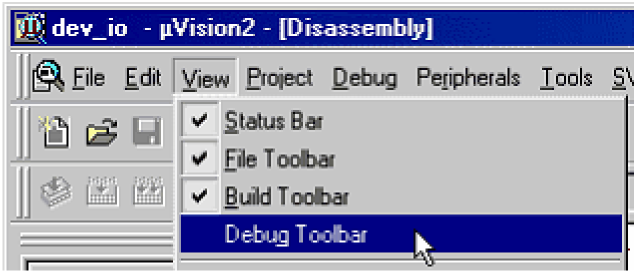

dev_io - µVision2 - [Disassembly]
File Edit View Project Debug Peripherals Tools SVCS Window Help
Target 1

Register	Value
Regs	
r0	0x10
r1	0x00
r2	0x50
r3	0x00
r4	0x00
r5	0x00
r6	0x01
r7	0x00

```
    17: main()
    18: {
    19:         BYTE    num = 0;
    20:         BYTE xdata      buttons;
C:0x4108    E4      CLR      A
C:0x4109    F50E    MOV      0x0E,A
    21:         BYTE    kstates = 0xff;
    22:         BYTE    kdeltas;
    23:         BYTE    key;
C:0x410B    750FFF  MOV      0x0F,#0xFF
    24:         BYTE    display = TRUE;
    25:
```

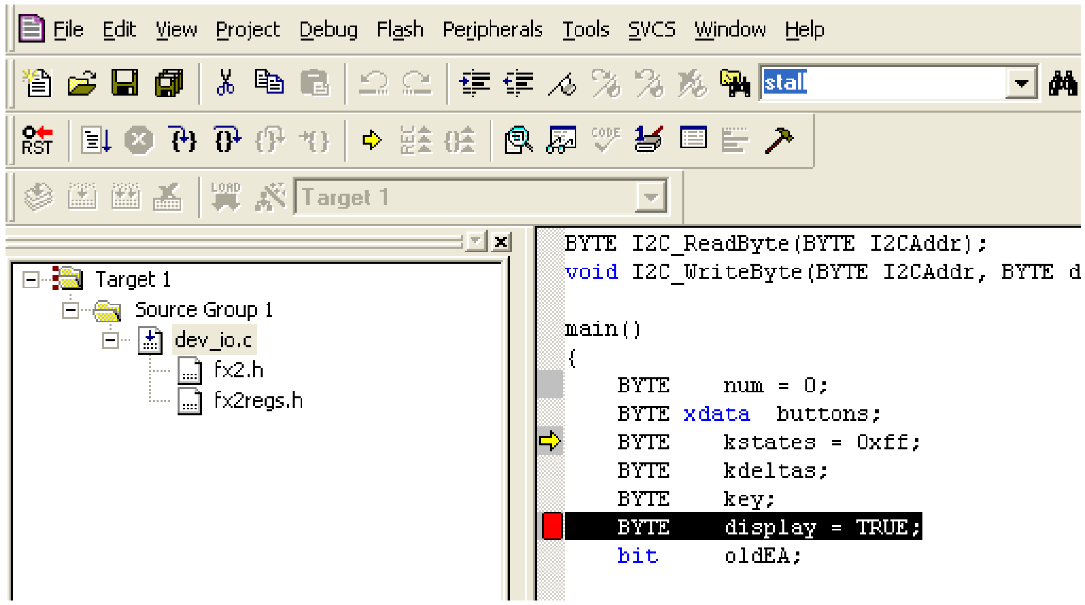
14. Use the **Step Over** button to step through the code by selecting **View > Debug** Toolbar.

Figure 8-29. Enabling Debug Toolbar View in Keil


dev_io - µVision2 - [Disassembly]
File Edit View Project Debug Peripherals Tools
Status Bar
File Toolbar
Build Toolbar
Debug Toolbar

15. View the output window to verify that you are connected to the monitor and that your program has loaded (it displays a message such as Connected to Monitor-51 V3.0).
16. In the **Project Window->Files Tab** click on dev_io.c.Observe the Arrow marked in yellow in Figure 8-30. The yellow arrow in the main indicates that the code execution stopped at that point.

