Introduction to EZ-USB® FX3™ High-Speed USB Host Controller

Author: Hingkwan Huen

Associated Project: No

Associated Part Family: EZ-USB® FX3™

Software Version: FX3 Software Development Kit (SDK)

Related Application Notes: AN75705 – Getting Started with FX3

AN77960 describes how you can use the high-speed Universal Serial Bus (USB) embedded host controller in the EZ-USB® FX3™ to connect to a variety of USB peripherals. Included is a hands-on USB host example to help you create USB host applications with FX3

Contents

Introduction

USB is so commonplace that it has almost completely replaced other communication methods between peripheral devices and a PC. This holds true both for general-purpose devices, such as flash drives and mice, and for special-purpose devices for specific applications. According to the standard USB 2.0 specification, USB peripherals do not communicate directly with one another; they may communicate only with a USB host, which fully controls data traffic on the bus. The Cypress EZ-USB FX3 with an integrated high-speed USB host controller, along with the USB function and On-The-Go (OTG) capabilities, accomplishes two things: It retains the device functions and allows embedded systems to act as a USB host.

This document has two goals. The first is to introduce the high-speed USB host controller integrated in the FX3 device. The second is to describe how to create applications based on the USB host example given in the FX3 Software Development Kit (SDK).

USB Overview

This section briefly defines the terms used in this document as well as those required to understand the system operation. In addition to the. USB 2.0 Specification, there are many excellent references about USB. We suggest reading "USB in a Nutshell", available at http://www.beyondlogic.org/usbnutshell.

USB Hosts and Peripheral Devices in General

USB is a tiered star network that consists of one host and one or more peripheral devices. The host initiates all communication on the bus. Peripherals may send data to the host only when the host requests it. Peripherals must be able to receive the data that the host sends. To expand the number of peripherals, you can use a hub. Typically, a hub allows four or seven peripherals to attach to an upstream port. A maximum of five hubs can be chained together, creating as many as five tiers. A maximum of 127 peripherals (including the hubs) can be connected on the bus to a single host.

Most USB peripheral devices are categorized by classes. Each class has special requirements for its communication protocol. The host must be able to recognize a peripheral device's class and meet the class requirements, or the host cannot communicate with the device. Two common classes are human interface device (HID), such as a mouse, and mass storage class (MSC), such as a flash drive. Class drivers running on the host give application-level support for class-specific communication. Because some USB peripheral devices are vendor-specific, they do not fall into one of the predefined classes. For those peripheral devices, specific client drivers must be written.

Host Versus Embedded Host

A USB embedded host differs from a USB host in several small, but important, ways. A USB embedded host does the following:

- supports only specific peripheral devices or classes of peripheral devices, or both;
- supports only transfer types required by the supported peripheral devices;
- offers optional hub support; and
- has a relaxed power requirement.

These restrictions allow an embedded host to be implemented on a device with fixed, limited memory.

USB embedded host specification can be found in the following link:

www.usb.org/developers/onthego/USB_OTG_and_EH_2-0.pdf

USB On-The-Go (OTG)

USB On-The-Go (OTG) allows two USB peripherals to talk to each other without requiring a personal computer. Although OTG appears to add peer-to-peer connections, in fact, USB OTG retains the standard USB host/peripheral model. OTG introduces the dual-role device (DRD) capable of functioning as either host or peripheral. Because OTG device can be a DRD, OTG specification defines new terms for host and peripheral to avoid confusions: An A-Device is the default host at the start of a session; A B-Device is the default peripheral at the start of a session. One important aspect of OTG is that a A-Device and B-Device can switch roles when necessary.

USB OTG includes three protocols--Session Request Protocol (SRP), Attach Detection Protocol (ADP), and Host Negotiation Protocol (HNP).

SRP allows the USB link to remain unpowered until a device requests power. Controlling the power activity is important for devices such as a mobile phone that use a battery, because it prolongs battery life.

ADP allows a device to check and display attachment status. When a device is detected, an A-device will look for connection and give power to the USB bus. A B-device will use SRP to request power.

HNP allows switching of host/peripheral roles between two OTG dual-role devices. This lets a USB OTG device control data transfer scheduling, so any OTG device is able to initiate data transfer.

USB OTG 2.0 specification can be found in the following link:

http://www.usb.org/developers/onthego/USB_OTG_and_EH_2-0.pdf

FX3 USB Subsystem Overview

The FX3 USB subsystem features the following controllers to support many of the latest USB advanced features:

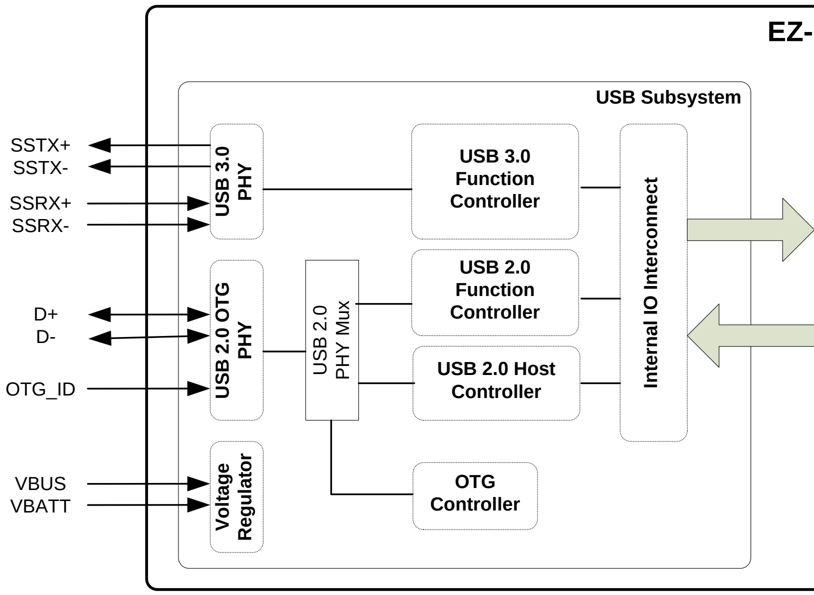
- USB 3.0 function controller
- USB 2.0 function controller
- USB 2.0 embedded host controller
- USB 2.0 OTG controller

The USB 3.0 and USB 2.0 subsystems have dedicated transceivers, but they share the same I/O interconnect in the backend.

The FX3 USB subsystem supports the following modes of operation:

- USB 3.0 peripheral in SuperSpeed (5 Gbps)
- USB 2.0 peripheral in high/full speed (480/12 Mbps, respectively)
- USB 2.0 host in high/full/low speed (480/12/1.5 Mbps respectively), with one downstream port. The hub is not supported.
- USB 2.0 OTG DRD, with Host Negotiation Protocol (HNP) and Session Request Protocol (SRP) support.

