EZ-USB Advanced Development Kit Users Guide version 1.1

Table of Contents

1 Introduction

Welcome to the EZ-USB Advanced Development Kit (DVK). This guide is a good place to get started with the DVK. Included are details of the DVK software contents and a tutorial to get you on the way in developing your EZ-USB based peripheral.

For those of you familiar with previous EZ-USB DVKs, this one should be quite familiar. It shares the same design philosophy and directory structure as previous DVKs. However, it has some new features that will allow for more easy and robust designs. Specifically, this DVK includes components from the new Cypress USB Studio.

The EZ-USB Advanced DVK is designed to work only with the EZ-USB FX2LP and FX1 chips. The DVK does not support older Cypress EZ-USB chips such as the EZ-USB AN21xx, FX, and FX2.

Section	Description
Development Kit Contents	Describes the directories and files that are installed on your system after running the DVK Setup program
EZ-USB Firmware Frameworks	The Firmware FrameWorks is template 8051 source code that can be customized for your device
Firmware Download Driver	Devices that use ReNumeration require a special driver to perform firmware download. This section describes the example download driver that is part of the DVK and has instructions for customizing the driver for your device.
Tutorials	This section contains step by step tutorials that demonstrate common development tasks

1.1 Cypress USB Studio

The EZ-USB DVK includes the Cypress USB Studio. Cypress USB Studio is a collection of components that enable rapid and robust development of a USB peripheral and associated host software and device drivers. The following sections describe the components that comprise the Cypress USB Studio.

1.1.1 Cypress Generic USB Driver

The Cypress Generic USB Driver is a robust high-performance Windows driver used for all device interaction in the DVK. You may also distribute the driver with your EZ-USB based device. Documentation for the driver is installed as part of the DVK and is available from the Windows Start menu under Cypress Help.

The driver is distributed in binary form only. Contact Cypress for information on source code licensing.

1.1.2 Cypress USB Class Library API

The Cypress USB Class Library API (Application Programming Interface) or CyApi is a C++ class library that simplifies access to your device via the Cypress Generic USB Driver. The DVK installation includes Microsoft and Borland versions of the library. Documentation for CyApi is installed as part of the DVK and is available from the Windows Start menu under Cypress Help.

The source code for the CyBulk and CyDesc examples located in the **\cypress\usb\util** directory demonstrate use of CyApi.

1.1.3 Cypress USB Console

The Cypress USB Console (CyConsole) is a Windows Application that you can use to exercise your device via the Cypress Generic USB Driver. CyConsole provides a generic USB interface as well as a special interface option for accessing features unique to EZ-USB. Documentation for CyConsole is accessible from the application itself or from the Windows Start menu under Cypress Help. Also, the Tutorials section at the end of this guide demonstrates some of the CyConsole features.

1.1.4 Cypress GPIF Designer

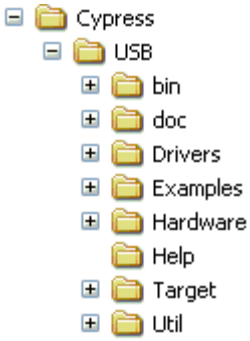
Cypress GPIF Designer allows a USB developer to easily create and modify the waveforms descriptors required to configure the GPIF. The GPIF (General Programmable InterFace) is a feature of EZ-USB that provides a highly configurable and flexible glueless peripheral interface allowing the highest possible bandwidth to be achieved over the physical layer.

GPIF Designer is an optional part of the EZ-USB DVK installation and is included in a separate folder on the distribution CD and can also be downloaded from the Cypress website. You can access it from the Start menu under Cypress.

2 Development Kit Contents

This section provides a detailed description and explanation of the structure and content of the EZ-USB Development Kit as it exists on the users PC after the installation from the EZ-USB DVK CD.

The following image shows the root level tree that is on the user system after DVK installation. This assumes that the user chose to install all DVK components (default). Subsequent sections will detail the contents of each sub-directory.



The DVK installer also installs several development board driver related files in the Windows directory tree. These files are discussed in the **Windows** section below.

2.1 Bin

This directory contains utility programs utilized within the EZ-USB Development Kit. Note that copies of these programs may exist in other directories within the users PC.

Files in the Bin directory:

cyconsole.exe	Cypress USB Console. The main GUI in the DVK. Cyconsole accesses devices bound to the Cypress Generic Driver.
cyconsole.chm	Help file for CyConsole
cyscript.exe	Utility for converting a firmware .HEX file into a download script file.
hex2bix.exe	This file converts the .hex output file to a .bix file that can be downloaded directly into target memory. It also converts the .hex output to a .iic file that can be directly loaded into an EEPROM.
setenv.bat	This batch file sets the environment for the command-line Keil Tools.
cybulk.exe	This is a bulk loopback test application that exercises the EZ-USB bulk endpoints. This test consists of a windows application that may be used to loop data to and from any device using the Cypress Generic Driver, and supporting bulk endpoints.

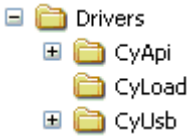
2.2 Doc

This directory contains the sub-directories and support files for the documentation of the EZ-USB Advanced Development Kit.

FX2LP	Contains the Technical Reference Manual, Datasheet, Errata, and other documentation specific to the EZ-USB FX2LP chip. The Technical Reference Manual is common to both FX2LP and FX1.
FX1	Contains the Datasheet, Errata, and other documentation specific to the EZ-USB FX1 chip.
General	Contains general DVK documentation - such as this guide. Printable PDF versions of the various help files in the kit are also located here.

2.3 Drivers

This directory tree contains the Cypress Generic USB Driver (CyUsb), the CyApi Class Library (CyApi), and the firmware download driver example (CyLoad).



Drivers\CYAPI	Contains documentation, library files (both Microsoft and Borland), and header files for the Cypress Generic USB Driver Class Library
Drivers\CYUSB	Contains documentation, driver binaries, .INF files, and scripts associated with the Cypress Generic USB Driver
Drivers\CyLoad	Contains an example for building a custom firmware download driver. See the section "Firmware Download Drivers" for more information.

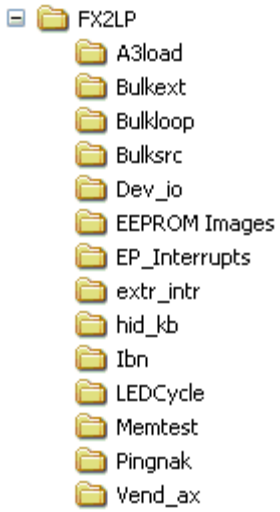
Files in Drivers\cyusb:

cymon.inf.bak	.INF file for the Development Board. This .INF matches the VID/PID of the development board to the CYUSB driver and automatically downloads the Keil monitor. [**note1**]
cyusb.chm	Help file for the Cypress Generic USB Driver. Contains details of the low level driver interface.
cyusb.inf.bak	.INF file for the generic FX2LP VID/PID and many of the firmware examples. [**note1**]
cyusb.sys	The Windows XP/2000 driver binary for the Cypress Generic USB Driver.
cyusbme.sys	The Windows ME driver binary for the Cypress Generic USB Driver.
mon.spt	Script file for downloading the Keil monitor to the Development Board.
inc\cyioctl.h	defines IOCTLs and structures for non-Class Library access to the Generic USB Driver

[**note1**] The .INF files stored in this directory are provided for the purpose of installing the Cypress Generic USB driver on a system that the DVK software has not been installed on. The DVK installer automatically pre-copies .INF and .SYS files for the EZ-USB development board to the \windows\inf and \windows\system32\drivers directories. To avoid user user interaction, the pre-copied versions of the .INF files do not contain a CopyFiles directive. The .INF files in this directory do contain a CopyFiles directive and are appropriate for plug-and-play installation of the driver files from floppy or cd-rom. The files have been renamed with a .BAK extension to prevent the Add New Hardware WIzard from attempting to use these .INF files on systems where the DVK software has been installed. For more information on the .INF file format, consult the Microsoft Windows DDK documentation.

2.4 Examples

This directory contain example 8051 Code for the EZ-USB FX2LP and FX1. See the readme.txt file in each directory for more detailed information on the example. The FX2LP tree is shown here. A separate FX1 directory contains the same examples for FX1. The firmware in the FX2LP and FX1 directories is for the most part identical, since the two chips are code and binary compatible.



a3load	Contains firmware that handles vendor specific SETUP command (0xA3). This command performs a download to external RAM.
bulkext	Contains a bulk loopback test that exercises the EZ-USB bulk endpoints. Use the windows application cybulk.exe to drive the firmware. 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
bulkloop	Contains a bulk loopback test that exercises the EZ-USB bulk endpoints. Use the windows application ..\CYPRESS\USB\Bin\ cybulk.exe to drive the firmware. It loops back EP2OUT to EP6IN and EP4OUT to EP8IN.
bulksrc	Contains bulk endpoint endless source/sink firmware. It may be driven using the CyConsole or CyBulk. EP2OUT will always accept a bulk OUT EP4OUT will always accept a bulk OUT EP6IN will always return a 512 byte packet, 64 bytes at full-speed. Two different packets are returned: All 0x02's and all 0x03's EP8IN will continuously return the packet most recently written to EP4OUT
dev_io	Contains the source files to build simple development board I/O sample. The purpose of this software is to demonstrate how to use the buttons and LED on the EZ-USB developer's kit. See the readme.txt file located within the directory for details on running the test.
EEPROM IMAGES	Contains the default values for EEPROM for the Dev Kit board.
EP_Interrupts	Bulk loopback firmware that demonstrates use of endpoint interrupts.
extr_intr	Firmware that demonstrates external interrupt handling.
hid_kb	Example firmware that emulates a HID-class keyboard using the buttons and seven-segment display on the DVK board.
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. See the readme.txt file located within the directory for details on running the test.
LEDCycle	Simple firmware example to demonstrate use of the general purpose indicator LEDs (D2,D3,D4,D5) on the Development Kit board.
pingnak	Contains firmware to perform bulk loopback of EP2OUT to EP6IN and EP4OUT to EP8IN using the PING NAK interrupt to initiate the transfer. Use the windows application cybulk.exe to drive the firmware.
vend_ax	Contains the source files to build a vendor specific command sample. The purpose of this software is to demonstrate how to implement vendor specific commands. See the readme.txt file for details.
memtest	Memory test firmware example. Tests on-chip and external RAM.

2.5 Hardware

The subdirectories located within this directory contain schematics, GAL code, and design files for the Development Board.

2.6 Help

Contains Windows Help files for various DVK components. These files may be duplicated elsewhere in the installation. There are also links to the help files under Cypress in the Windows Start Menu.

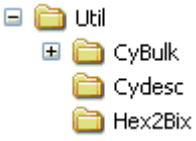
2.7 Target

The following subdirectories located within this directory contain files and utilities for building the EZ-USB Firmware FrameWorks. These files are described in detail in the section EZ-USB Firmware FrameWorks.

Fw	Firmware FrameWorks 8051 source code. Copy these files to a new directory to begin your device firmware.
Inc	C and assembly header files for EZ-USB. These files contain register declarations, macros, and function prototypes for the Firmware FrameWorks.
Lib	Contains the compiled EZ-USB Library and source code.
Monitor	Contains alternate versions of the Keil debug monitor to use either serial port for debug.

2.8 Util

This directory contains the Windows source files for various utilities that are part of the DVK installation. To compile these applications you will need Microsoft Developers Studio 6.0 or later.



2.9 Windows

In order to simplify first-time plug-in of the development board hardware, the DVK installer pre-copies several files to the Windows directory tree. Depending on the version of Windows installed on your system, the actual name of the root Windows directory may be different (e.g. \Windows, \WinMe, \WinNt, etc.). In the following descriptions the Windows folder will be referred to generically as \Windows.

When Windows encounters a USB device for the first time, it must identify the device and determine

the appropriate driver to load for that device. Devices are identified by an INF file that will tell Windows which driver to install. Upon first time device plug-in, Windows will invoke the "Add New Hardware Wizard". If the device has built in support within Windows, user interaction is not required. This would occur for a device in a Windows supported device class (such as a HID-class mouse) or with a custom device that Microsoft has chosen to support in the Windows distribution. All other device plug-ins will usually result in user interaction to point Windows to the location where the INF and driver files for the device are located. This could be at some location on the hard drive or on removable media such as a CD.

The DVK installer pre-copies the INF and driver files to avoid user interaction with the "Add New Hardware Wizard". Without this pre-copy, you would need to direct Windows to the appropriate directory in the DVK installation tree: c:\cypress\usb\driver\cyusb by default. In order to avoid all user interaction, the INF files that are pre-copied are slightly different than the standard plug-and-play INF file that would be distributed on removable media. Specifically, the "copyfiles" section of the INF files have been removed, since the necessary files have already been pre-copied. The special versions of the INF files are identical to the same files in the DVK driver directory, except that the "copyfiles" section has been commented out, and "pre" has been appended to the filename.