Figure 8-30. Enabling Debug Toolbar View in Keil

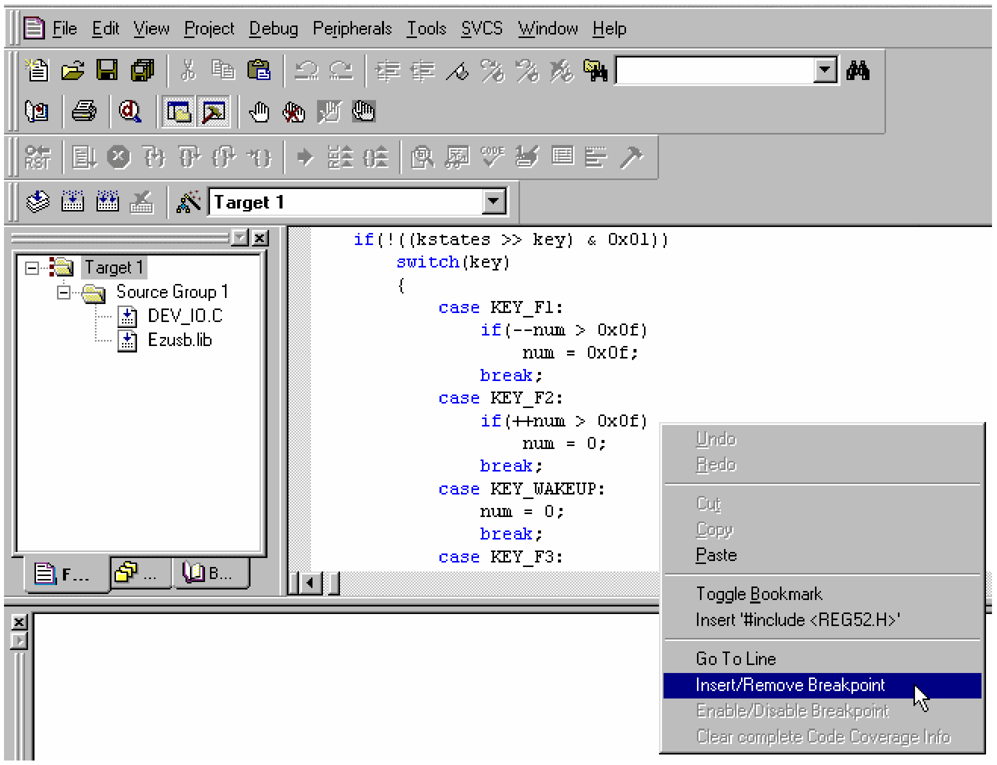

File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
stall
RST
Target 1
Target 1
Source Group 1
dev_io.c
fx2.h
fx2regs.h

```
BYTE I2C_ReadByte(BYTE I2CAddr);
void I2C_WriteByte(BYTE I2CAddr, BYTE d

main()
{
    BYTE    num = 0;
    BYTE xdata  buttons;
    BYTE    kstates = 0xff;
    BYTE    kdeltas;
    BYTE    key;
    BYTE    display = TRUE;
    bit     oldEA;
```

17. Set a breakpoint by selecting the first line in the "case KEY_F2" section (which is in file dev_io.c). To set or remove a breakpoint, double-click the line or right-click on the line as and select Insert/Remove Breakpoint shown in Figure 8-31.

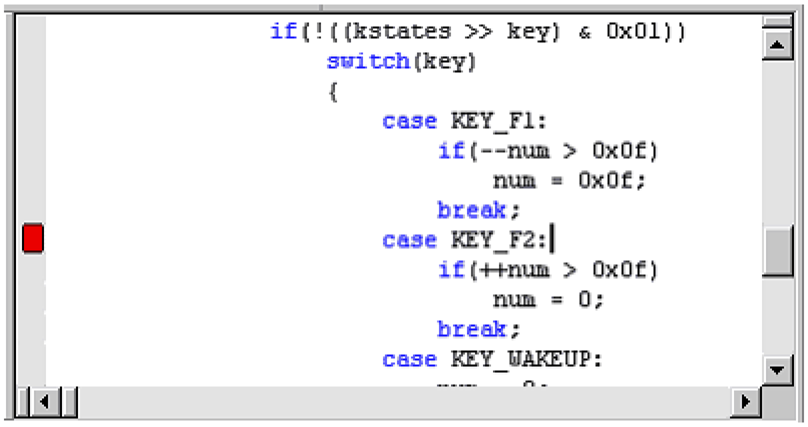
Figure 8-31. Setting Breakpoint in Keil uVision2 IDE


File Edit View Project Debug Peripherals Tools SVCS Window Help
Target 1
Target 1
Source Group 1
DEV_IO.C
Ezusb.lib
F...
...
B...
Undo
Redo
Cut
Copy
Paste
Toggle Bookmark
Insert '#include <REG52.H>'
Go To Line
Insert/Remove Breakpoint
Enable/Disable Breakpoint
Clear complete Code Coverage Info

```
if(!((kstates >> key) & 0x01))
    switch(key)
    {
        case KEY_F1:
            if(--num > 0x0f)
                num = 0x0f;
            break;
        case KEY_F2:
            if(++num > 0x0f)
                num = 0;
            break;
        case KEY_WAKEUP:
            num = 0;
            break;
        case KEY_F3:
```

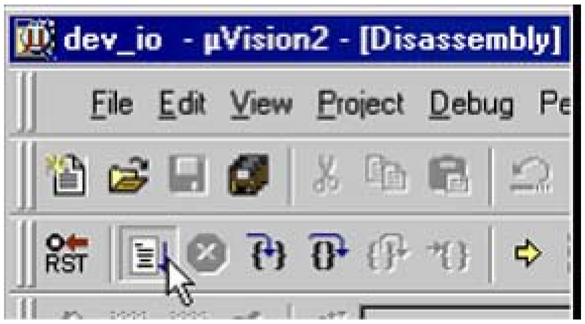
18. A red breakpoint indication is seen in the margin next to the new breakpoint as shown in Figure 8-32. Press RUN button as shown in Figure 8-33.