Figure 1. EZ-USB FX3 Subsystem



FX3 High-Speed USB Host Controller

The FX3 high-speed USB host controller has the following features:

- Complies with the USB 2.0 standard for high-speed (480 Mbps), full-speed (12 Mbps) and low-speed (1.5 Mbps) functions
- Supports all transfer types: control, bulk, isochronous, and interrupt
- Supports SRP and HNP.
- Supports suspend/resume and remote wakeup
- Supports high-bandwidth isochronous and interrupt transfers
- 15 IN endpoints and 15 OUT endpoints in addition to control endpoint 0
- Flexible endpoint configurations with the following properties:
  - All endpoints are mapped to system memory
  - All available memory can be allocated to endpoints
  - Can be dynamically sized by firmware
- Performs all transaction scheduling in hardware
- USB hub not supported

FX3 USB Host API from SDK

In most cases, developers are required to know about the host internal, as well as USB's protocol, to program a USB host. The FX3 SDK includes a set of plug-and-play building blocks, including the USB host API module, to reduce the complexity and simplify the system development of the FX3 USB host. Available API functions are highlighted below.

- Start and stop the USB host stack
- Enable/disable the USB host port
- Reset/suspend/resume the USB host port
- Get/set the device address
- Add/remove/reset an endpoint
- Schedule and perform EP0 transfers
- Set up/abort data transfers on endpoints

Details of each function call and its usage are described and documented in the following C header file.

${FX3_INSTALL_PATH}\firmware\u3p_firmware\inc\cyu3usbhost.h

Running the FX3 USB Host Example

The FX3 SDK includes a set of examples that demonstrate the usage case of the FX3 high-speed USB host. These examples include:

cyfxusbhost – FX3 as a USB host

This example demonstrates the use of FX3 as a USB 2.0 single-port host. It supports simple HID-class and simple MSC-class devices.

cyfxusbotg – FX3 as an OTG device

This example demonstrates the use of FX3 as an OTG device. When FX3 is connected to a PC USB host, it acts as a bulk loopback peripheral device. When it is connected to a USB mouse, it can detect and track the mouse clicks, X-Y coordinates, and scroll changes.

cyfxbulklpotg – FX3 connected to FX3 as an OTG device

This example demonstrates the full OTG capability of the FX3. When FX3 is connected to a PC USB host, it acts as a bulk loopback peripheral device. When it is connected to another FX3 running the same firmware, it demonstrates SRP and HNP.

Although a step-by-step procedure is given only to the **cyfxusbhost** example in this section of the document, the other two examples follow the same idea.

Figure 2. cyfxusbhost Example Setup

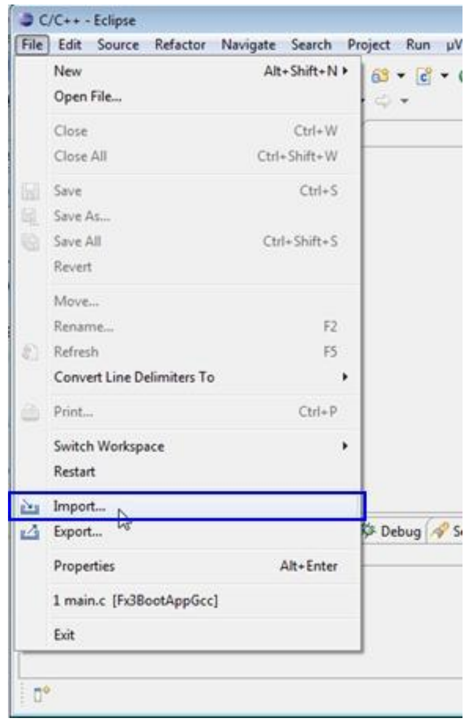
To successfully run the **cyfxusbhost** example, you need the following software tools and hardware components:

- FX3 SDK
  - Eclipse IDE with Zylin Embedded CDT plug-in
  - Example source and API libraries
  - GNU ARM compiler tool chain
- JTAG debugger. Any JTAG debuggers supporting ARM GDB hardware debugging will work. Following are the two models that have been tested by Cypress.
  - Segger J-Link with J-Link GDB server
  - Olimex ARM-USB-OCD-H with OpenOCD
- UART terminal and application (e.g., Teraterm)
- FX3 DVK with 5-V power supply
- Standard USB mouse and USB flash drive
- USB micro-B to standard-A adapter

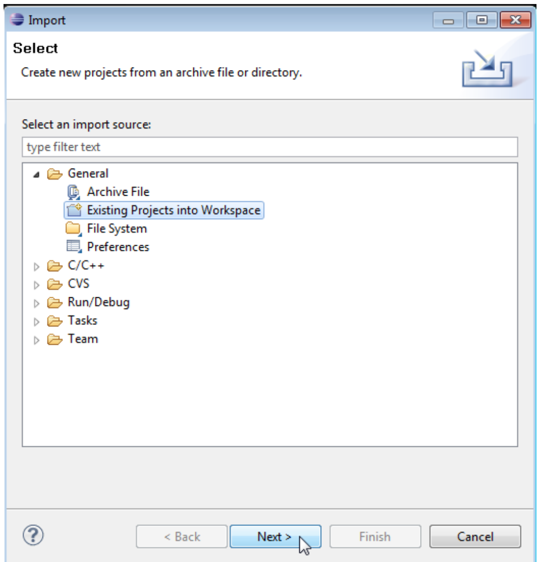
For general instructions of using the FX3 development tools, refer to "FX3 Development Tools" section from FX3 Programmers Manual.

Step 1: Import the Project

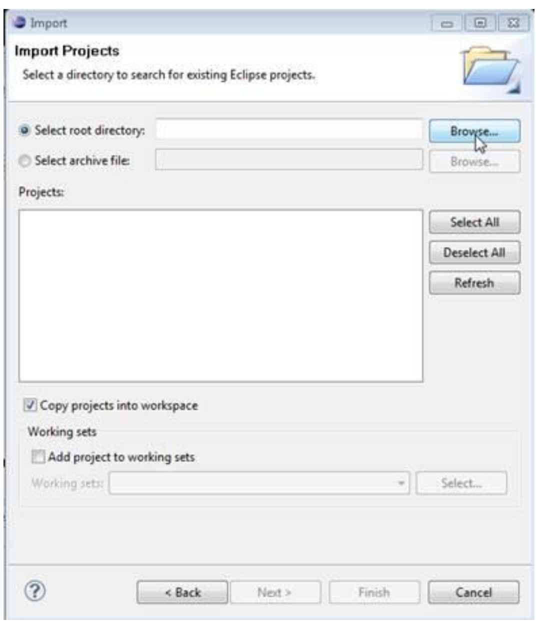
1. To start the Eclipse CDT IDE, double-click the  icon.
2. Within the Eclipse IDE, select **File > Import**.