Be aware that pre-copy of driver files needs to be handled differently for end-user devices. Microsoft provides an API for correctly pre-copying driver files. The DVK installer does not use these APIs. Consult the Windows DDK documentation for more information on driver distribution and Setup APIs.

Files pre-copied to the Windows tree:

\windows\inf\cyusbpre.inf	INF file for the default FX2LP and FX1 VID/PID. Also for the VID/PID of the example firmware included in the DVK.
\windows\inf\cymonpre.inf	INF file for the FX2LP and FX1 Development Boards. This INF file automatically loads the Keil debug monitor using the driver script feature.
\windows\system32\drivers\cyusb.sys	The Cypress Generic USB Device Driver binary
\windows\system32\cymon\mon.spt	The monitor script file used by cymonpre.inf. This file contains the script to download the Keil debug monitor to the development board.

3 EZ-USB Firmware Frameworks

The Firmware FrameWorks simplifies and accelerates USB peripheral development using the EZ-USB chip. The FrameWorks implements 8051 code for EZ-USB chip initialization, USB standard device request handling, and USB suspend power management services for the user. The user simply provides a USB descriptor table, and code to implement the peripheral function to complete a fully compliant USB device. The FrameWorks provides function hooks and example code to help with this process. The FrameWorks uses the EZ-USB Library to carry out common functions and for EZ-USB register definitions. The Library is also documented in this section.

Most of the firmware examples in the EZ-USB DVK are based on the FrameWorks.

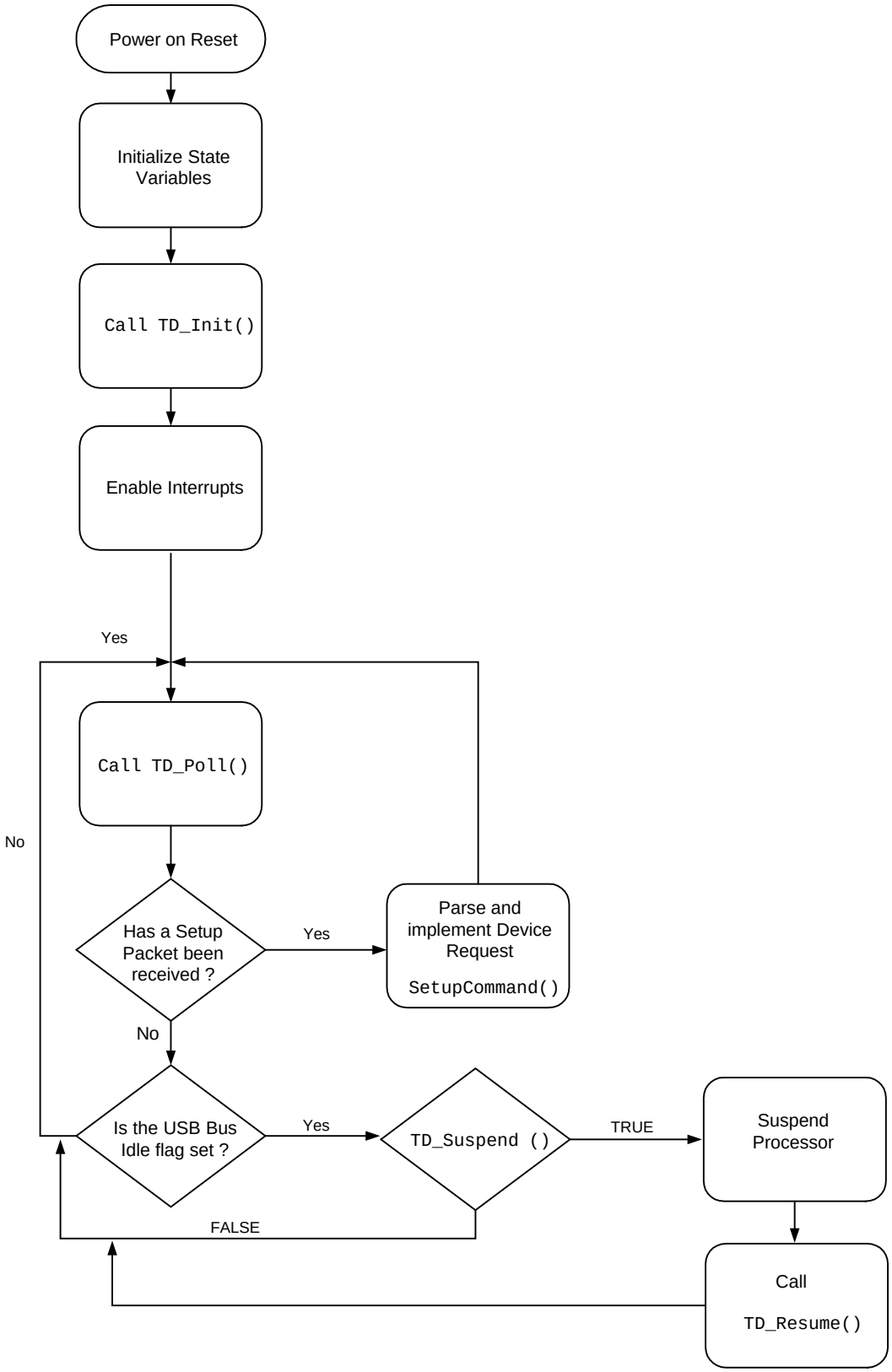
3.1 FrameWorks Overview

The FrameWorks implements 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 (USB spec Chapter 9) device without writing a line of code. By linking code associated with the provided hooks, it is possible to incrementally build a fully functional device.

The FrameWorks implements a simple co-operative tasking executive (See figure below). At start-up, the FrameWorks first initializes all of its internal state variables. It then calls the user initialization function `TD_Init()`. Upon return, the FrameWorks initializes the USB interface to the unconfigured state and enables interrupts. The firmware then ReNumerates and starts the cooperative 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 handles the standard USB requests but allows 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().

EZ-USB USB interrupts are handled by the FrameWorks. It provides hooks for user code notification of USB events.



3.2 Building the FrameWorks

The FrameWorks was 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 under the c:\cypress\usb\target\fw directory. The sub-directory is chip dependent. For FX2LP and FX1, the firmware is located in the "LP" sub-directory. Before editing the firmware, you should create a new directory for your project and copy the various FrameWorks source files into it.

The Keil tools include both DOS command line tools and a development IDE. You can compile your code using either an MS-DOS batch file or a uVision2 Project. The DVK includes examples of both methods. The tutorial section of this guide demonstrates building a project using the uVision2 IDE. If you plan to use the command line tools to build firmware, you must set up a few environment variables and add the Keil tools to your path. After starting the Windows Command Prompt, run setenv.bat (located in the bin directory) to setup 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:

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 will 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	Head file containing common EZ-USB constants, macros, data types, and library function prototypes
FX2REGS.H	EZ-USB register declarations and bit mask constants
SYNCDLY.H	Contains the synchronization delay macro.
EZUSB.LIB	EZ-USB Library object code. See the EZ-USB Library section 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

3.3 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. Each section following contains a complete list of functions and their descriptions.

3.3.1 Task Dispatcher Functions

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

3.3.1.1 TD_Init()

```
void TD_Init()
```

This function is called once during the initialization of the FrameWorks. It is called prior to ReNumeration and the Task Dispatcher starting. It is intended for user global state variable and device initialization.

3.3.1.2 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 may be completed prior to returning from this function. However, failure to return from this function will prevent 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()`.

3.3.1.3 TD_Suspend()

```
BOOL TD_Suspend()
```

This function is called prior to the FrameWorks entering suspend mode. This function should contain code that places the device in a low power state and returns TRUE. However, the user code may prevent the FrameWorks from entering suspend mode by returning FALSE.

3.3.1.4 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 may resume full power operation.

3.3.2 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.

3.3.2.1 DR_GetDescriptor()

```
BOOL DR_GetDescriptor()
```

Description: This function is called prior to the FrameWorks decoding and implementing the GetDescriptor device request. The register array SETUPDAT contains the current eight byte setup command. It may be parsed by the user's code to determine which Get Descriptor command was issued. If TRUE is returned, the FrameWorks will parse and implement the command. If FALSE is returned, it will do nothing.

3.3.2.2 DR_GetInterface()

```
BOOL DR_GetInterface()
```

Description: This function is called prior to the FrameWorks implementing 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.

3.3.2.3 DR_SetInterface()

```
BOOL DR_SetInterface()
```

Description: This function is called prior to the FrameWorks implementing 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.

3.3.2.4 DR_GetConfiguration()

```
BOOL DR_GetConfiguration()
```

Description: This function is called prior to the FrameWorks implementing 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.

3.3.2.5 DR_SetConfiguration()

```
BOOL DR_SetConfiguration()
```

Description: This function is called prior to the FrameWorks implementing 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 (See the section on FrameWorks optimization for more details). It then configures the EZ-USB 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.

3.3.2.6 DR_GetStatus()

```
BOOL DR_GetStatus()
```

Description: This function is called prior to the FrameWorks implementing 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.

3.3.2.7 DR_ClearFeature()

```
BOOL DR_ClearFeature()
```

Description: This function is called prior to the FrameWorks implementing 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.

3.3.2.8 DR_SetFeature()

```
BOOL DR_SetFeature()
```

Description: This function is called prior to the FrameWorks implementing 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.

3.3.2.9 DR_VendorCmnd()

```
void DR_VendorCmnd()
```

Description: This function is called when the FrameWorks determines 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.

3.3.3 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, consult the **Interrupts** chapter in the **EZ-USB Technical Reference Manual**.

3.3.3.1 ISR_Sudav()

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

Description: This function is called upon reception of the Setup Data Available interrupt. This function needs to set GotSUD to TRUE so that the device request handler can process the SETUP command.

3.3.3.2 ISR_Sof()

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

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

3.3.3.3 ISR_Ures()

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

Description: This function is called upon reception of the USB Reset interrupt. In your custom code, you should 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.

3.3.3.4 ISR_Susp()

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

Description: This function is called upon reception of 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.

3.3.3.5 ISR_Highspeed()

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

Description: This function is called upon reception of the USB HISPEED interrupt. In your custom code, you should 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.

3.4 EZ-USB Library

The EZ-USB Library is an 8051 .LIB file that implements functions that are common to many firmware projects. Typically, there is no reason to modify these functions so they are 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 of the EZ-USB registers.

The source code and the compiled .LIB file are located in the \cypress\usb\target\lib\lp directory.

3.4.1 Building the Library

There is usually no need to modify or rebuild the EZ-USB Library, however, this section documents the build process if you should choose to do so.

Only the full retail version of the Keil tools can build library files. The evaluation version will not build this library.

You must build the library using the command-line version of the Keil tools. Before building the library you must set up a few environment variables and add the Keil tools to your path. After starting the Windows Command Prompt, run setenv.bat (located in the bin directory) to setup the build environment. This batch file assumes that you have installed the DVK and Keil tools in the default directories. To build the library simply 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 EZ-USB Manual for more information on autovector interrupts.

3.4.2 Library Functions

3.4.2.1 EZUSB_Delay()

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

Description: 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.

3.4.2.2 EZUSB_Discon()

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

Description: 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 EZ-USB 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 EZ-USB serial interface engine handles most of the USB device requests on endpoint 0.

3.4.2.3 EZUSB_GetStringDscr()

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

Description: 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.

3.4.2.4 EZUSB_Susp()

void EZUSB_Susp(void)

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

3.4.2.5 EZUSB_Resume()

void EZUSB_Resume(void)

Description: 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 was result of a USB resume or a remote wake-up, and generates the K-state accordingly.

3.4.2.6 I2C 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 I2C devices such as the EEPROM, seven-segment display and buttons on the DVK board. See the vend_ax and dev_io firmware examples for usage of these functions.

4 Firmware Download Drivers

A unique feature of the EZ-USB family is the ability to dynamically change device personality through firmware download and ReNumeration. An EZ-USB based device that uses this feature uses a small I2C EEPROM to store a unique Vendor ID (VID) and Product ID (PID). This VID/PID combination is bound to a specific device driver on the host system whose only function is to download firmware to the device. This section describes such a driver.

The Cypress Generic USB Driver has the ability to playback a script of USB commands upon device startup. This feature is used to automate the firmware download. The .INF file for a device specifies the filename and location of a file that contains a download script for the device firmware. The DVK includes a tool called CyScript that will convert a firmware .HEX file into a download script file.