Figure 8-32. Breakpoint Indicator in Keil uVision2 IDE



```
            if(!((kstates >> key) & 0x01))
                switch(key)
                {
                    case KEY_F1:
                        if(--num > 0x0f)
                            num = 0x0f;
                        break;
                    case KEY_F2:|
                        if(++num > 0x0f)
                            num = 0;
                        break;
                    case KEY_WAKEUP:
```

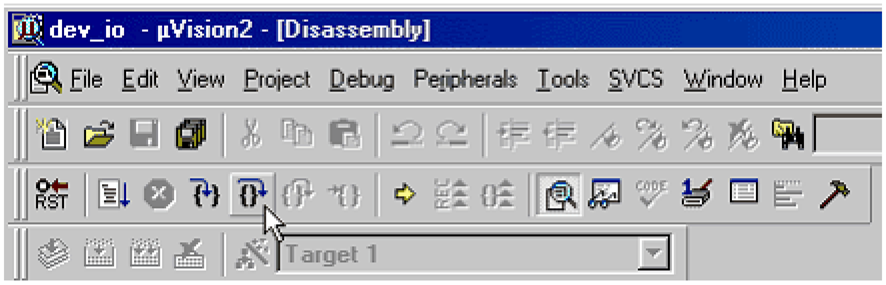
Figure 8-33. Run button in Keil


dev_io - µVision2 - [Disassembly]
File Edit View Project Debug Pe
RST

19. Now press F3 on the development board (the KEY_F2 label equates to the F3 button). Program execution halts in the Keil IDE and the LED does not increment.

20. Press Step Over as shown in Figure 8-34. Then press Run key again on the debugger.

Figure 8-34. Step Over Debug Button in Keil uVision2 IDE


dev_io - µVision2 - [Disassembly]
File Edit View Project Debug Peripherals Tools SVCS Window Help
RST
Target 1

21. Execution proceeds normally until F3 key is pressed on the development board again. When finished, press the Stop Debugging key and exit the Keil debugger.

Note: If the Keil debugging does not proceed then re-build the firmware project and redo steps 12-21

9. Resources

9.1 Hardware Resources

The CY3687 MoBL-USB development kit contains several hardware resources that guide you in designing your own custom board. The documents in the /Hardware directory of the DVK content.

- **CY3687_MOBL-USB_FX2LP18_DVK_BOM.xls:** This document lists all the hardware components like resistors,capcacitors,,etc used in designing the development board.
- **CY3687_MOBL-USB_FX2LP18_DVK_schematic.pdf/ CY3687_MOBL-USB_FX2LP18_DVK_Schematic.DSN:** These documents show the schematic design of the MoBL-USB development board. The .DSN file can be viewed using Orcad software tools.
- **CY3687_MOBL-USB_FX2LP18_DVK_Gerber.zip:** This file can be used to understand the via, trace lengths, electrical connections, etc., of the CY3687 MoBL-USB FX2LP18 development board.
- **CY3687_MOBL-USB_FX2LP18_DVK_Board_Layout.brd/ CY3687_MOBL-USB_FX2LP18_DVK_Board_Layout.pdf:** The file contains PCB layout information of CY3687 MoBL-USB FX2LP18 development board. The editable layout file (.brd) can be viewed using Allegro PCB software.

9.2 Reference Designs

9.2.1 CY4625 - MoBL-USB Bridge Reference Design

This CY4625 reference design depicts a use-case of MoBL-USB™ firmware, for morphing the MoBL-USB™ device as a bridge between a 'Master' and external USB 2.0 compliant host / peripheral. The 'master' can be standard microprocessors, DSPs, ASICs, or FPGAs.This reference firmware design also has modifications, for allowing any MTP compliant host application like 'windows media player' to communicate to the 'master', connected to MoBL_USB™ device.

9.3 Application Notes

- AN65209 - Getting Started with FX2LP

  This application note presents the features and resources available to speed up the EZ-USB® FX2LP™-based design from concept to production. This document serves as a starting point for the new user to get familiar with FX2LP. It also gives an overview of the design resources available
- AN1168 - High-speed USB PCB Layout Recommendations

  This application note details guidelines for designing, controlled-impedance; high-speed USB printed circuit boards to comply with the USB specification. This note is applicable to all Cypress high-speed USB solutions. Some Cypress high-speed USB chips have separate application notes that address chip-specific PCB design guidelines
- AN45197 - Using the Hex2bix Conversion Utility

Hex2bix is a program used to convert a .hex file to a raw binary, A51, or IIC format. This application note describes how to use the Hex2bix conversion utility for successful file conversion.

- AN15456 - Guide to Successful EZ-USB(R) FX2LP(TM) and EZ-USB FX1(TM) Hardware Design and Debug

  This application note outlines a process that isolates many of the most likely causes of EZ-USB® FX2LP™ and EZ-USB FX1™ hardware problems. 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.
- AN5078 - EZ-USB Hardware - Design considerations for EEPROM usage

  EZ-USB® downloads firmware automatically into the on-chip RAM from the EEPROM connected to it. The purpose of this application note is to present recommended design guidelines for assuring the data integrity of serial EEPROM devices when used in EZ-USB designs.
- AN064 - EZ-USB FX2LP™/AT2LP™ Reset and Power Considerations

  The Cypress EZ-USB FX2LP(TM) is a USB 2.0 high-speed device. It contains an 8051, 16K of program/data memory, 4K of endpoint buffers and a General Programmable Interface (GPIF) block. The EZ-USB AT2LP(TM) is a USB 2.0 high-speed ATA/ATAPI bridge chip. Both these chips have similar power and reset needs. This application note refers to the FX2LP, but is also applicable to AT2LP.
- AN15813 - Monitoring the EZ-USB FX2LP™ VBUS

  This application note explains the purpose and methods of monitoring VBUS from the upstream connector using the EZ-USB FX2LP.
- AN43841 - EZ-USB® FX2LPTM/FX2LP18 56-Ball BGA PCB Layout Guidelines