3. An **Import** window pops up. Select **Existing Projects into Workspace**, then click **Next >**.

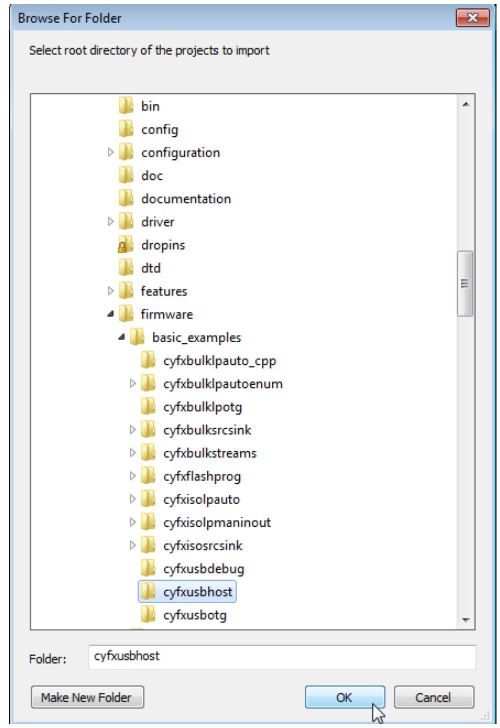


4. Click the radio button for **Select root directory**, then click **Browse.**

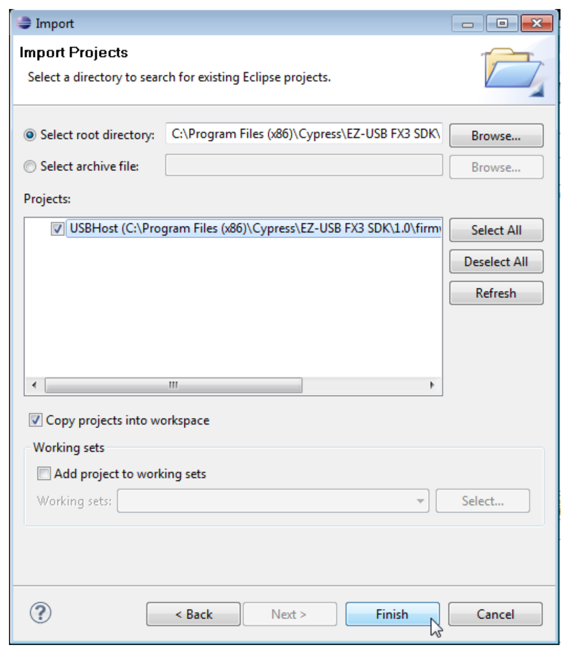


5. Browse to the example project directory and select the **cyfxusbhost** example, then click **OK**. The example project directory is usually located at

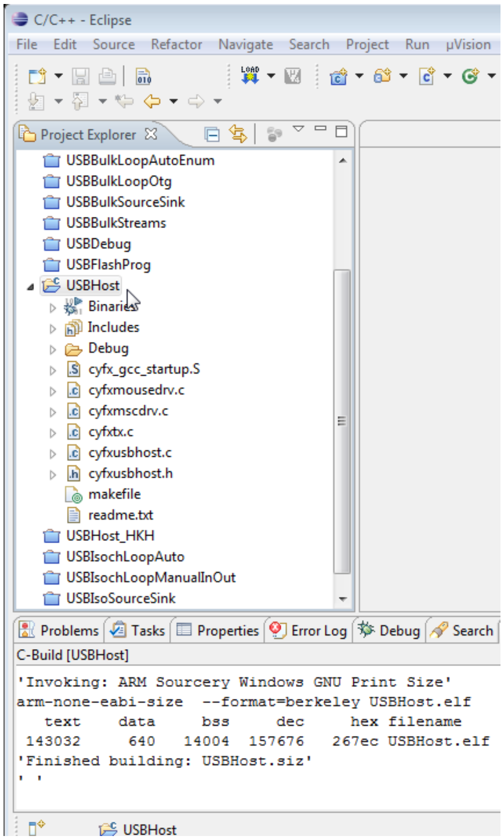
   "${FX3_INSTALL_PATH}\firmware\basic_examples".



6. The **USBHost** project shows up in the **Projects** list. Select the project and **Copy projects into workspace** and then click **Finish**.



7. After the USBHost project is imported into your workspace, it appears in the **Project Explorer** pane.



Step 2: Compile the Project

Eclipse will automatically build the project if “Build Automatically” is enabled. In the ‘USBHost’ example, VBUS control logic is the reverse of DVK. Below are instructions on how to change it for the DVK and how to recompile the project.

From the **Project Explorer** pane, expand the **USBHost** project, and then double-click "*cyfxusbhost.h*" to open the file. Within the file, change lines 44 and 45 from

```
#define CY_FX_HOST_VBUS_ENABLE_VALUE
(CyFalse)
#define CY_FX_HOST_VBUS_DISABLE_VALUE    (CyTrue)
```

to

```
#define CY_FX_HOST_VBUS_ENABLE_VALUE     (CyTrue)
#define CY_FX_HOST_VBUS_DISABLE_VALUE
(CyFalse)
```

After you make the changes, it is ready to re-compile.

If “Build Automatically” is enabled, then select

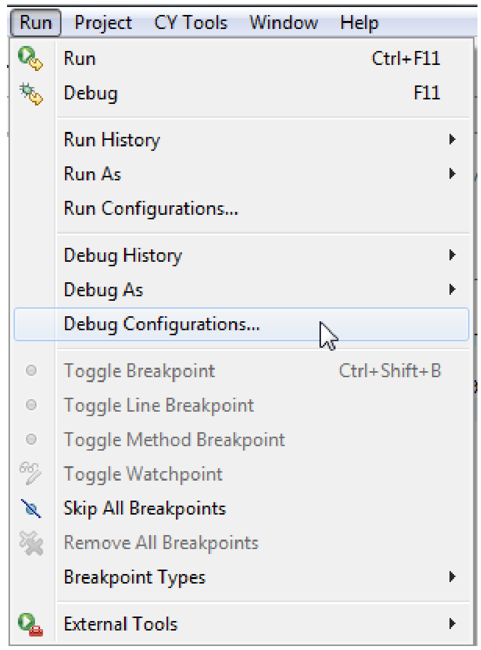
Project > Clean

That step cleans and rebuilds the project automatically.

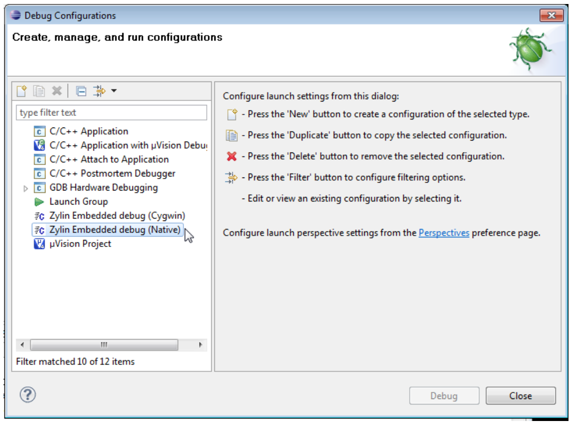
If “Build Automatically” is disabled, then simply select **Project >Build Project** to invoke the build process.