A sample firmware download driver is located in the **\cypress\usb\drivers\cyload** directory. In general, the user will make a copy of this directory and then customize it for their device.

Files in the CyLoad sample:

cyload.inf	.INF file that automates the firmware download
cyload.sys	The download device driver for Windows XP/2000. This is just the Cypress Generic USB Driver renamed.
cyloadme.sys	The download device driver fow Windows ME. This is just the Cypress Generic USB Driver renamed.
cyload.spt	The download script file. This file was generated by CyScript and is the bulkloop example firmware
cyload.iic	EEPROM image that contains the VID/PID bound to the CyLoad example

The sample .INF file is bound to VID=0x04B4 and PID=0x0084. If you want to experiment with the CyLoad example you will need to reprogram the EEPROM on your Development Board with **cyload.iic**. Use the **S EEPROM** button on the Cypress USB Console EZ-USB interface to program your EEPROM. To return your EEPROM to its default value, repeat the process using **fx2lp_c0.iic** located in **\cypress\usb\examples\fx2lp\EEPROM IMAGES**.

It is important that you completely customize your firmware download driver package to prevent conflicts with the EZ-USB DVK and EZ-USB based devices from other vendors that may be installed on a system. This means that all files in the driver package must be renamed to something unique. Incorporating your company name or an abbreviation of your company name in the filenames is a good way to do this. Also, the VID/PID you use for your device and the Device GUID in your .INF file should also be unique. You should never use the Cypress VID in a distributed product. You can obtain a unique VID for your company from the USB-IF (www.usb.org).

NOTE: It is strongly recommended that you test and debug driver loading on a "clean" system that can be easily restored. Editing of INF files is notoriously difficult and error prone. A single formatting error can prevent the .INF file from working correctly. Further, once an error has occurred it may prevent subsequent installations from working, even after you correct the error. Do a clean installation of the target operating system and then use partition cloning software such as Norton Ghost (from Symantec) to create a backup copy of your "clean" system. Cypress support will find it difficult, if not impossible, to aid in debugging driver installation problems on "dirty" systems.

1. Develop and debug your firmware
2. Convert your firmware hex file to a Cypress USB Script file
3. Test your script using the script playback button on the Cypress USB Console
4. Copy the entire contents of cyload to another directory
5. Pick a name. In the example, the name is cyload. The name can be anything, but should say

something about your product. For example, if this is the firmware download for a gizmo, you could choose gizmoload.
6. Rename all of the files in your copy of the cyload directory, replacing cyload with whatever name you have chosen.
7. Edit gizmoload.inf and do a search and replace. Replace all instances of cyload with gizmoload.
8. Generate a new GUID using guidgen. Replace the GUID that is in the [Strings] section of the .INF file with this GUID. Guidgen is a Microsoft application that will generate a guaranteed unique identifier. It ships with Microsoft Developers Studio and with the Microsoft Platform SDK.
9. Customize other strings in the INF file

This tutorial does not attempt to explain the intricacies of INF file formatting. For more information on INF files, consult the Windows DDK Documentation or any of the many books available on WDM programming.

note: It is strongly recommended that you test and debug driver loading on a clean machine that can be easily restored. Editing of INF files is notoriously difficult. A single formatting error can prevent the INF file from working correctly. Further, once an error has occurred it may prevent subsequent installations from working, even after you correct the error. Use partition cloning software such as Norton Ghost (from Symantec) to create a backup copy of your "clean" system.

4.1 Customizing For Your Device

This section will guide you step-by-step in creating a custom firmware download driver package for your device. This tutorial does not attempt to explain the intricacies of .INF file formatting. For more information on .INF files, consult the Windows DDK Documentation or any of the many books available on WDM programming.

1. Develop and debug your firmware (This is the hard part!)

This firmware must ReNumerate. The VID/PID stored in the firmware descriptor must be unique and may not be the same as the VID/PID of the pre-ReNumerated device. The pre-ReNumerated VID/PID is stored on the I2C EEPROM.

2. Convert your firmware .HEX file to a Cypress USB Script file

The DVK installation includes a Windows application called CyScript that makes this step simple. CyScript is located in the **\cypress\usb\bin** directory and in the Windows Start menu under Cypress. By default, this tool will create a script file in the same directory as the selected .HEX file. The script file will have the same filename but with a .SPT extension.

3. Test your script using the Play Script button on the Cypress USB Console

The Load Script and Play Script buttons are located on the CyConsole toolbar. The result of playing your script back should be the same as if you were to download your .HEX file using the CyConsole EZ-USB interface.

4. Copy the entire contents of cyload to another directory

You are about to make changes to the files and you don't want to mess up the originals.

5. Pick a name

In the example, the name is cyload. The name you choose will serve as the prefix for many of the changes to follow. The name can be anything, but should say something about your product. The name should not contain spaces or non alpha-numeric characters. Finally,

although there is no practical limit on the length of the name you should try to keep it to 8 characters or less. As an example, if this is the firmware download driver for a **widget** device, you could choose **wdgtld** (for widget load).

6. **Rename all of the files in your directory, replacing cyload with the name you have chosen**

   As an example, **cyload.inf** becomes **wdgtld.inf** and so on.

7. **Update the labels in your .INF file**

   The cyload example .INF was created to simplify the customization process. A simple search-and-replace will convert for your device. Sticking with the widget example, load **wdgtld.inf** into your text editor. Perform a search-and-replace of the entire file, replacing the string **cyload** with **wdgtld**.

   At this time, you should also perform a search-and-replace for the string **yourcompany**. Replace all instances of this string with your company name. Once again, alpha-numeric only and no spaces.

8. **Generate a custom GUID using guidgen**

   GUID stands for Globally Unique Identifier. Microsoft provides a tool called guidgen that will generate a GUID for you. Guidgen ships with Microsoft Developers Studio and the Microsoft Platform SDK. Generate a GUID and replace the GUID that is in the [Strings] section of your .INF file with this GUID. Windows uses the GUID to identify and track your device. Failure to use a truly unique GUID for each of your devices can lead to many conflicts. This includes your pre- and post-ReNumerated device which may not share the same GUID.

9. **Customize other strings in the INF file**

   There are some additional strings located in the [Strings] section of your .INF file that you should customize for your device. These strings may display during first-time enumeration. You should replace them with strings that describe your device. Like the GUID, the SvcDesc string you choose must be unique. Failure to use a unique SvcDesc can result in conflicts.

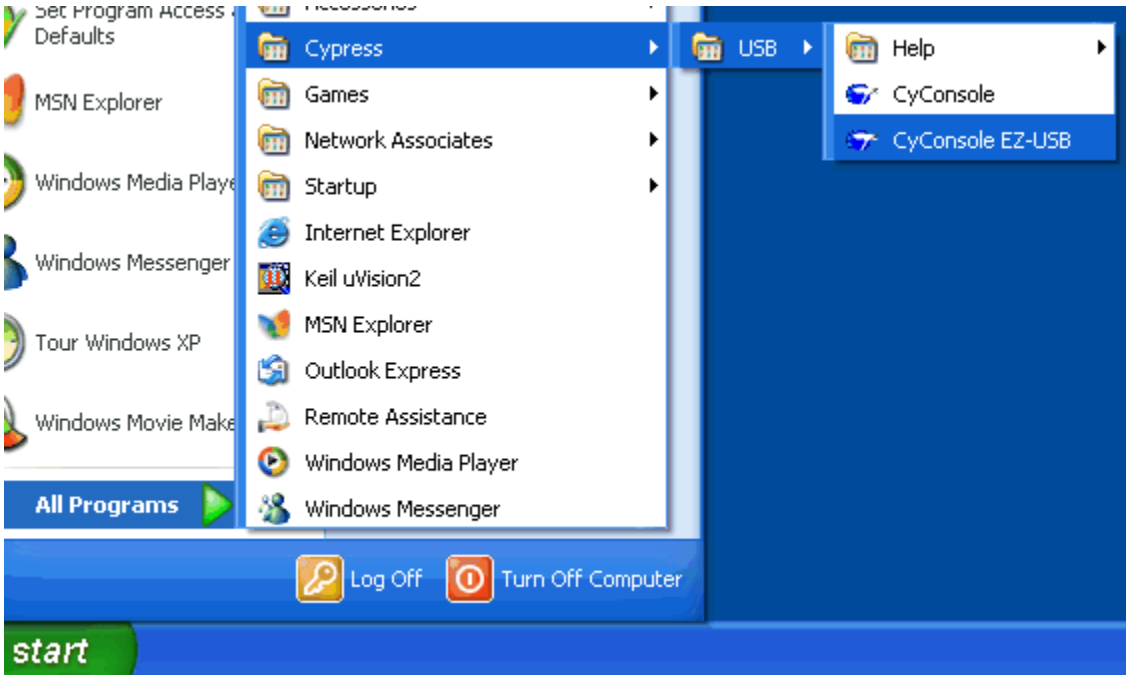
This tutorial was designed to simplify the process of creating the download driver package. There are other advanced methods, such as sharing the same driver for both your pre- and post-ReNumerated device, that are beyond the scope of this tutorial and require a thorough knowledge of INF files. This tutorial does not attempt to explain the intricacies of .INF file formatting. For more information on INF files, consult the Windows DDK Documentation or any of the many books available on WDM programming.

5 Tutorials

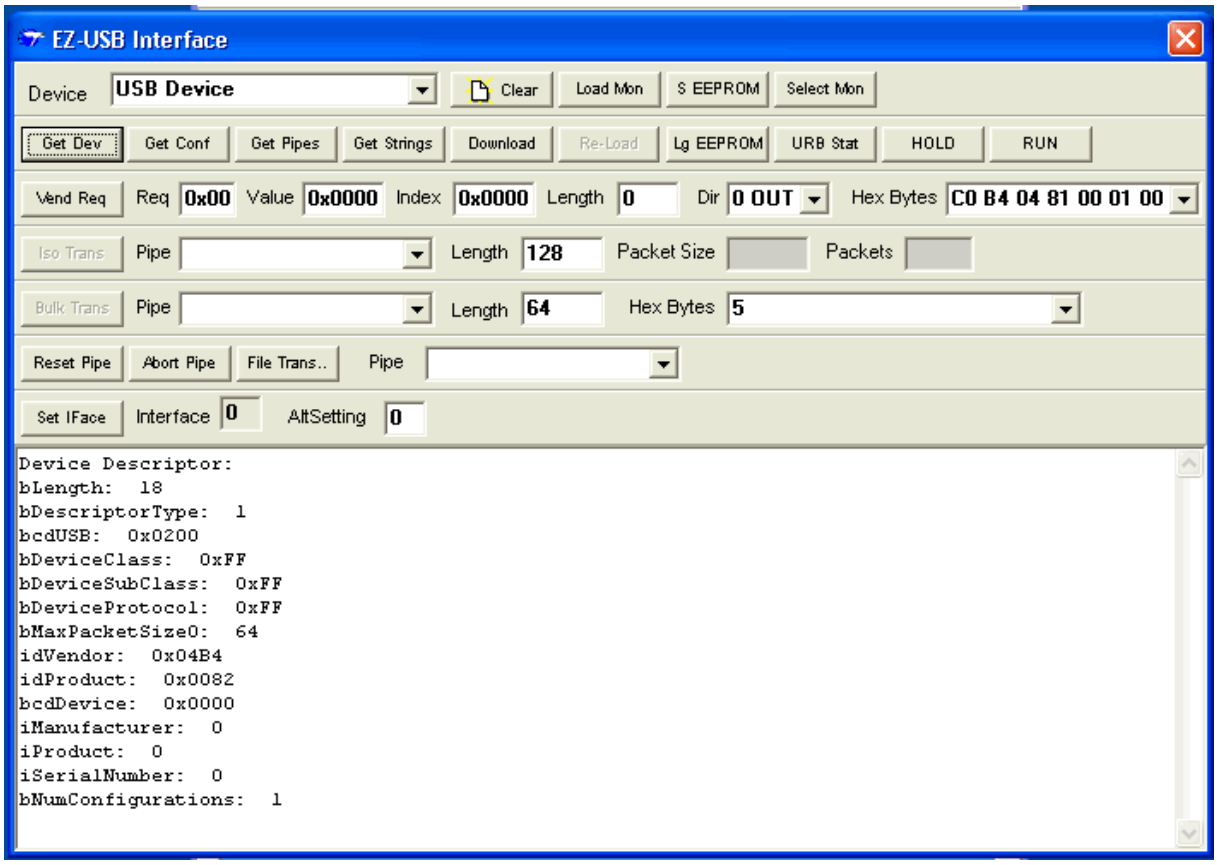
5.1 Verifying Correct Operation - Get Descriptor

Verifying correct operation of the board by doing a "Get Device Descriptor"

1. Plug the EZ-USB Development board into the PC using a USB cable.
2. Start the Cypress USB Console EZ-USB Interface



3. Send a "Get Device Descriptor" operation by pressing the "Get Dev" button. You should get Device Descriptor information back as shown below. Note: the idProduct shown below is for the FX2LP Development Board.