  The 56-ball VFBGA version of the FX2LP(CY7C68013A) or FX2LP18(CY7C68053) USB microcontroller chips is a smaller package version of the QFN package. The 56-ball package meets the needs of space sensitive printed circuit board (PCB) designs. This application note provides guidelines for designing a PCB with either FX2LP(CY7C68013A) or FX2LP18(CY7C68053).
- AN4067 - Endpoint FIFO Architecture of EZ-USB FX1/FX2LP™

  This application note describes the FIFO architecture of the EZ-USB FX1, the full speed USB microcontroller and the EZ-USB FX2LP"·, the high-speed USB microcontroller. The purpose of this application note is to help the user understand the very basics of the FX1/FX2LP and get familiar with the terminologies used while describing the data flow in FX1/FX2LP. The application note addresses three modes of operation of the FX1/FX2LP, Endpoint Configuration and Multiple Buffering, Three Domains that form the basic component of the FIFO architecture, Arming and committing endpoint buffers Endpoint operation in manual vs. auto mode.
- AN4053 - Streaming Data Through Isochronous/Bulk Endpoints on EZ-USB® FX2™ and EZ-USB FX2LP™

  This application note provides brief background information on what is involved while designing for a streaming application using the EZ-USB FX2(TM) or the EZ-USB FX2LP(TM) part. It provides information on streaming data through bulk endpoints, isochronous endpoints, and high-bandwidth isochronous endpoints along with pitfalls to consider and avoid while using the FX2/FX2LP for designing high-bandwidth applications.
- AN67442 - SPI Implementation Using Serial Mode-0 of EZ-USB FX2LP™

  This application note describes the implementation of serial peripheral interface (SPI) protocol using the FX2LP UART port in serial mode 0. This demonstration uses FX2LP as the SPI master for transferring data to and from an AT25080A EEPROM device. The example code includes functions to the Write/Read byte to and from AT25080A EEPROM.
- 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. The hardware connections and example code are included along with this application note.

- AN58170 - Code/Memory Banking Using EZ-USB®

  The EZ-USB® family of chips has an 8051 core. The 8051 core has a 16-bit address line and is only able to access 64 KB of memory. However, the firmware size sometimes exceeds 64 KB This application note describes methods of overcoming this 64 KB limitation and also demonstrates the implementation of one such method.
- AN57322 - Interfacing SRAM with FX2LP over GPIF

  This application note discusses how to connect Cypress SRAM CY7C1399B to FX2LP over the General Programmable Interface (GPIF). It describes how to create read and write waveforms using the GPIF Designer. This application note is also useful as a reference to connect FX2LP to other SRAMs.
- AN14558 - Implementing a SPI Interface with EZ-USB FX2LP™

  This application note demonstrates how to implement a SPI interface. It uses the EZ-USB FX2LP as a SPI Master and a SPI Serial EEPROM (25AA256) as a SPI slave. This example comes with a host application with which the user can access the EEPROM. The EZ-USB FX2LP firmware uses the ports mode and bit-bangs the General Purpose IOs to create the SPI interface. The hardware connection diagram and code listing is included.
- AN1193 - Using Timer Interrupt in Cypress EZ-USB® FX2LP™ Based Applications

  This application note is aimed at helping EZ-USB® FX2LP™ based firmware developers use timer interrupts in their applications, by providing a framework based timer interrupt program written in C. The assumption is made that one has a general understanding of how interrupts work within the 8051 concept. When this program is run, you should be able to light the seven-segment LED on the FX2LP Development Board (CY3684) with a 0-9 count, and control the step rate (1s - 5s) using BULK OUT endpoint transfers from the EZ-USB Control Panel.
- AN63787 - EZ-USB FX2LP™ GPIF and Slave FIFO Configuration Examples using FX2LP Back-to-Back Setup

  AN63787 discusses how to configure the general programmable interface (GPIF) and slave FIFO's of 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 in back-to-back setup; the first one acting in master mode and the second in slave mode.
- AN61244 - Firmware Optimization in EZ-USB®

  The EZ-USB® family of chips has an 8051 core and uses the standard 8051 instruction set. However, it has a few enhancements compared to the standard 8051. This application note describes firmware optimization methods in EZ-USB. Some of these methods are common for any processor and some specific to the 8051 core of EZ-USB.
- AN70983 - EZ-USB FX2LP™ Bulk Transfer Application in C# Using SuiteUSB C# Library (CyUSB.dll)

  AN70983 demonstrates how to build an application on Visual C# to send bulk data out and receive it back over a bulk endpoint of FX2LP, which is developed using Cypress SuiteUSB C# library (CyUSB.dll) for creating Windows applications using Microsoft Visual Studio. This document also explains associated firmware used in FX2LP to implement loopback transfers on bulk endpoints, and the application is tested with FX2LP Development kit.
- AN70486 - EZ-USB® FX2LP™ Host Application in VC++ 2008 Using Suite USB Library (CYUSB.dll)

This application note demonstrates how to build a host application on Microsoft Visual C++ platform, using the Cypress SuiteUSB C# library, CyUSB.dll, to perform USB BULK IN and OUT transfers with FX2LP and the associated project is tested with FX2LP Development kit.