Step 3: Configure the Zylin Embedded CDT Plug-in for GDB Debug

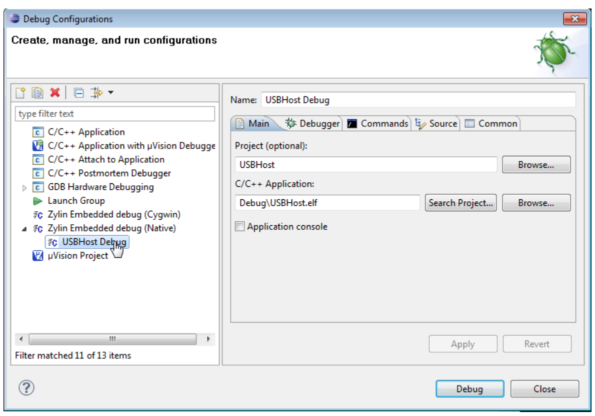
1. Select USBHost as the current project, and then select **Run > Debug** Configurations.



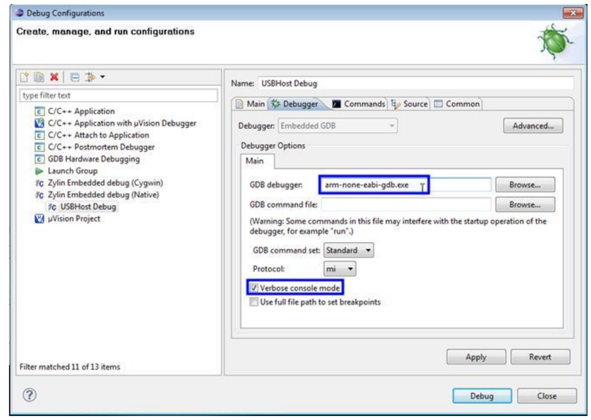
2. A Debug Configurations window pops up. Double-click on **Zylin Embedded debug (Native)**.



3. If the USBHost project is selected as current project, an entry with USBHost Debug appends below Zylin Embedded debug (Native). On the left pane under the Main tab, the Project and C/C++ Application fields automatically populate. If not, click on Browse and select them manually.



4. On the left pane, select the Debugger tab. Type "arm-none-eabi-gdb.exe" in the GDB debugger field. Select Verbose console mode.



5. Select Commands tab. Below the Initialize commands, type the following and replace <Port_Num> with 2331 for J-Link, or 3333 for Olimex.

   target remote localhost:<Port_Num>

   monitor halt

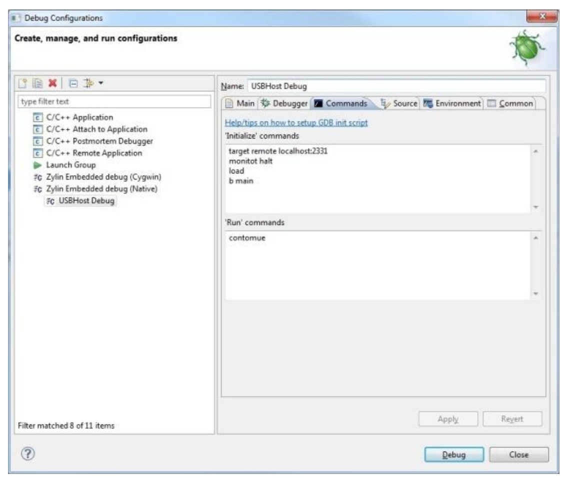
   b main

   load

   Below the Run commands, type the following:

   continue

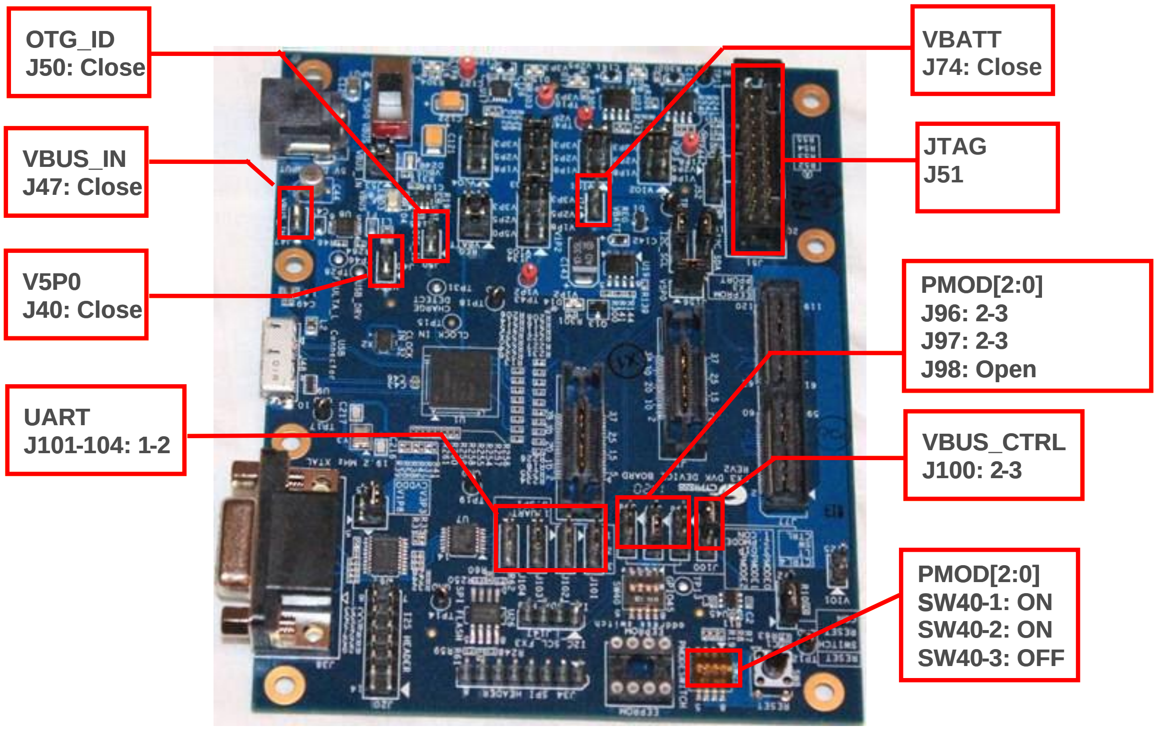
   Then select **Apply**.



Step 4: Configure the DVK to Run in USB Host Mode

Figure 3 shows the required jumper and switch settings.

Figure 3. Required Jumper and Switch Settings



Step 5: Connect the JTAG debugger and start the GDB Hardware Debugging software

Connect the JTAG debugger to the 20-pin ARM JTAG connector (J51) on the FX3 DVK and power up the FX3 DVK board.