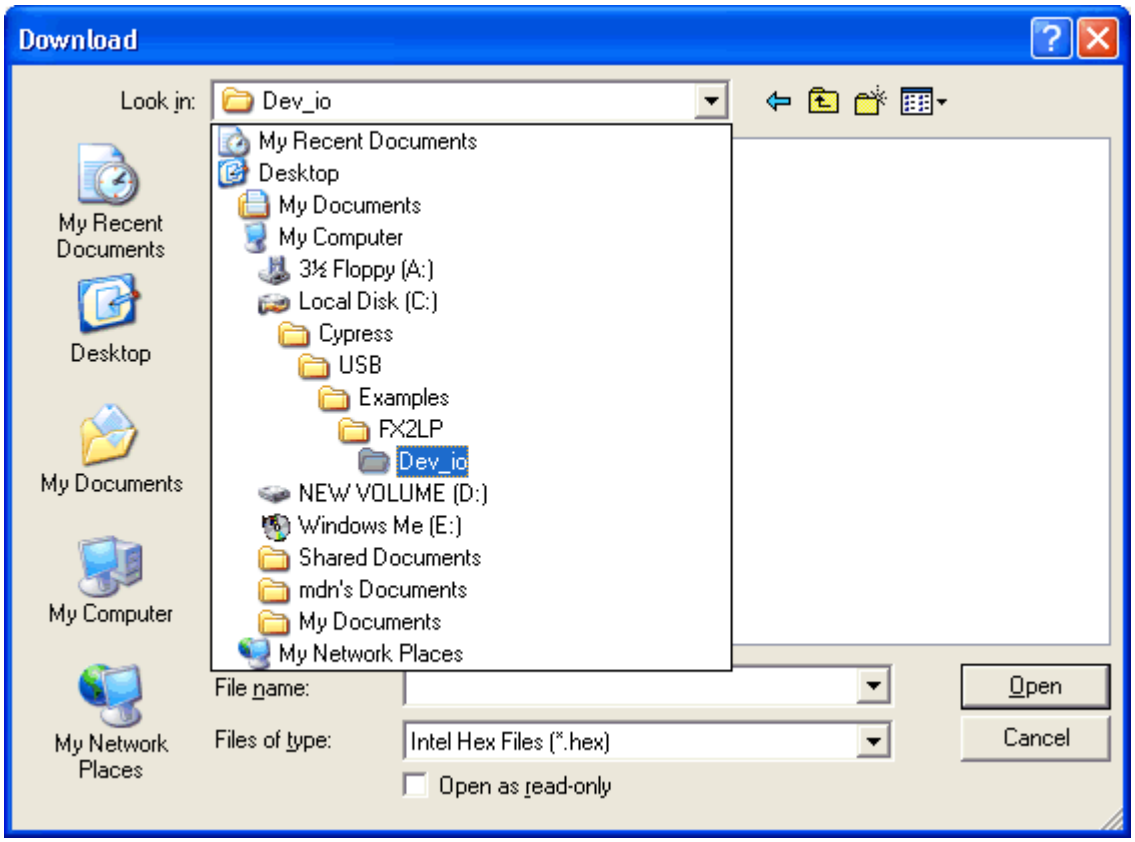


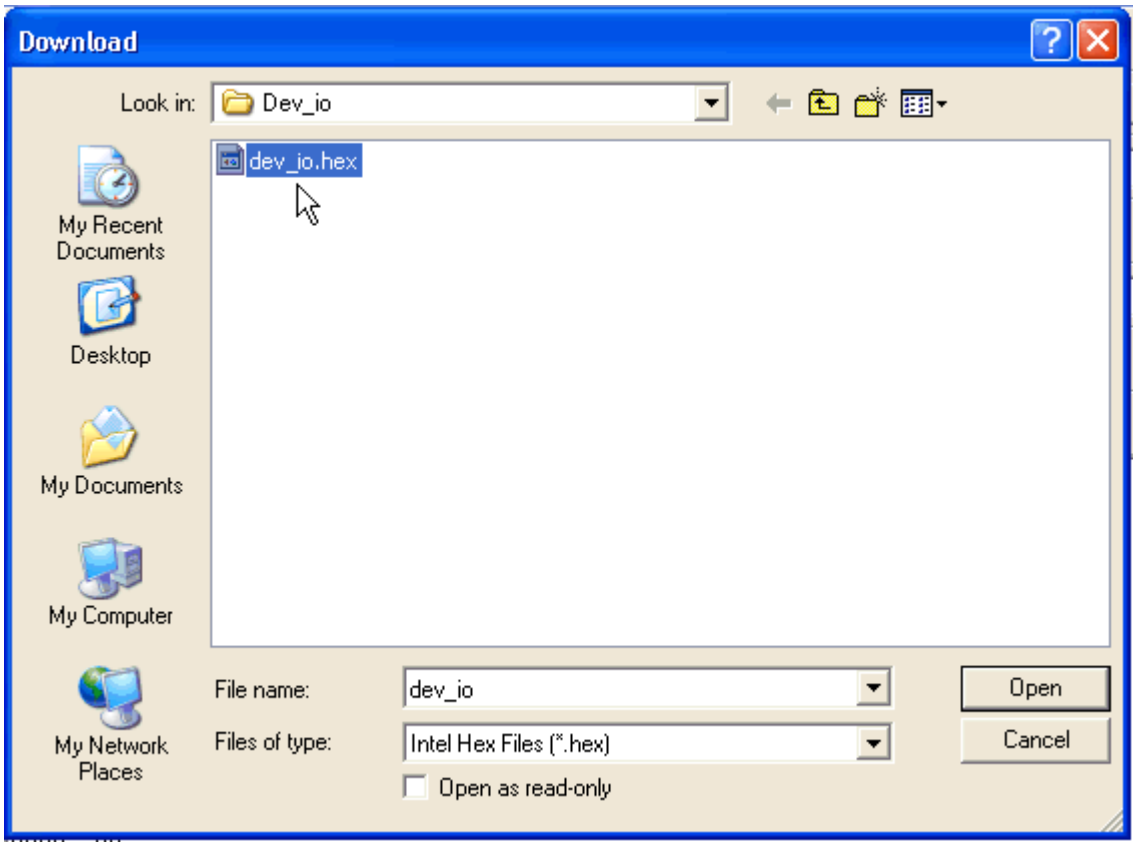
5.2 Running the dev_io example

Running the dev_io example program by loading it from the Cypress USB Console.

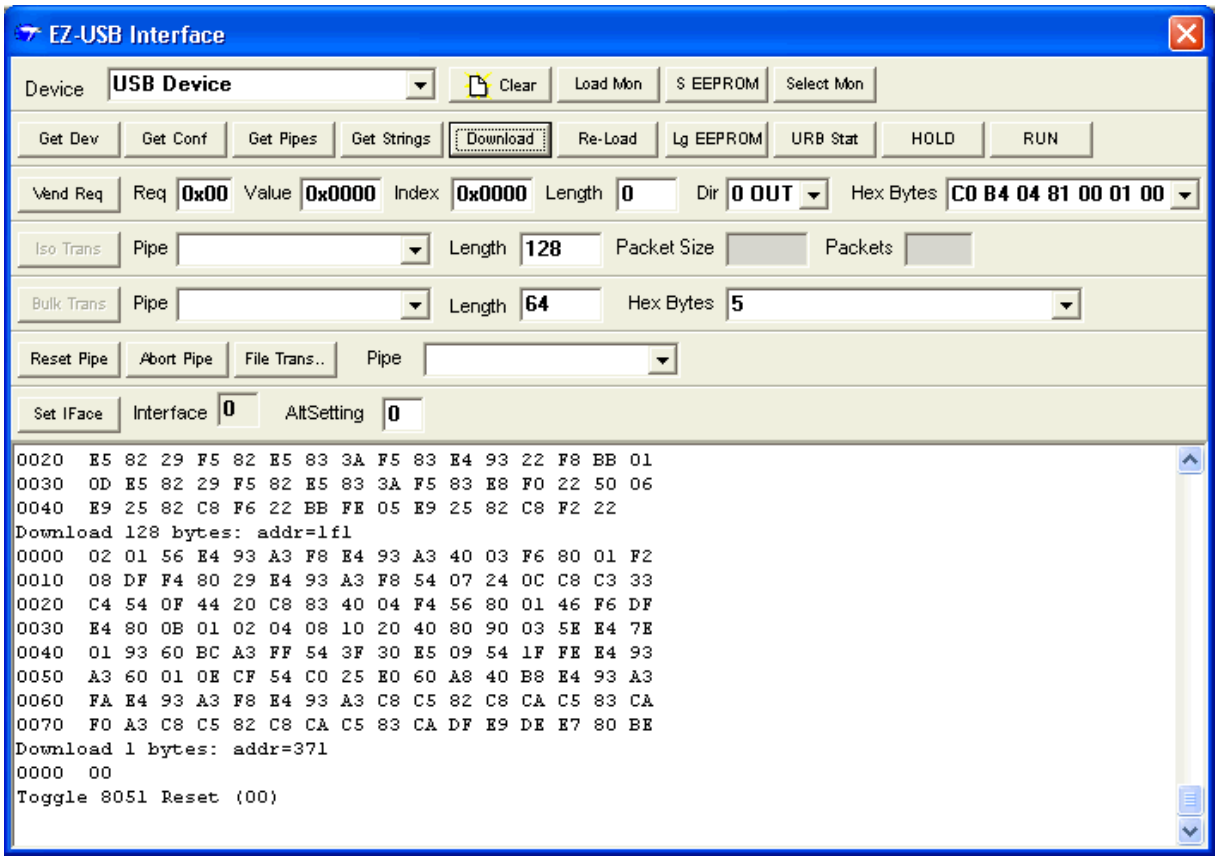
1. Select the "Download…" button to prepare to download a file.

2. Using the Download file selection box, select the dev_io.hex file to download.

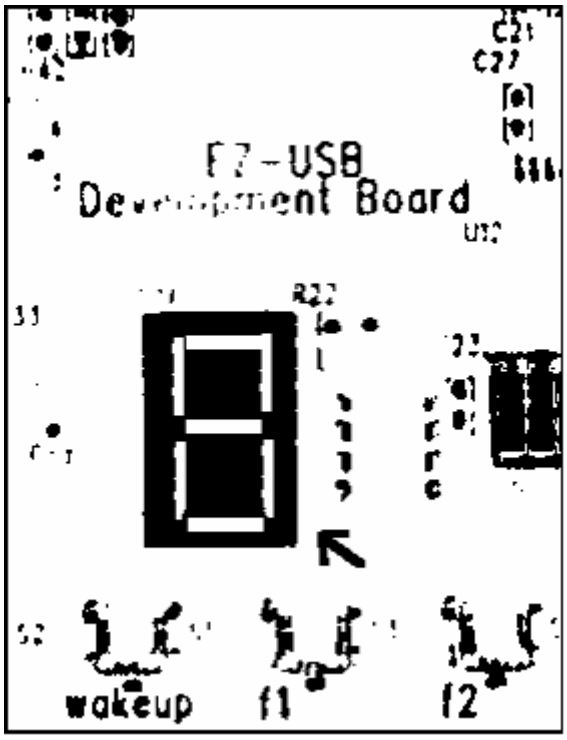




3. You should see the download indication as shown below.



4. At the same time, the seven segment LED should light up on the development board.



5. The F2, F3 keys, when pressed, will count down, and up, respectively. This indicates that the dev_io.hex file is running correctly on the development board.

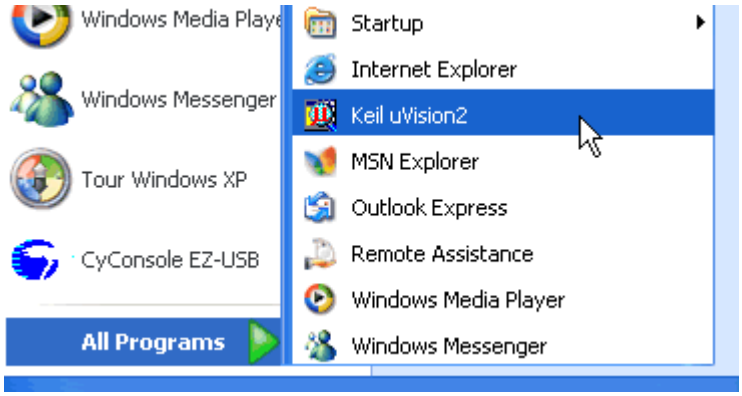
5.3 Debug the dev_io example

Run the dev_io example program by loading it from the Keil debugger.