- AN74505 - EZ-USB® FX2LP™ - Developing USB Application on MAC OS X using LIBUSB

  AN74505 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 of FX2LP, using the interfaces provided by the APIs of libusb. This host application implements the transfer only with devices that pass the particular VID/PID(=0x04B4/0x1004) identification.
- AN6077 - Implementing an 8-Bit Asynchronous Interface with FX2LP™

  AN6077 discusses how to configure the general programmable interface (GPIF) and slave FIFOs of the EZ-USB FX2LP™ to implement an 8-bit asynchronous interface. The GPIF is a programmable 8- or 16-bit parallel interface that reduces system costs by providing a glueless interface between the EZ-USB FX2LP and different types of external peripherals. The GPIF allows the EZ-USB FX2LP to perform local bus mastering to external peripherals implementing a wide variety of protocols. For example, EIDE/ATAPI, printer parallel port (IEEE P1284), Utopia, and other interfaces are supported using the GPIF block of the EZ-USB FX2LP. In this example, it masters the slave FIFO interface of another EZ-USB FX2LP.
- AN58764 - Implementing a Virtual COM Port in FX2LP

  This application note explains how to implement a virtual COM port device using the standard Windows driver in FX2LP. This information helps in easy migration from UART to USB. The example code is provided with the application note, along with the required descriptors, class specific request handling, and the INF file required for enumeration.
- AN50963 - Firmware Download Methods to FX1/FX2LP

  This is an advanced document on firmware download techniques and readers expected to be familiar with VC++ programming, USB 2.0 protocol, FX1/FX2LP architecture and device configuration options. Refer FX1/FX2LP datasheet and Technical Reference Manual available in Cypress website for more details on FX1/FX2LP product architecture and configuration details.
- AN45471 - Vendor Command Design Guide for the FX2LP

  Vendor commands are used to issue commands to a device, by which tasks unique to an application are accomplished. This application note demonstrates how you can quickly design USB vendor commands to perform specific features of products. In addition, using the Cypress CyConsole utility to issue vendor commands is also explained.
- AN58009 - Serial (UART) Port Debugging of FX1/FX2LP Firmware

  This application note describes the code needed in the FX2LP firmware for serial debugging. This code enables the developer to print debug messages and real time values of the required variables in the HyperTerminal of the PC or capture it in a file using the UART engine in FX2LP.
- AN42499 - Setting Up, Using, and Troubleshooting the Keil(TM) 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, the guidelines to start and stop a debug session, set breakpoints, step through code, and solve potential problems are considered.
- AN023 - USB Compliance Testing Overview

  This program verifies that your USB devices meets the specification and works well with other USB devices.
- AN6076 - Differences Between EZ-USB FX2LP™ and MoBL-USB™ FX2LP18

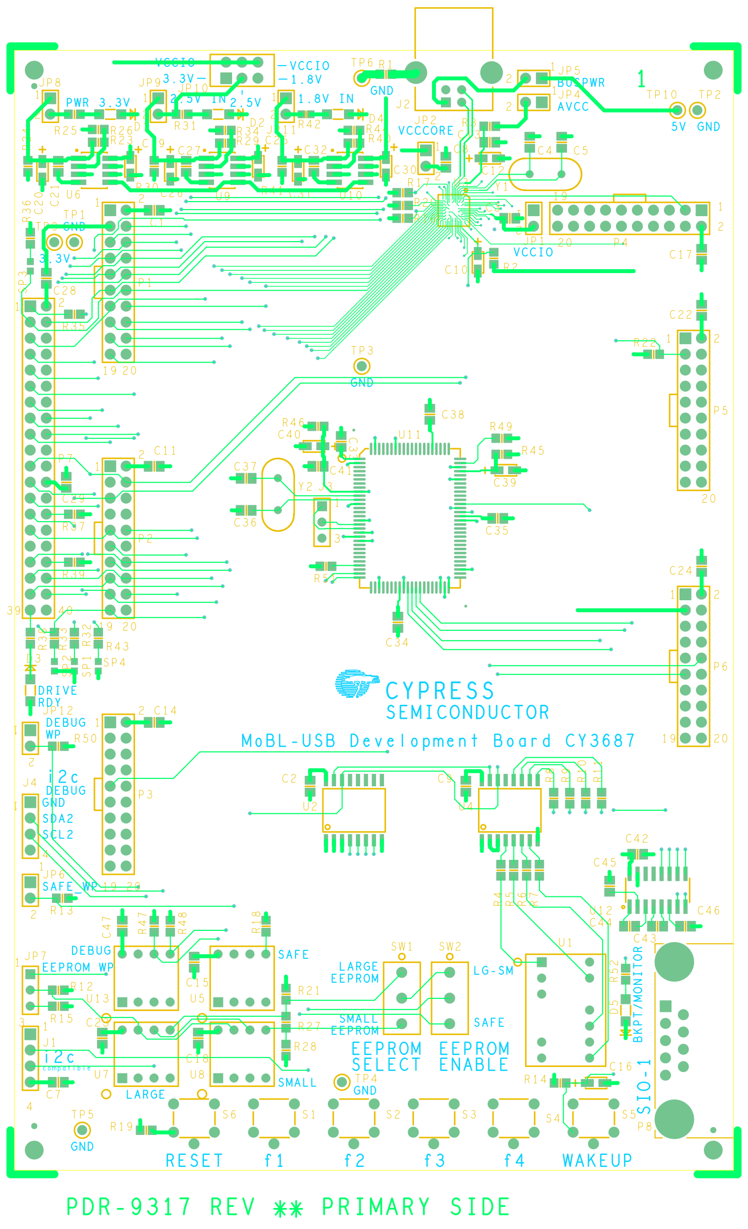
  This application note explains the differences between the EZ-USB FX2LP™ and MoBL-USB™ FX2LP18 USB Microcontroller. This application note will be useful to engineers experienced with

EZ-USB FX2LP who are switching to the MoBL-USB FX2LP18, as well as engineers starting off with the MoBL-USB FX2LP18.