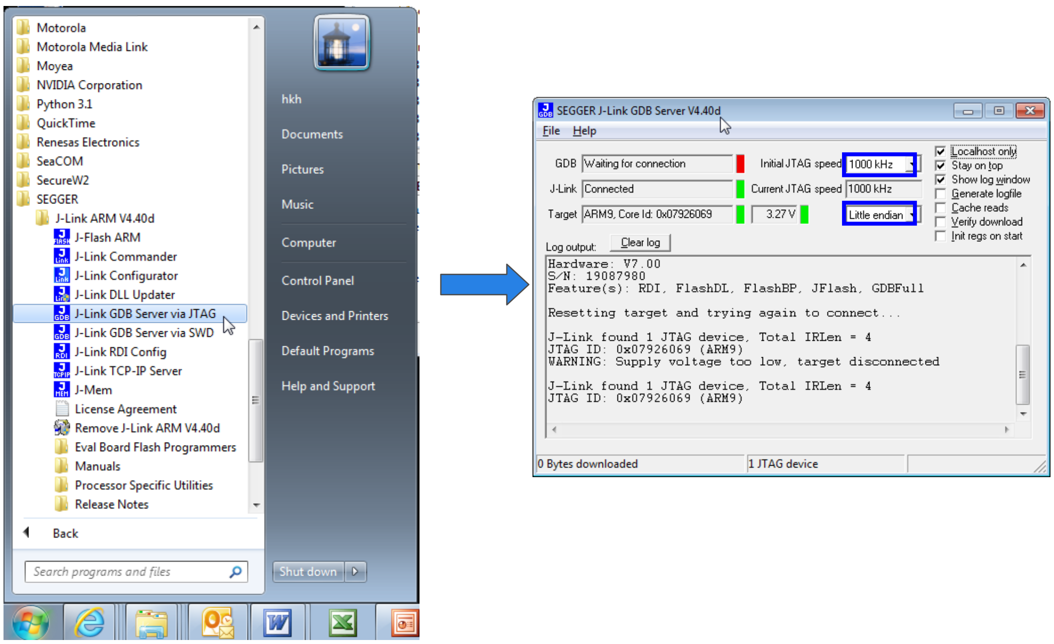
This application note assumes that the software and driver for the JTAG debugger have been installed properly on the PC.

For Segger J-Link, you can obtain the software package from Segger's website: http://www.segger.com/jlink-software.html. Extract and install the package according to the J-Link User Guide available from the same link.

Start the Segger J-Link GDB Server by

Start >All Programs >SEGGER >J-Link ARM Vx.xxx >J-Link GDB Server via JTAG

The SEGGER J-Link GDB Server Vx.xxx window pops up. and then the "ARM9 Core Id: 0x07926069" shows up in the Target field. Change the initial JTAG speed to 1000 kHz and select "Little endian".



For Olimex ARM-USB-OCD-H, the OLIMEX ARM DEVELOPMENT PACKAGE V1.1 is needed from Olimex's website: https://www.olimex.com/Products/ARM/JTAG/ARM-USB-OCD-H/. Simply extract this package under a working directory. The example in this application note places the package under C:\Cypress\OpenOCD_Olimex\.

OpenOCD can be executed from Eclipse IDE directly as an external tool. Follow these steps:

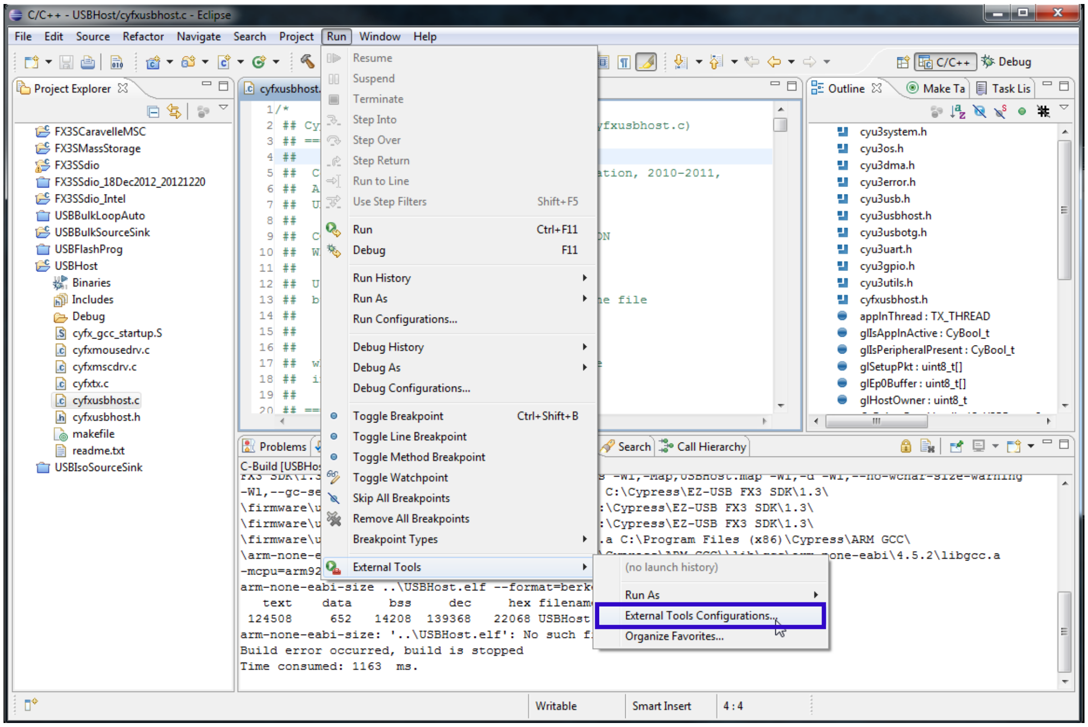
1. Copy and paste the following text into a text file named "arm926ejs.cfg" and save the file in the same directory as the OpenOCD executable openocd-libftdi.exe, which is under C:\Cypress\OpenOCD_Olimex\OpenOCD\.

```
### Start of arm926ejs.cfg
# Olimex ARM-USB-OCD-H
# http://www.olimex.com/dev/arm-usb-ocd.html

#interface
interface ft2232
ft2232_device_desc "Olimex OpenOCD JTAG ARM-USB-OCD-H"
ft2232_layout olimex-jtag
```

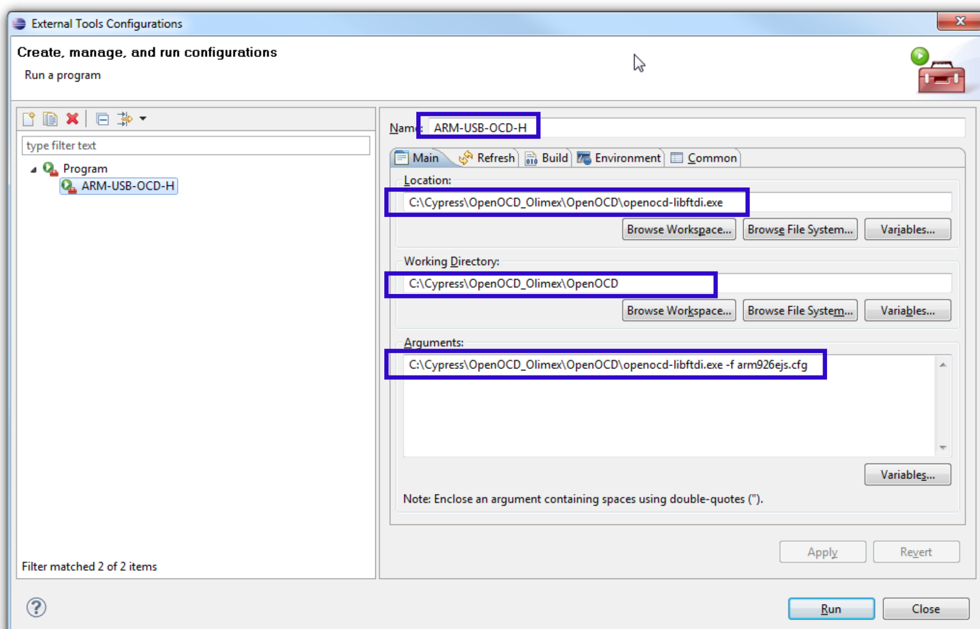
```
ft2232_vid_pid 0x15ba 0x002b

######################################
# Target:    CY FX3 ARM926ejs
######################################
if { [info exists CHIPNAME] } {
           set _CHIPNAME $CHIPNAME
} else {
           set  _CHIPNAME fx3
}

if { [info exists ENDIAN] } {
           set _ENDIAN $ENDIAN
} else {
           set _ENDIAN little
}

if { [info exists CPUTAPID] } {
           set _CPUTAPID $CPUTAPID
} else {
           set _CPUTAPID 0x07926069
}

#delays on reset lines
jtag_nsrst_delay 200
jtag_ntrst_delay 200
jtag_khz 1000
reset_config trst_and_srst srst_pulls_trst
jtag newtap $_CHIPNAME cpu -irlen 4 -ircapture 0x1 -irmask 0xf -expected-id $_CPUTAPID

######################
# Target configuration
######################
set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME arm926ejs -endian $_ENDIAN -chain-position $_TARGETNAME -variant arm926ejs

### End of arm926ejs.cfg
```