In the previous example, we used the Cypress USB Console (CyConsole) to load our 8051 program code. This time we will use the Keil debugger to load our 8051 program. This is the Keil debug monitor that was customized to run on the EZ-USB development board. This is necessary if we wish to debug the 8051 code using single step, start, stop, etc. The debug monitor is now automatically loaded into the development board when it is plugged in. It is also possible to use CyConsole to load the monitor. For now, simply unplug and re-plug the USB cable, or press the reset button on the Development Board to reload the monitor.

NOTE: pressing the reset button does not re-initialize the LED, so the LED will stay lit when you press the reset button. Unplugging the board and plugging it back in will reset the LED.

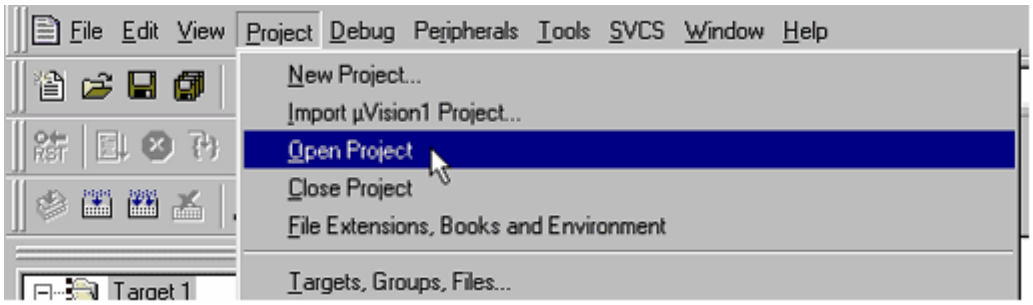
1. You should see that the monitor is loaded by verifying that the green BKPT/Monitor light is lit on the development board. Once the monitor has started, start the Keil uV2 IDE by selecting "Start\Programs\Keil uVision2"

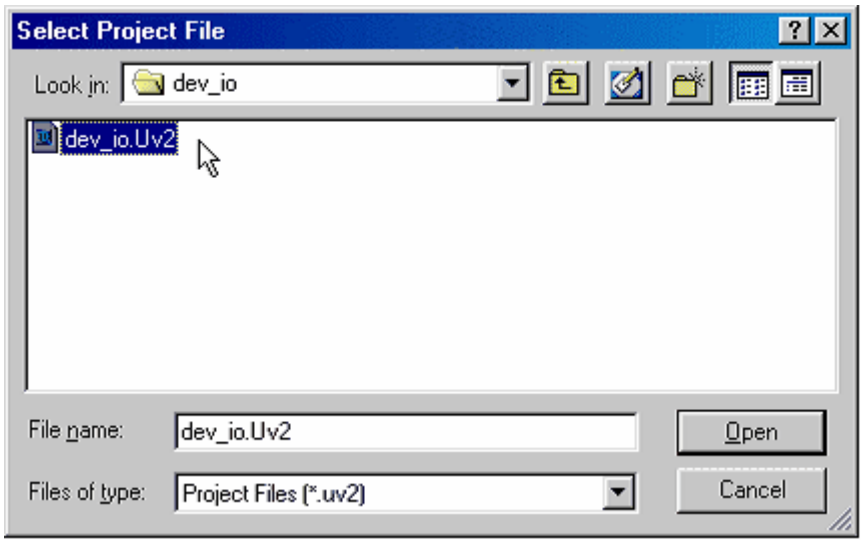


2. Now, with Keil uV2 running, we will want to establish a serial port connection from the Keil debugger to the EZ-USB Development board. Use a standard serial cable to connect your PC to the development board. At this time, there are two serial ports on the development board, labeled "SIO-0", and "SIO-1" You should connect the serial cable to the "SIO-1" port.

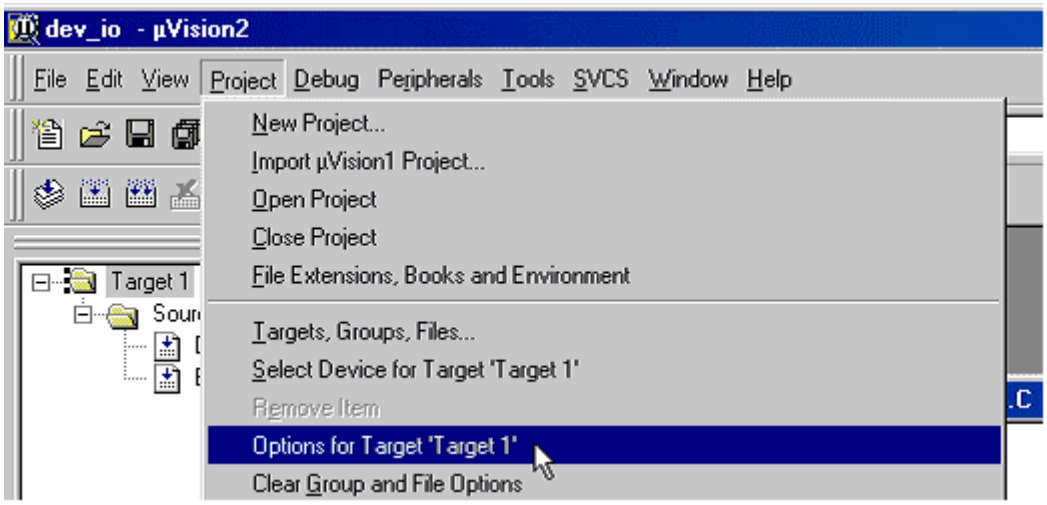
   IMPORTANT NOTE: To establish a connection to the debugger, use the "SIO-1" port.

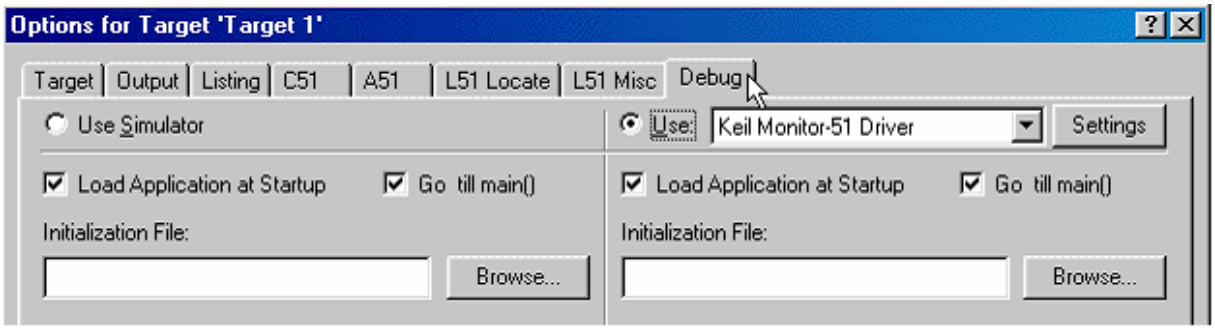
3. Open the dev_io project file as shown below.

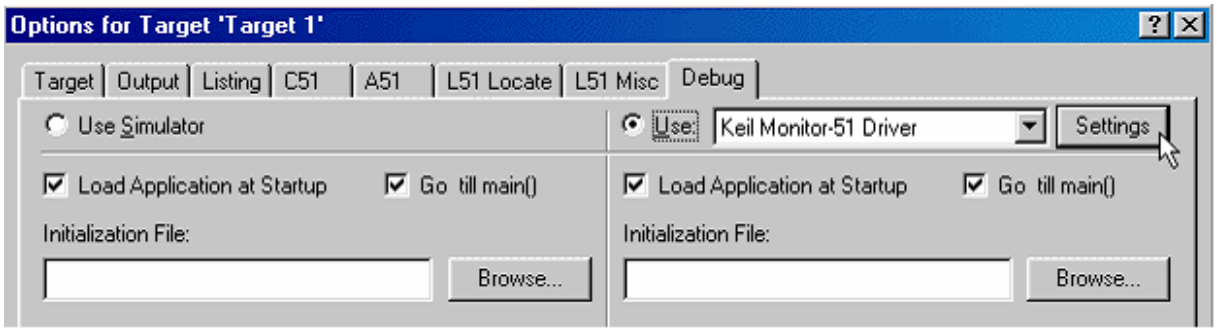


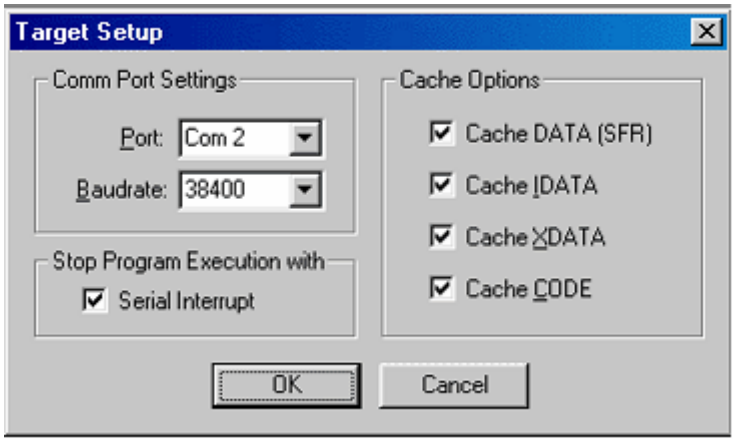


4. Make sure you are using the correct serial port. Make sure baud rate is set correctly. To do this, select "Project\Options for Target 'Target 1'\Debug\Settings".





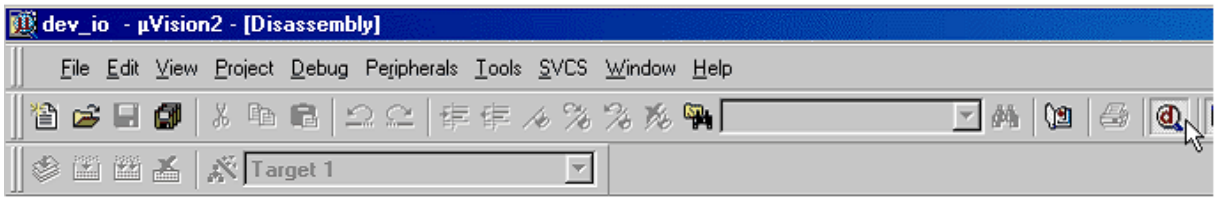




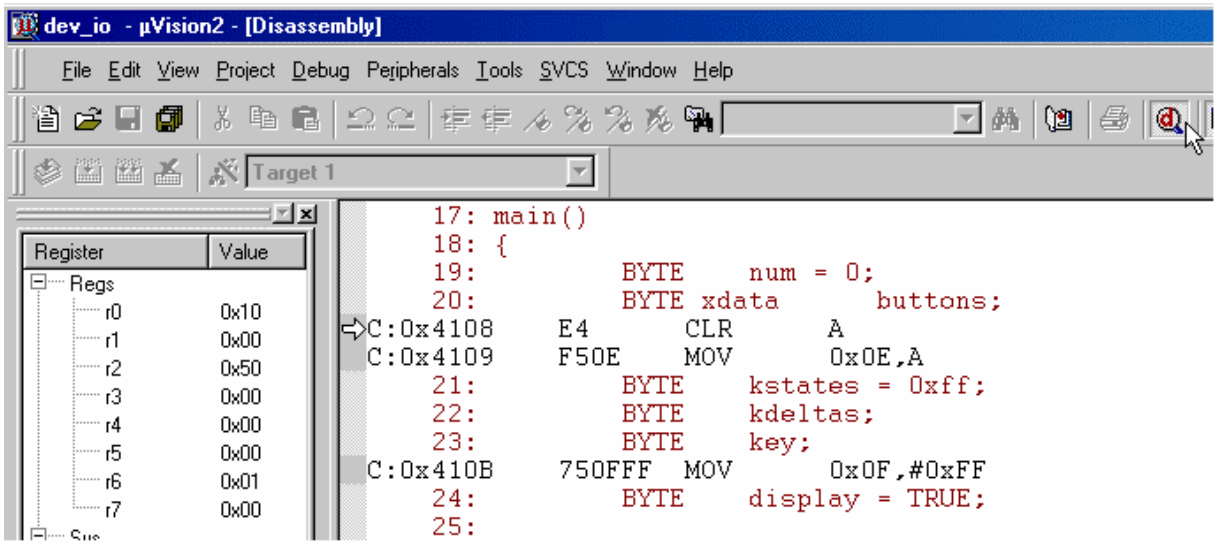
IMPORTANT NOTE: Your PC may have a single serial port. In this case, you would use COM 1 instead of COM2 as shown in the picture above. Check the box labeled "Serial Interrupt"

5. Click OK to close "Options for Target "Target 1" window.

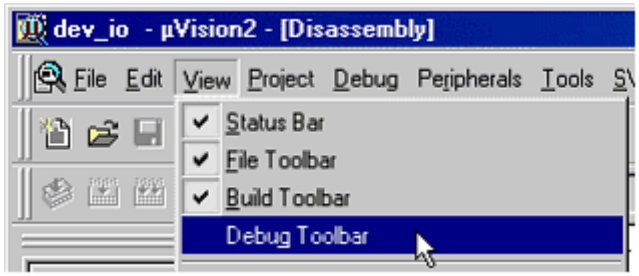
6. Select the "Debug" button on the Keil IDE.