A. Appendix

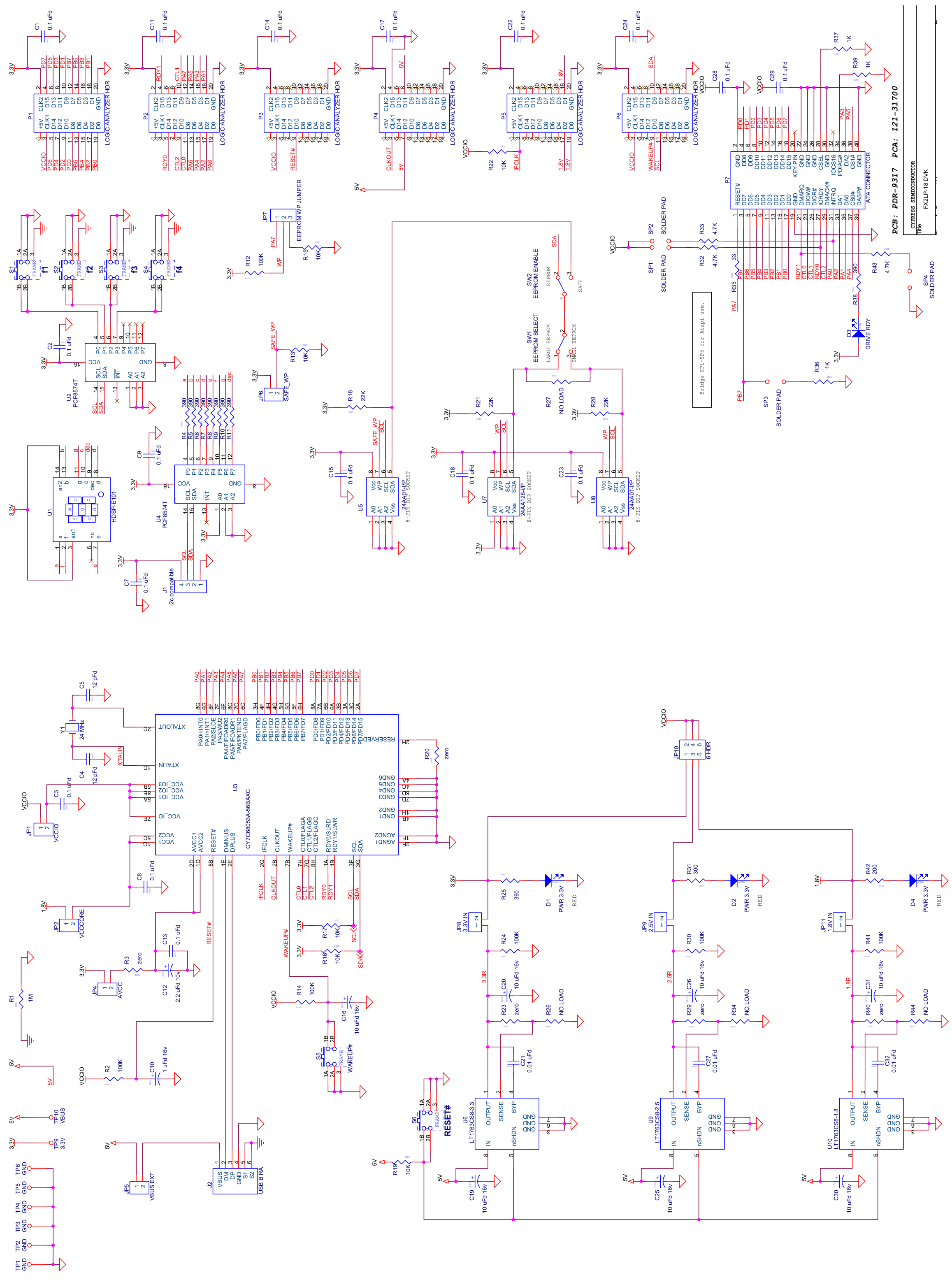
A.1 Board Layout

Figure A-1. CY3687 Assembly Development Board Layout

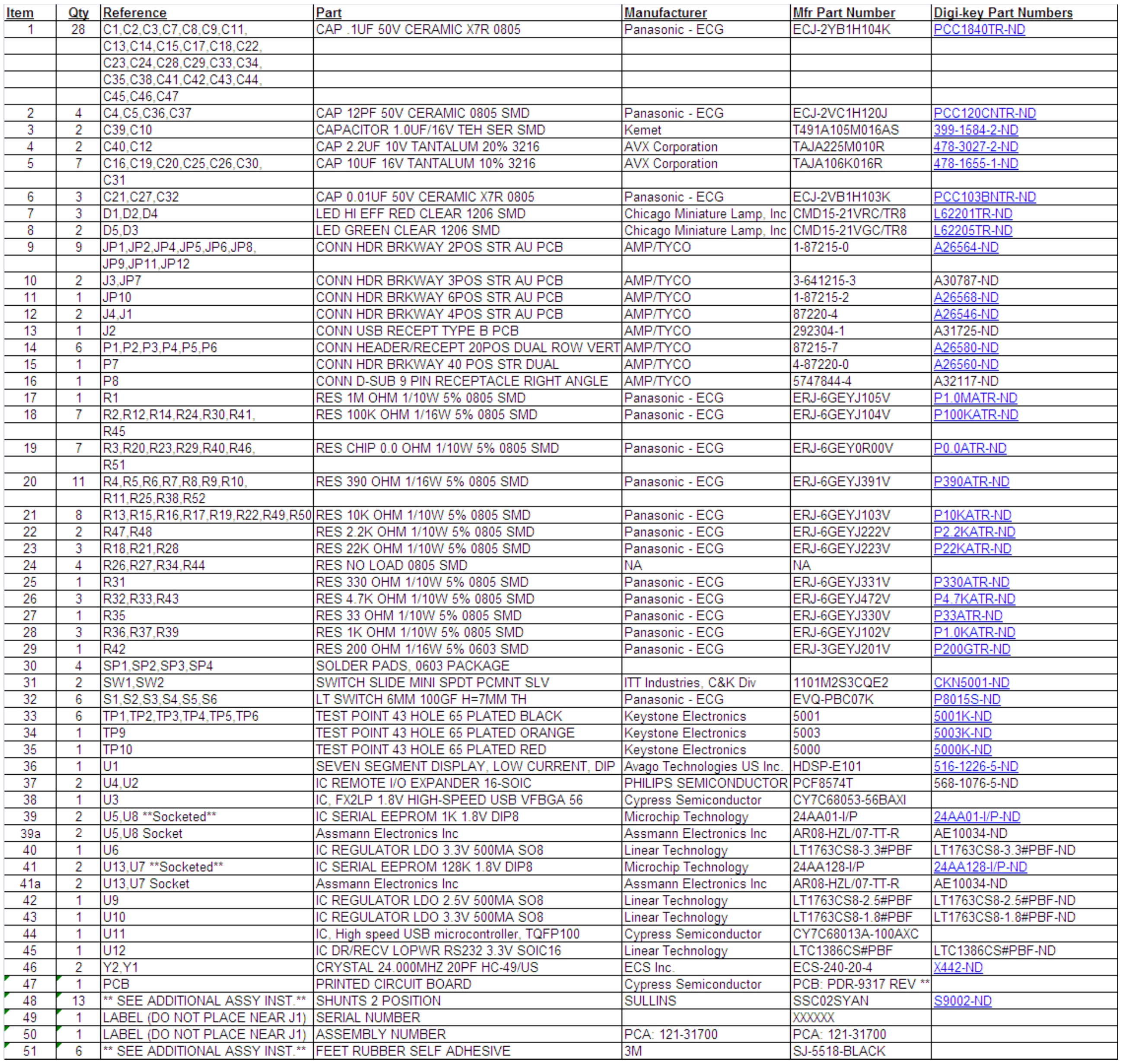


A.2 Schematic

Figure A-2. CY3687 Schematics



A.3 PCB BOM



Item	Qty	Reference	Part	Manufacturer	Mfr Part Number	Digi-key Part Numbers
1	28	C1,C2,C3,C7,C8,C9,C11,	CAP .1UF 50V CERAMIC X7R 0805	Panasonic - ECG	ECJ-2YB1H104K	PCC1840TR-ND
		C13,C14,C15,C17,C18,C22,				
		C23,C24,C28,C29,C33,C34,				
		C35,C38,C41,C42,C43,C44,				
		C45,C46,C47				
2	4	C4,C5,C36,C37	CAP 12PF 50V CERAMIC 0805 SMD	Panasonic - ECG	ECJ-2VC1H120J	PCC120CNTR-ND
3	2	C39,C10	CAPACITOR 1.0UF/16V TEH SER SMD	Kemet	T491A105M016AS	399-1584-2-ND
4	2	C40,C12	CAP 2.2UF 10V TANTALUM 20% 3216	AVX Corporation	TAJA225M010R	478-3027-2-ND
5	7	C16,C19,C20,C25,C26,C30,	CAP 10UF 16V TANTALUM 10% 3216	AVX Corporation	TAJA106K016R	478-1655-1-ND
		C31				
6	3	C21,C27,C32	CAP 0.01UF 50V CERAMIC X7R 0805	Panasonic - ECG	ECJ-2VB1H103K	PCC103BNTR-ND
7	3	D1,D2,D4	LED HI EFF RED CLEAR 1206 SMD	Chicago Miniature Lamp, Inc	CMD15-21VRC/TR8	L62201TR-ND
8	2	D5,D3	LED GREEN CLEAR 1206 SMD	Chicago Miniature Lamp, Inc	CMD15-21VGC/TR8	L62205TR-ND
9	9	JP1,JP2,JP4,JP5,JP6,JP8,	CONN HDR BRKWAY 2POS STR AU PCB	AMP/TYCO	1-87215-0	A26564-ND
		JP9,JP11,JP12				
10	2	J3,JP7	CONN HDR BRKWAY 3POS STR AU PCB	AMP/TYCO	3-641215-3	A30787-ND
11	1	JP10	CONN HDR BRKWAY 6POS STR AU PCB	AMP/TYCO	1-87215-2	A26568-ND
12	2	J4,J1	CONN HDR BRKWAY 4POS STR AU PCB	AMP/TYCO	87220-4	A26546-ND
13	1	J2	CONN USB RECEPT TYPE B PCB	AMP/TYCO	292304-1	A31725-ND
14	6	P1,P2,P3,P4,P5,P6	CONN HEADER/RECEPT 20POS DUAL ROW VERT	AMP/TYCO	87215-7	A26580-ND
15	1	P7	CONN HDR BRKWAY 40 POS STR DUAL	AMP/TYCO	4-87220-0	A26560-ND