2. From the Eclipse IDE, select

   Run >External Tools >External Tools Configurations…



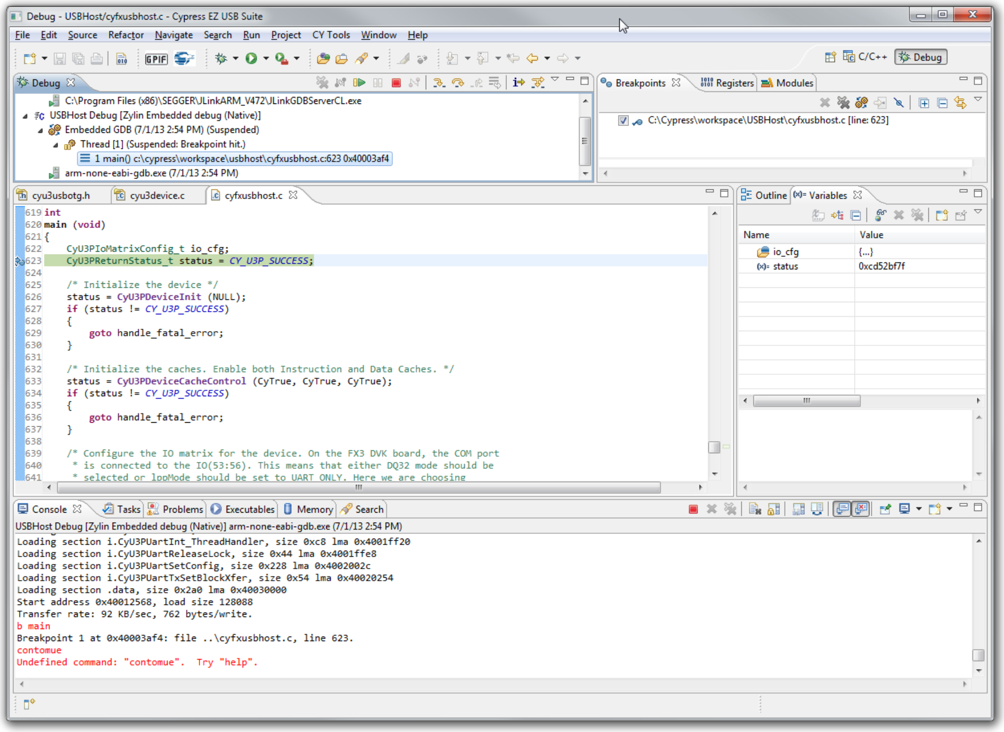
3. From the External Tools box with Configurations window, double-click Program from the left pane to add a new configuration entry. Fill in the configurations on the right pane as follows and then select **Run**.



Step 6: Download and Execute the USBHost Project on FX3 DVK

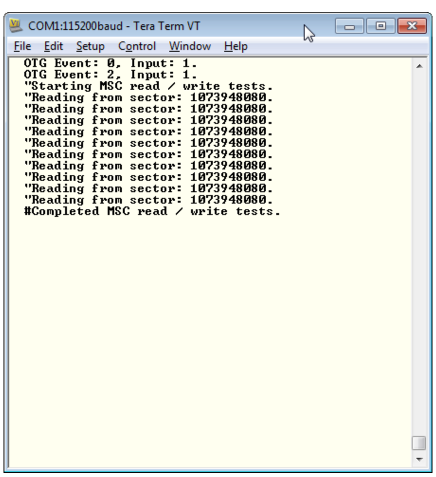
In the Eclipse Debug Configurations window, click **Debug**. Remember to use the correct port number for the “target remote localhost:<port number>” command. The port number for J-Link is 2331, and for Olimex it’s 3333.

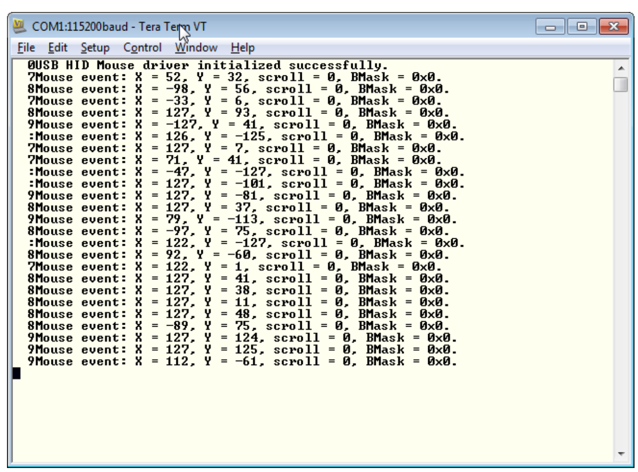
1. After the Debug session starts, the Eclipse changes to Debug perspective, and firmware execution stops at the breakpoint in main() function.



2. On top of the Debug pane, select the Resume button. If the FX3 USB host controller is running, a line that says “OTG Event: 2, Input: 1” shows up in the UART terminal set to 1152008N1N.

3. Plug in either the USB flash drive or mouse to the DVK’s USB port using the Micro-B-to-A adapter. The respective output shows in the UART terminal.