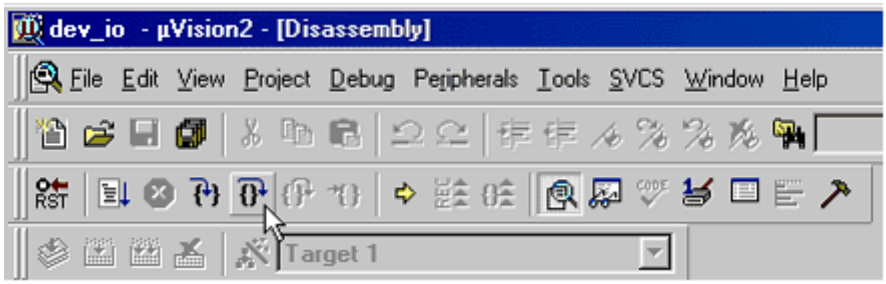
7. You should see the IDE switch to Debug mode, with a yellow arrow indicating the Program Counter location.



8. You can use the "Step Over" button to step through the code by selecting View\Debug Toolbar.



9. Next select the "Step Over" button.

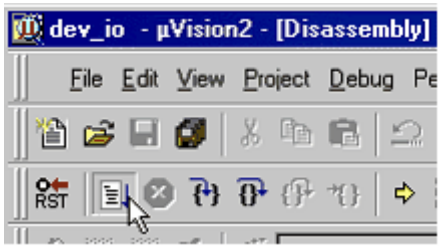


10. View the output window to verify that you are connected to the monitor and that your program has loaded (i.e. it will say something like: Connected to Monitor-51 V3.0).

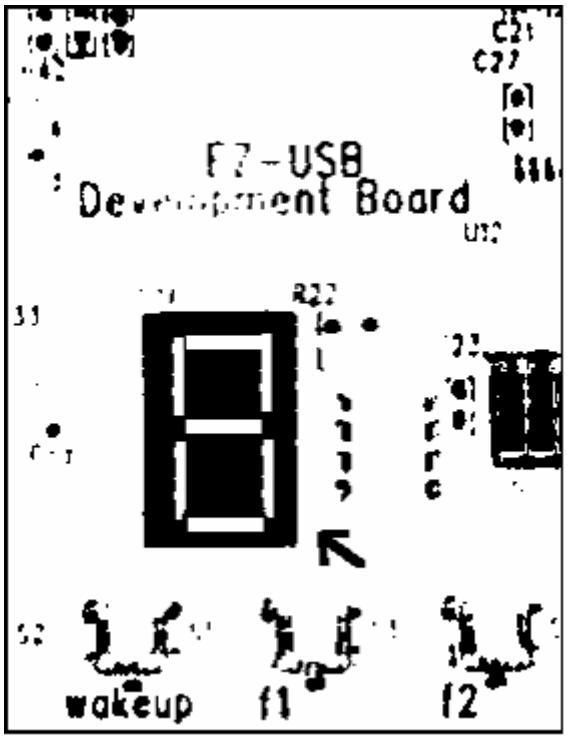
11. Click on the stop button on the debugger (so you can halt the 8051), be sure to check (enable) "Serial interrupt" in the "Target Setup" as indicated above.

PLEASE NOTE: Since the dev_io program uses the iic bus to write to the LED, it uses interrupts. Randomly stopping execution of the 8051 while interrupts are enabled may cause you to stop while servicing an interrupt. If this happens, you will not be able to hit the "Go" button to continue execution; you will have to reset the development board and reload your application.

12. To run the code, press the "Run" button.



13. The dev_io code should now be running on the Development board evidenced by the seven segment LED lighting up on the development board.



14. Click on the "Stop" button to stop the program.

15. Close the Keil IDE.

5.4 Rebuild dev_io example

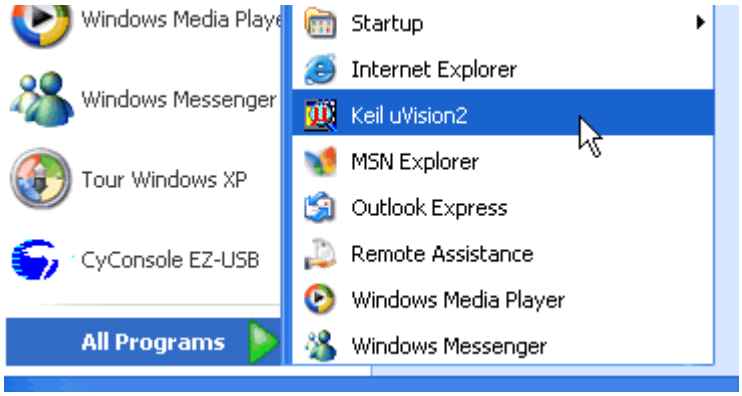
Removing the dev_io target file and rebuilding it using the Keil IDE.

As an exercise, we will now remove the dev_io.hex file and rebuild it using the Keil IDE.

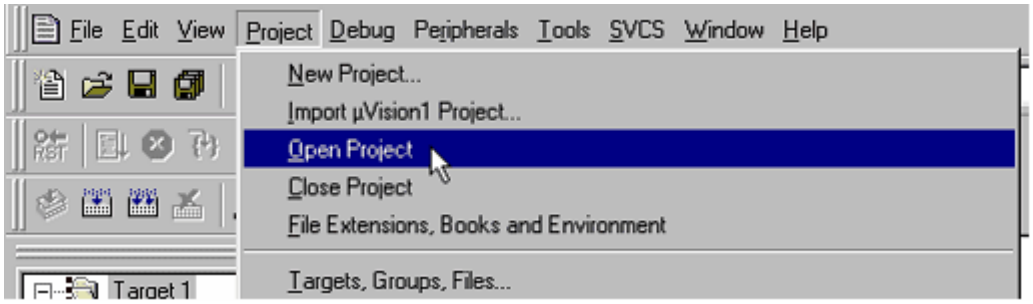
1. Disconnect and reconnect the A – B USB Cable on the development board to clear the memory.

2. Remove (or rename) the dev_io.hex file from the "..\Cypress\USB\Examples\FX2LP\dev_io" directory.

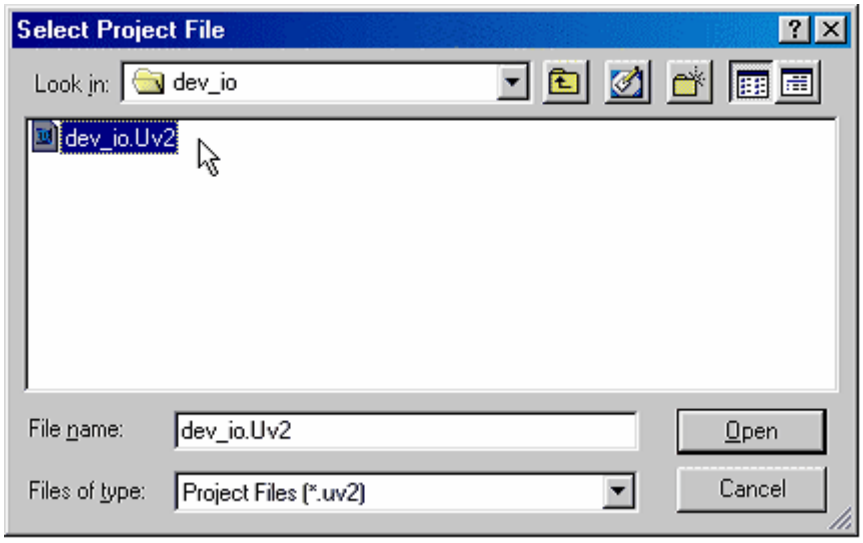
3. Now, to build it using the IDE, run "Start\Programs\Keil uVision2".



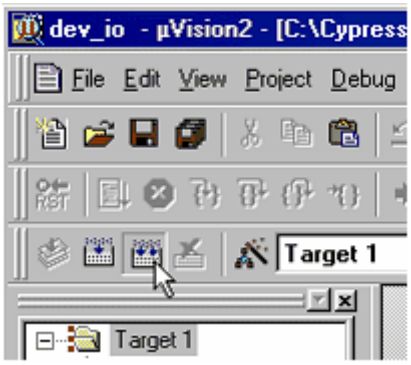
4. Select "Project\Open Project".



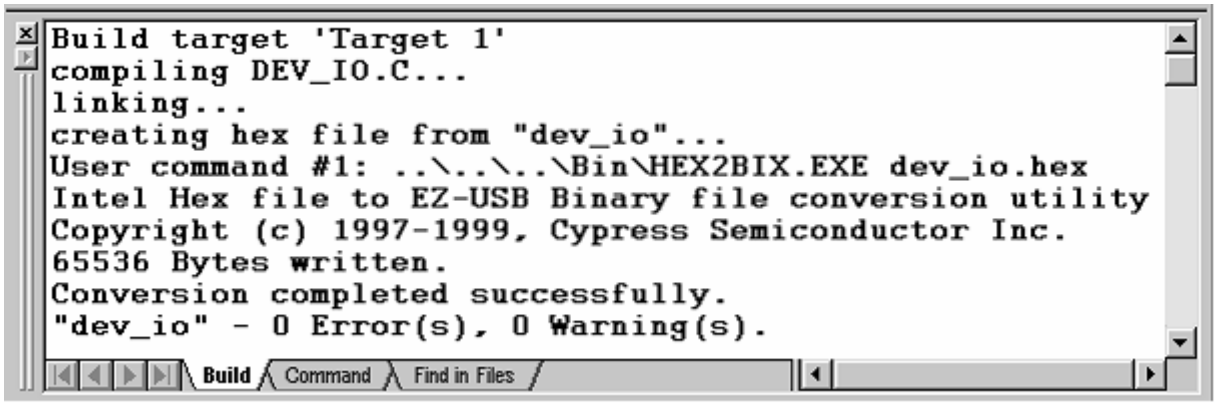
5. Open Project dev_io.Uv2.



6. Select "Rebuild all Project Files" button.



7. The project should complete the make successfully

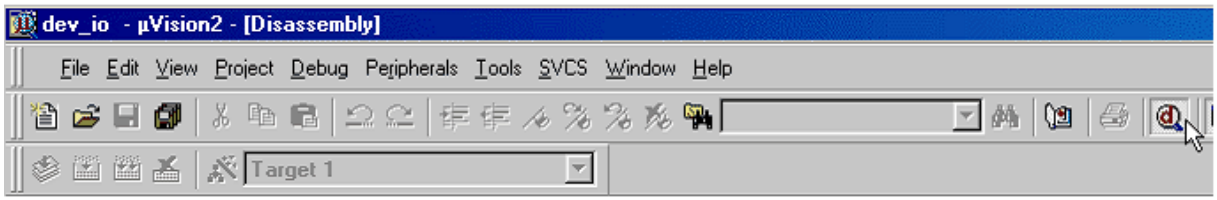


5.5 Setting a debug breakpoint

Running the dev_io target under the debugger and using the debugging capabilities of the IDE.