16	1	P8	CONN D-SUB 9 PIN RECEPTACLE RIGHT ANGLE	AMP/TYCO	5747844-4	A32117-ND
17	1	R1	RES 1M OHM 1/10W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ105V	P1.0MATR-ND
18	7	R2,R12,R14,R24,R30,R41,	RES 100K OHM 1/16W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ104V	P100KATR-ND
		R45				
19	7	R3,R20,R23,R29,R40,R46,	RES CHIP 0.0 OHM 1/10W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEY0R00V	P0.0ATR-ND
		R51				
20	11	R4,R5,R6,R7,R8,R9,R10,	RES 390 OHM 1/16W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ391V	P390ATR-ND
		R11,R25,R38,R52				
21	8	R13,R15,R16,R17,R19,R22,R49,R50	RES 10K OHM 1/10W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ103V	P10KATR-ND
22	2	R47,R48	RES 2.2K OHM 1/10W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ222V	P2.2KATR-ND
23	3	R18,R21,R28	RES 22K OHM 1/10W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ223V	P22KATR-ND
24	4	R26,R27,R34,R44	RES NO LOAD 0805 SMD	NA	NA	
25	1	R31	RES 330 OHM 1/10W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ331V	P330ATR-ND
26	3	R32,R33,R43	RES 4.7K OHM 1/10W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ472V	P4.7KATR-ND
27	1	R35	RES 33 OHM 1/10W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ330V	P33ATR-ND
28	3	R36,R37,R39	RES 1K OHM 1/10W 5% 0805 SMD	Panasonic - ECG	ERJ-6GEYJ102V	P1.0KATR-ND
29	1	R42	RES 200 OHM 1/16W 5% 0603 SMD	Panasonic - ECG	ERJ-3GEYJ201V	P200GTR-ND
30	4	SP1,SP2,SP3,SP4	SOLDER PADS, 0603 PACKAGE			
31	2	SW1,SW2	SWITCH SLIDE MINI SPDT PCMNT SLV	ITT Industries, C&K Div	1101M2S3CQE2	CKN5001-ND
32	6	S1,S2,S3,S4,S5,S6	LT SWITCH 6MM 100GF H=7MM TH	Panasonic - ECG	EVQ-PBC07K	P8015S-ND
33	6	TP1,TP2,TP3,TP4,TP5,TP6	TEST POINT 43 HOLE 65 PLATED BLACK	Keystone Electronics	5001	5001K-ND