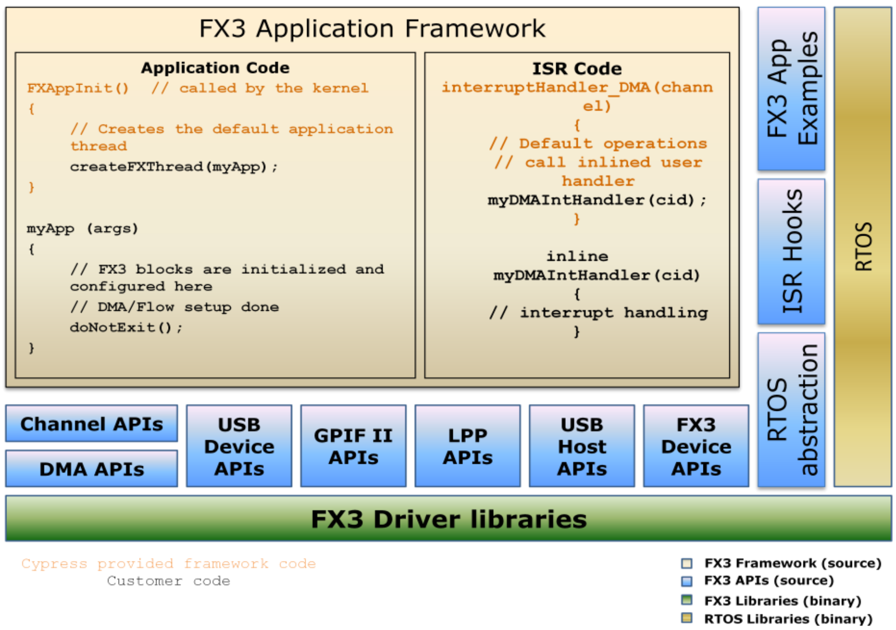


Using the FX3 USB Host with Host Mode API

The last section of this application note has demonstrated how to run the USBHost example project on the FX3 DVK. Using the same example, this section presents details on how to work with the FX3's embedded USB host with the host mode API.

The FX3 application firmware has the same basic structure as the one illustrated in Figure 4.

Figure 4. FX3 Application Firmware Structure

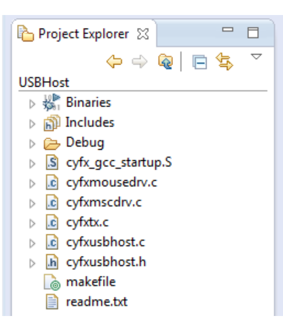


The FX3 firmware application runs on top of a real-time operating system (RTOS). The RTOS efficiently manages FX3 device internal resources.

The FX3 firmware application communicates with the FX3 hardware peripherals through a set of APIs that abstracts the details of the device physical interface and simplifies the application code.

USBHost Example Application Framework

Using the same USBHost project as an example, the firmware framework consists of the following source files:



- **cyfx_gcc_startup.S:** Start-up code for the ARM-9 core on the FX3 device. This assembly source file follows the syntax for the GNU assembler.
- **cyfxmousedrv.c:** USB HID mouse driver implementation, which works with a simple single interface USB HID mouse. The driver will enumerate the mouse when connected and report the current offset via the UART debug terminal.
- **cyfxmscdrv.c:** USB Mass-Storage device Class (MSC) driver implementation, which works with a simple single interface USB BOT MSC device. The driver will enumerate and query the storage parameters when device is connected. It performs read / write tests to fixed sectors which is repeated at an interval of 1 minute. The write operation is disabled by default. It can be enabled by changing the value of CY_FX_MSC_ENABLE_WRITE_TEST to 1 in the *cyfxusbhost.h* file.
- **cyfxusbhost.h:** Constant definitions for the host application.

- **cyfxtx.c:** ThreadX RTOS wrappers and utility functions required by the FX3 API library.
- **cyfxusbhost.c:** Main C source file that implements the host mode example.

USBHost Example Walk-Through

When the USBHost firmware execution starts, it performs an initialization sequence for the FX3 device and the compiler tool chain library, followed by the RTOS. The RTOS begins by calling `CyU3PKernelEntry()` from the `main()` function. Before the RTOS starts its thread scheduling, at least one thread is created to perform the application task. For the USBHost example project, the application thread is `ApplnThread_Entry()`, which ends up in an infinite for-loop executing the appropriate task based on the value of the global variable `glIsPeripheralPresent`.

```
    for (;;)
    {
        CyU3PThreadSleep (100);
        if (isPresent != glIsPeripheralPresent)
        {
            /* Stop previously started application. */
            if (glIsApplnActive)
            {
                CyFxApplnStop ();
            }

            /* If a peripheral got connected, then enumerate
             * and start the application. */
            if (glIsPeripheralPresent)
            {
                status = CyU3PUsbHostPortEnable ();
                if (status == CY_U3P_SUCCESS)
                {
                    CyFxApplnStart ();
                }
            }

            /* Update the state variable. */
            isPresent = glIsPeripheralPresent;
        }

        /* Since the test needs to be done from a thread,
         * this function is called at fixed interval. */
        if (glHostOwner == CY_FX_HOST_OWNER_MSC_DRIVER)
        {
            CyFxMscDriverDoWork ();
        }
    }
```

The `glIsPeripheralPresent` variable is updated whenever there is a peripheral connect or disconnect event from the FX3 USB host controller. All host events are handled within a callback function `CyFxHostEventCb()` registered during the host controller initialization within `CyFxUsbHostStart()`.

```
void
CyFxHostEventCb (CyU3PUsbHostEventType_t evType, uint32_t evData)
{
    /* This is connect / disconnect event. Log it so that the
     * application thread can handle it. */
    if (evType == CY_U3P_USB_HOST_EVENT_CONNECT)
    {
        glIsPeripheralPresent = CyTrue;
    }
    else
    {
        glIsPeripheralPresent = CyFalse;
    }
}
```

Host events are predefined in the API. The current API version supports only two host events: connect and disconnect.

```
typedef enum CyU3PUsbHostEventType_t
{
   CY_U3P_USB_HOST_EVENT_CONNECT = 0,   /* USB Connect event. */
   CY_U3P_USB_HOST_EVENT_DISCONNECT   /* USB Disconnect event. */
} CyU3PUsbHostEventType_t;
```

Connect Event

After the USBHost firmware detects that a USB mouse or flash drive is connected to the host, the `CY_U3P_USB_HOST_EVENT_CONNECT` event triggers the `CyFxHostEventCb()` callback, which updates the `glIsPeripheralPresent` to `CyTrue`. The application thread `ApplnThread_Entry()` then enables the host by calling the `CyU3PUsbHostPortEnable()`, follow by starting the application task in `CyFxApplnStart()`.

The device enumeration occurs inside the `CyFxApplnStart()`. Enumeration begins with initializing the endpoint data structure `epCfg` for EP0 control endpoint, and then adding the EP0 to the host schedule with `CyU3PUsbHostEpAdd()`.

```
    CyU3PMemSet ((uint8_t *)&epCfg, 0, sizeof(epCfg));
    epCfg.type = CY_U3P_USB_EP_CONTROL;
```

```
    epCfg.mult = 1;
    epCfg.maxPktSize = 8;
    epCfg.pollingRate = 0;
    epCfg.fullPktSize = 8;
    epCfg.isStreamMode = CyFalse;
    status = CyU3PUsbHostEpAdd (0, &epCfg);
```

Once the EP0 is in the host schedule, firmware starts to send standard USB requests to the attached USB mouse or flash drive using `CyFxSendSetupRqt()`. The enumeration process consists of a series of standard USB requests; the host obtains the device information and configuration from the USB descriptors. From the device descriptor firmware, the host determines the device type (only mouse and flash drive are supported by USBHost example), and then calls the appropriate driver initialization function: `CyFxMouseDriverInit()` for mouse or `CyFxMscDriverInit()` for flash drive.