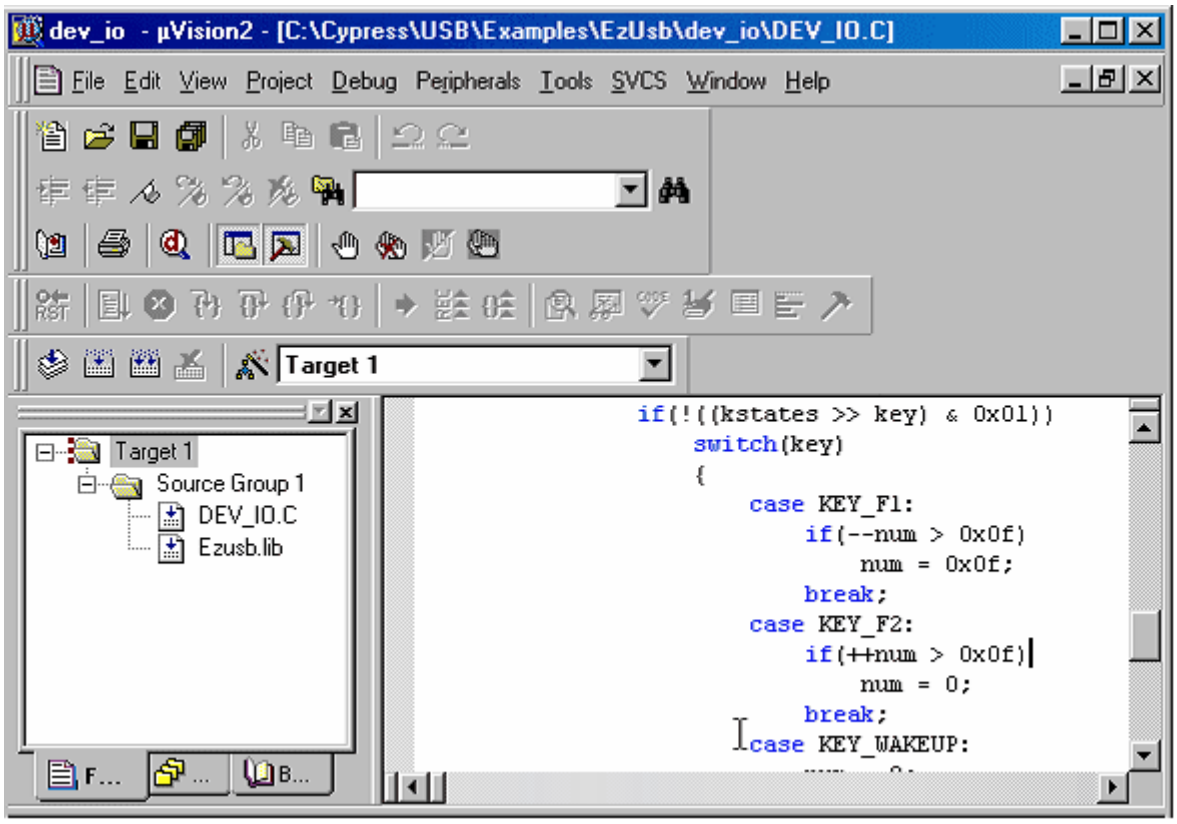
1. Run the newly built dev_io file from the Keil debugger. The LED should be lit after downloading and hitting "Run" (indicating that the application is running).

NOTE: You may have left the Keil debugger running from before.

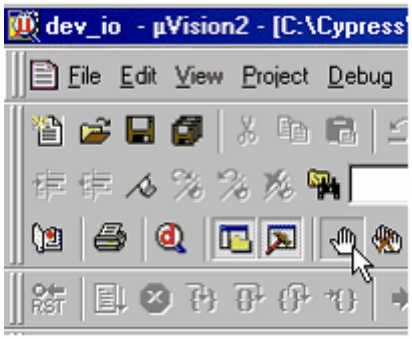


2. Now stop execution of the program using the "Halt" button in the Keil debugger. The "Halt" button appears in the "Debug window" as shown below.

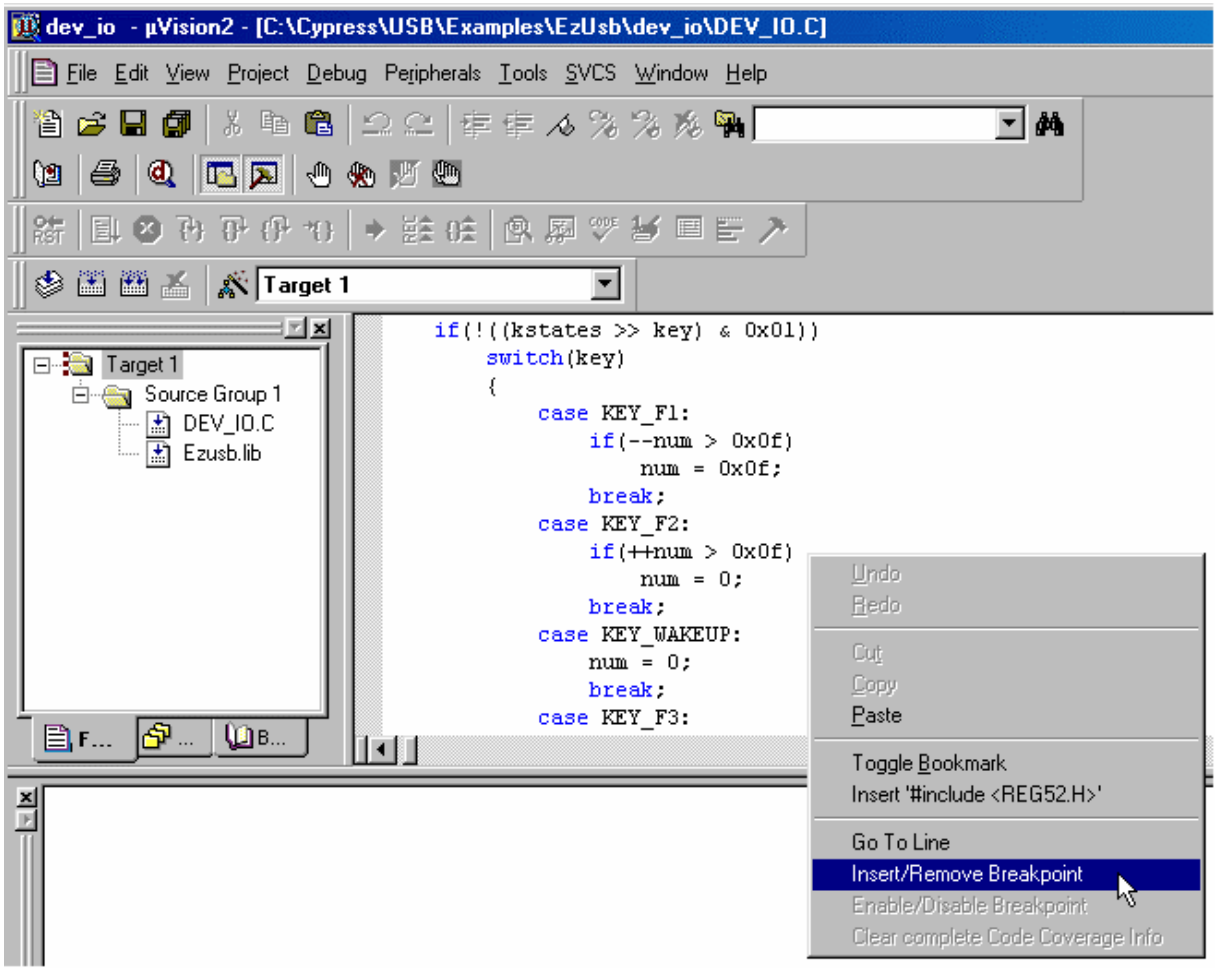
3. Set a breakpoint by selecting the first line in the "case KEY_F2" section (which is in file dev_io.c).



4. Then click on the "Insert/Remove Breakpoint" button.

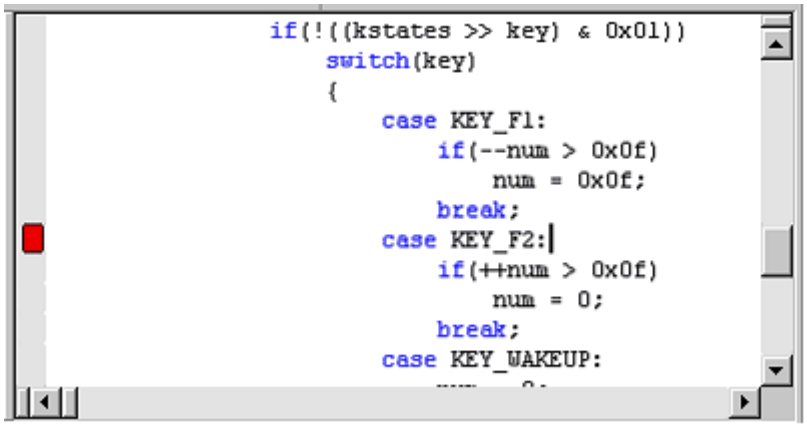


5. Or you may insert a breakpoint by right-clicking and selecting "Insert\Remove Breakpoint" on the first line in the "case KEY_F2" section (which is in file dev_io.c).

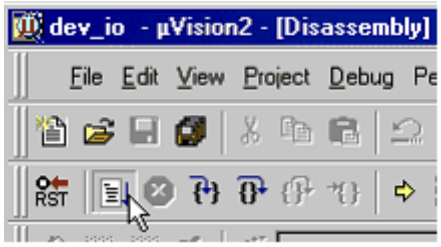


NOTE: You may also set or remove a breakpoint by double-clicking the line.

6. You should see a red breakpoint indication in the margin next to the new breakpoint.

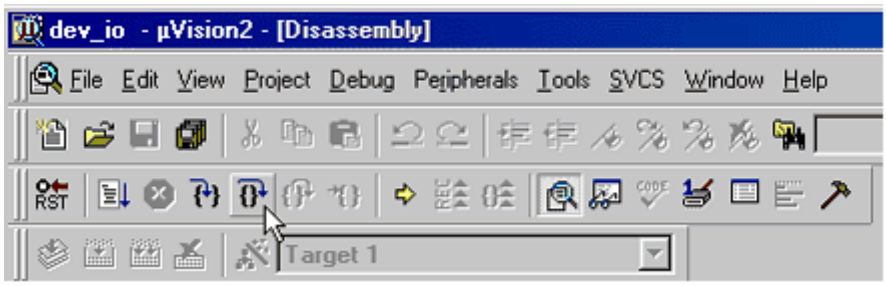


7. Then hit "Run".



8. Now press "F3" on the development board (the "KEY_F2" label equates to the F3 Button). You should see execution of the program halt in the Keil IDE. The LED will not increment as it does normally since program execution has been halted

9. Now press "Step Over".



10. Then press "Run" key on the debugger.

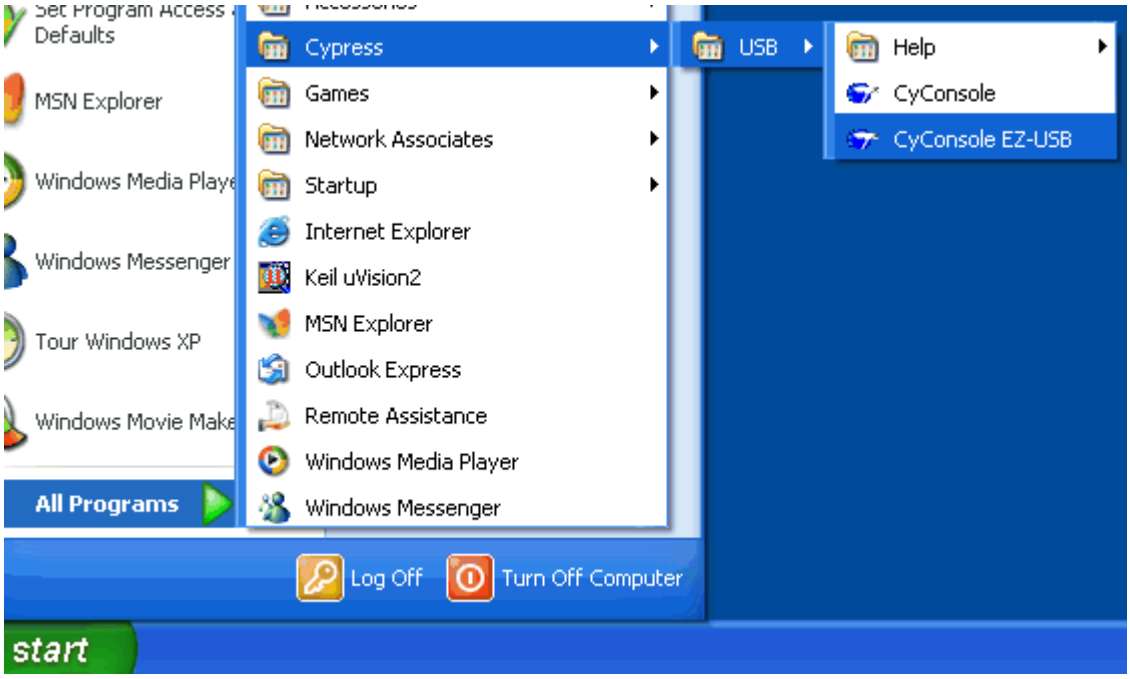
11. Execution will proceed normally until you press the F3 key (On the development board) again. When you are finished, hit the "Stop Debugging" key and exit the Keil debugger.