34	1	TP9	TEST POINT 43 HOLE 65 PLATED ORANGE	Keystone Electronics	5003	5003K-ND
35	1	TP10	TEST POINT 43 HOLE 65 PLATED RED	Keystone Electronics	5000	5000K-ND
36	1	U1	SEVEN SEGMENT DISPLAY, LOW CURRENT, DIP	Avago Technologies US Inc.	HDSP-E101	516-1226-5-ND
37	2	U4,U2	IC REMOTE I/O EXPANDER 16-SOIC	PHILIPS SEMICONDUCTOR	PCF8574T	568-1076-5-ND
38	1	U3	IC, FX2LP 1.8V HIGH-SPEED USB VFBGA 56	Cypress Semiconductor	CY7C68053-56BAXI	
39	2	U5,U8 **Socketed**	IC SERIAL EEPROM 1K 1.8V DIP8	Microchip Technology	24AA01-I/P	24AA01-I/P-ND
39a	2	U5,U8 Socket	Assmann Electronics Inc	Assmann Electronics Inc	AR08-HZL/07-TT-R	AE10034-ND
40	1	U6	IC REGULATOR LDO 3.3V 500MA SO8	Linear Technology	LT1763CS8-3.3#PBF	LT1763CS8-3.3#PBF-ND
41	2	U13,U7 **Socketed**	IC SERIAL EEPROM 128K 1.8V DIP8	Microchip Technology	24AA128-I/P	24AA128-I/P-ND
41a	2	U13,U7 Socket	Assmann Electronics Inc	Assmann Electronics Inc	AR08-HZL/07-TT-R	AE10034-ND
42	1	U9	IC REGULATOR LDO 2.5V 500MA SO8	Linear Technology	LT1763CS8-2.5#PBF	LT1763CS8-2.5#PBF-ND
43	1	U10	IC REGULATOR LDO 3.3V 500MA SO8	Linear Technology	LT1763CS8-1.8#PBF	LT1763CS8-1.8#PBF-ND
44	1	U11	IC, High speed USB microcontroller, TQFP100	Cypress Semiconductor	CY7C68013A-100AXC	
45	1	U12	IC DR/RECV LOPWR RS232 3.3V SOIC16	Linear Technology	LTC1386CS#PBF	LTC1386CS#PBF-ND
46	2	Y2,Y1	CRYSTAL 24.000MHZ 20PF HC-49/US	ECS Inc.	ECS-240-20-4	X442-ND
47	1	PCB	PRINTED CIRCUIT BOARD	Cypress Semiconductor	PCB: PDR-9317 REV **	
48	13	** SEE ADDITIONAL ASSY INST.**	SHUNTS 2 POSITION	SULLINS	SSC02SYAN	S9002-ND
49	1	LABEL (DO NOT PLACE NEAR J1)	SERIAL NUMBER		XXXXXX	
50	1	LABEL (DO NOT PLACE NEAR J1)	ASSEMBLY NUMBER	PCA: 121-31700	PCA: 121-31700	
51	6	** SEE ADDITIONAL ASSY INST.**	FEET RUBBER SELF ADHESIVE	3M	SJ-5518-BLACK	