In either of the two driver initialization functions, firmware continues the enumeration process by reading the full length of configuration descriptor from the device. Then, the firmware sets the supported configuration. Before the firmware can communicate with the device, the drivers also do the following:

- initialize the endpoint data structures for the matching endpoints of the device
- add the matching endpoints to host schedule
- initialize and create DMA channels for each endpoint

Once the initialization is done, the HID driver sets up an infinite transfer to the interrupt IN endpoint that constantly sends out IN token to request update of mouse data.

For the MSC driver, it continues the MSC initialization with `CyFxMscTestUnitReady()` and `CyFxMscReadCapacity()`, and then exits the `CyFxMscDriverInit()` function. After the MSC driver fully initializes, the application main thread `ApplnThread_Entry()` starts the MSC task by calling the `CyFxMscDriverDoWork()` periodically, which it generates reads (and writes if enabled) to the flash disk.

Note that besides control endpoint, all endpoint transfers from host are initiated by `CyU3PUsbHostEpSetXfer()` if the endpoint is in the host schedule. The function submits the transfer request to the host scheduler. The host scheduler determines when to execute the request according to the policy defined in the OHCI and EHCI specifications. The following two functions from the MSC driver, `CyFxMscSendBuffer()` and `CyFxMscRecvBuffer()`, demonstrate simple ways to do OUT and IN bulk transfers respectively. Other endpoint types besides the control endpoint can use the same sequence to submit transfer request to the host scheduler.

```
CyU3PReturnStatus_t
CyFxMscSendBuffer (
        uint8_t *buffer,
        uint16_t count)
{
    CyU3PDmaBuffer_t buf_p;
    CyU3PUsbHostEpStatus_t epStatus;
    CyU3PReturnStatus_t status =
CY_U3P_SUCCESS;

    /* Setup the DMA for transfer. */
    buf_p.buffer = buffer;
    buf_p.count  = count;
    buf_p.size   = ((count + 0x0F) &
~0x0F);
    buf_p.status = 0;
    status = CyU3PDmaChannelSetupSendBuffer
(&glMscOutCh, &buf_p);
    if (status == CY_U3P_SUCCESS)
    {
        status = CyU3PUsbHostEpSetXfer
(glMscOutEp,

CY_U3P_USB_HOST_EPXFER_NORMAL, count);
    }
    if (status == CY_U3P_SUCCESS)
    {
        status =
CyU3PUsbHostEpWaitForCompletion
(glMscOutEp, &epStatus,
                CY_FX_MSC_WAIT_TIMEOUT);
    }
    if (status == CY_U3P_SUCCESS)
    {
        status =
CyU3PDmaChannelWaitForCompletion
(&glMscOutCh, CYU3P_NO_WAIT);
    }

    if (status != CY_U3P_SUCCESS)
    {
        CyFxMscErrorRecovery ();
    }

    return status;
}

CyU3PReturnStatus_t
CyFxMscRecvBuffer (
        uint8_t *buffer,
        uint16_t count)
{
    CyU3PDmaBuffer_t buf_p;
    CyU3PUsbHostEpStatus_t epStatus;
    CyU3PReturnStatus_t status =
CY_U3P_SUCCESS;

    /* Setup the DMA for transfer. */
    buf_p.buffer = buffer;
    buf_p.count  = 0;
    buf_p.size   = ((count + 0x0F) &
~0x0F);
    buf_p.status = 0;
```

```
    status = CyU3PDmaChannelSetupRecvBuffer
(&glMscInCh, &buf_p);
    if (status == CY_U3P_SUCCESS)
    {
        status = CyU3PUsbHostEpSetXfer
(glMscInEp,

CY_U3P_USB_HOST_EPXFER_NORMAL, count);
    }
    if (status == CY_U3P_SUCCESS)
    {
        status =
CyU3PUsbHostEpWaitForCompletion (glMscInEp,
&epStatus,
                CY_FX_MSC_WAIT_TIMEOUT);
    }
    if (status == CY_U3P_SUCCESS)
    {
        status =
CyU3PDmaChannelWaitForCompletion
(&glMscInCh, CYU3P_NO_WAIT);
    }

    if (status != CY_U3P_SUCCESS)
    {
        CyFxMscErrorRecovery ();
    }

    return status;
}
```

Disconnect Event

When the USB mouse or flash drive is disconnected from the host, it follows the same logic as the connect event. The `CY_U3P_USB_HOST_EVENT_DISCONNECT` event triggers the `CyFxHostEventCb()` callback, which updates the `glIsPeripheralPresent` to `CyFalse`. The application thread `ApplnThread_Entry()` then calls `CyFxApplnStop()` to stop the application task. Within `CyFxApplnStop()` firmware disables the active driver with `CyFxMouseDriverDeInit()` or `CyFxMscDriverDeInit()`, which removes all active endpoints from the host schedule and the associated DMA channels. Before returning to the application main thread, `CyFxApplnStop()` removes the control endpoint from the host schedule and then disables the host port. Once `CyFxApplnStop()` exits, firmware returns to the same state as it initially comes up and waits for connect event.

Other Useful Host API Functions

The USBHost example shown in this application note did not use every FX3 USB host API function available. Below is a list of some of the commonly used ones while working with the FX3 USB host. What each function does is self-explanatory. For a full list of these API functions and detailed usage of each, refer to the FX3 API Guide from the SDK document.

```
CyU3PUsbHostStart()
CyU3PUsbHostStop()

CyU3PUsbHostGetPortStatus()
CyU3PUsbHostPortEnable()
CyU3PUsbHostPortDisable()
CyU3PUsbHostPortReset()
CyU3PUsbHostPortSuspend()
CyU3PUsbHostPortResume()

CyU3PUsbHostEpAdd()
CyU3PUsbHostEpRemove()
CyU3PUsbHostEpReset()
```

Summary

This introduction of the EZ-USB FX3 high-speed USB host controller showed you a simple way to bring USB host capability to embedded applications. Included in the document were associated library and firmware examples in the SDK. As a result, this application note gave you everything you need to start an implementation with ease.

About the Author

Name:	Hingkwan Huen
Title:	Systems Engineer Senior Staff
Contact:	hkh@cypress.com

Document History

Document Title: Introduction to EZ-USB® FX3™ High-Speed USB Host Controller – AN77960

Document Number: 001-77960

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3606545	HKH	05/02/2012	New Spec.
*A	3786154	HKH	10/29/2012	Added external link for USB primer Updated to support SDK v1.2 Updated the running procedure for example project from SDK v1.2 Added details how to work with the FX3 embedded host with API
*B	3822643	CFT	11/27/2012	Minor ECN to match document title with the title in the spec system
*C	4049586	HKH	07/03/2013	Removed the SDK v1.2 reference in the title summary Added settings for Olimex ARM-USB-OCD-H in the instruction Minor edits and cleanup for clarity and grammar.

Worldwide Sales and Design Support

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

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com