5.6 Running the bulkloop example

Running bulkloop using CyConsole

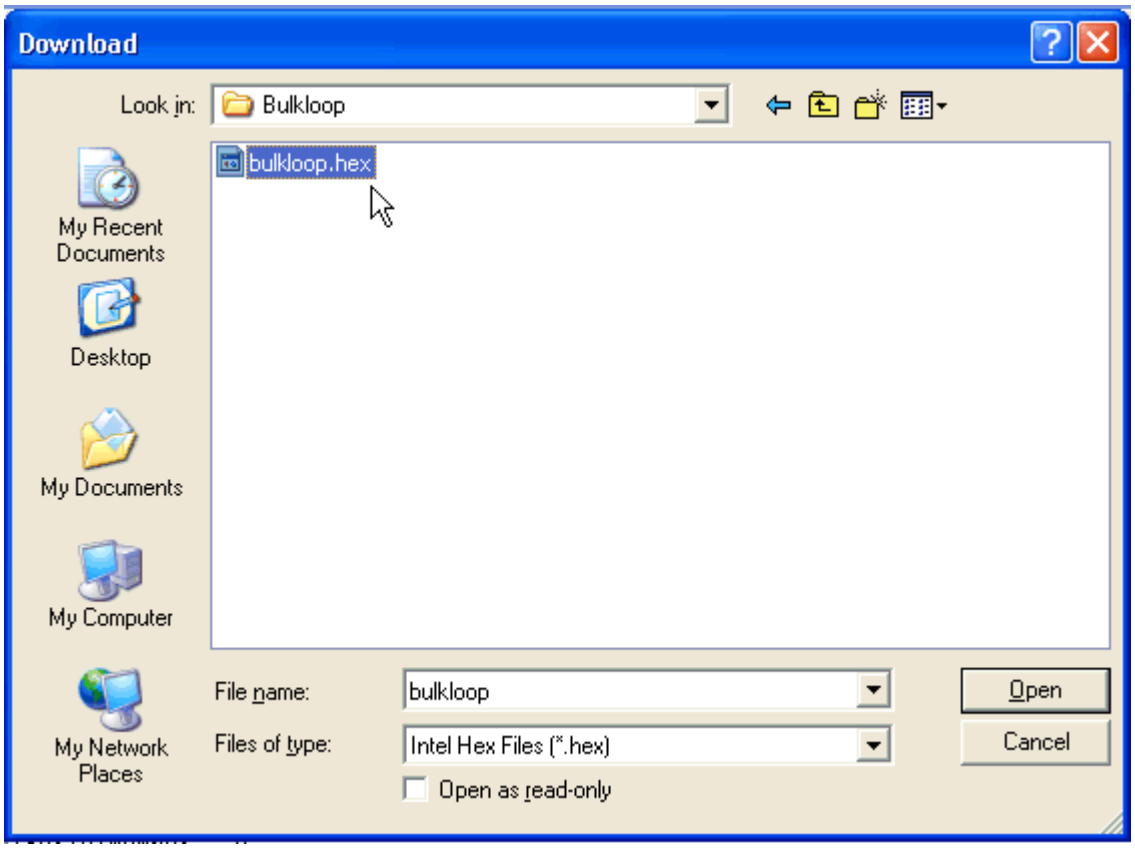
The bulkloop example demonstrates bulk transfers to and from the 8051 target device. It loops back bulk endpoint packets from EP2OUT to EP6IN and from EP4OUT to EP8IN. The readme.txt file in the sample directory describes its usage as does the readme.txt for all the other example programs.

1. Start the Cypress USB Console



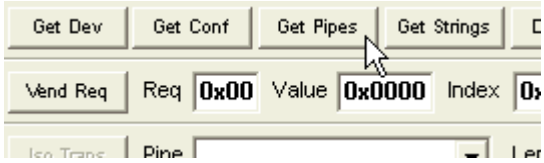
2. Select the "Download…" button.

3. Select the bulkloop example.

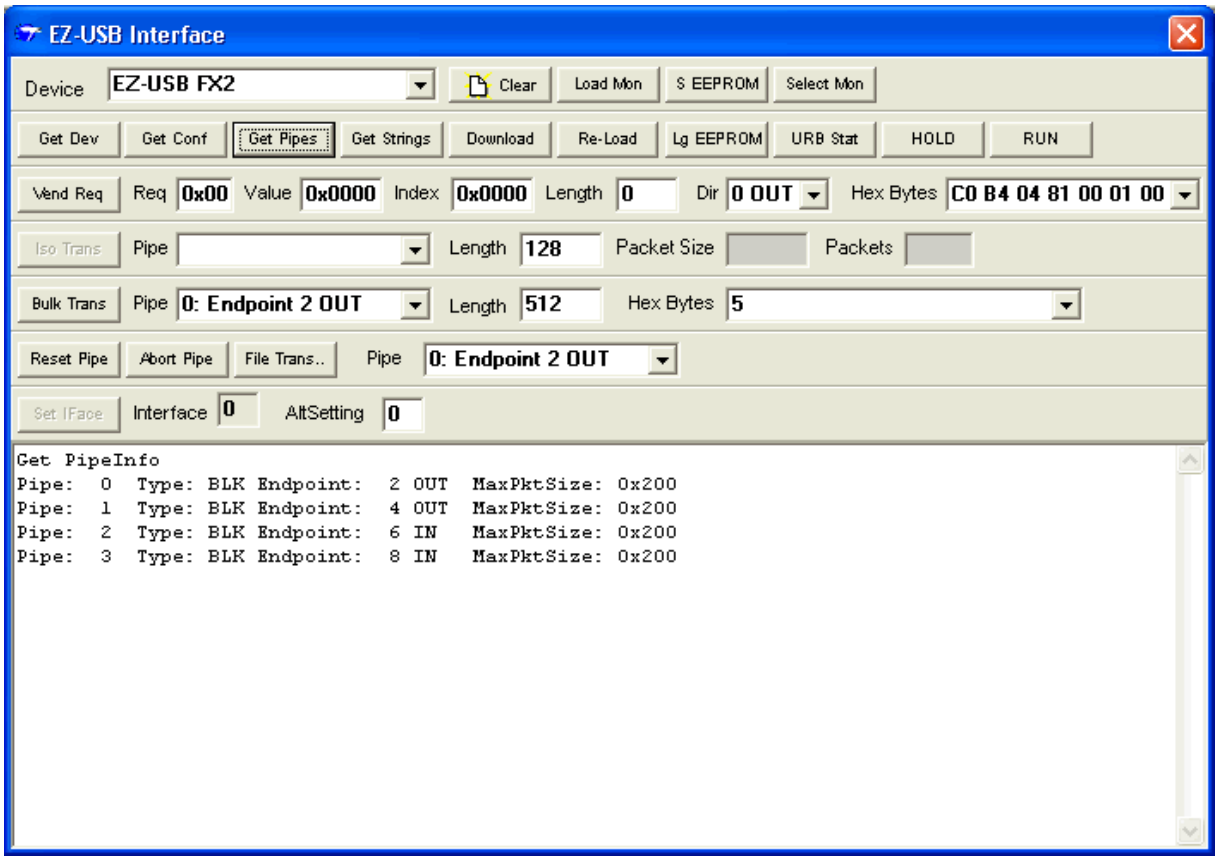


Note: You may select bulkloop.hex (as shown) and then press "Open", or you may simply double-click on the bulkloop.hex icon.

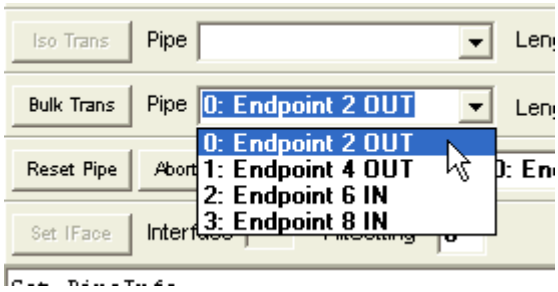
4. After bulkloop is loaded, do a "Get Pipes".



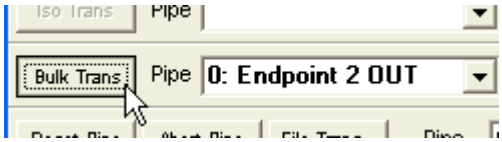
5. Verify the pipes as shown below.



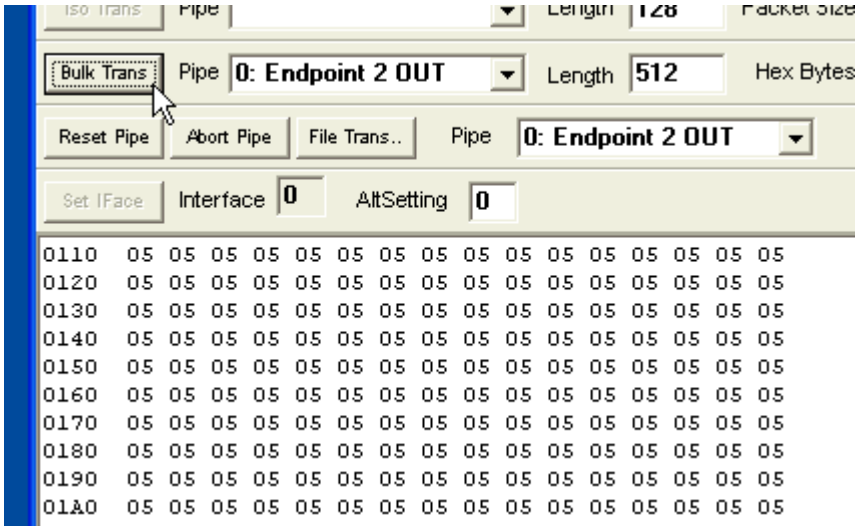
6. Send data out the OUT pipe. Do this by selecting the "Out" endpoint from the pull down menu on the "Bulk Trans" bar (it should have defaulted to this selection).



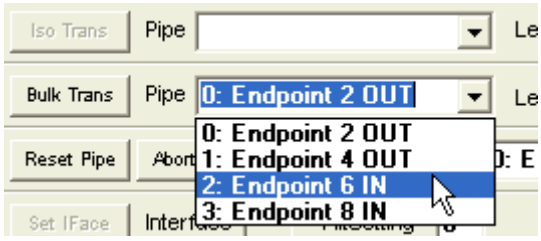
7. Pressing the "Bulk Trans" button will initiate the bulk transfer out.



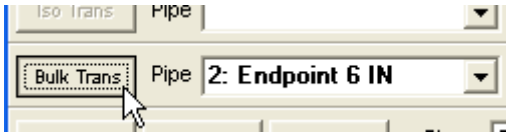
8. You should now see the following data in the Control Panel window.



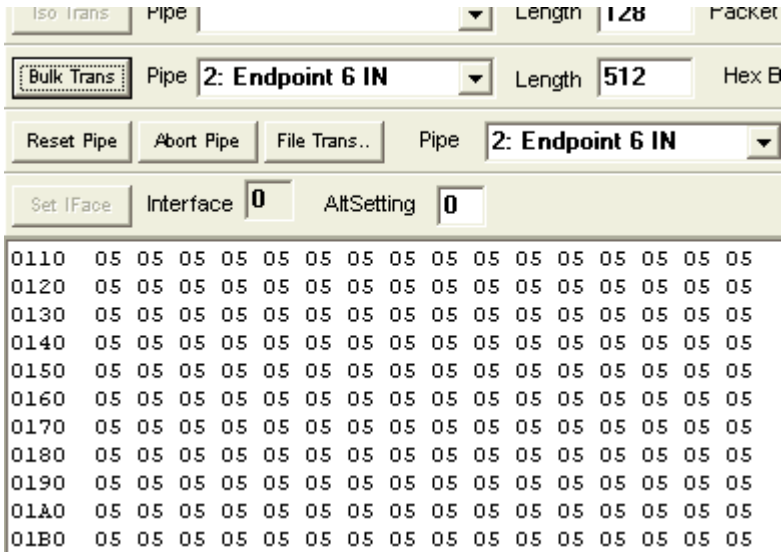
9. Read the data back into the IN pipe. Do this by selecting the "In" endpoint from the pull down menu on the "Bulk Trans" bar.



10. Pressing the "Bulk Trans" button will initiate the bulk transfer in.



11. Note that the same data came back in, as was sent out.



NOTE: If you modify the data in the "Hex Bytes" field to send out different data, that data will be sent out and be available to read back in.