Cypress EZ-USB® FX3™ SDK

Troubleshooting Guide

Version 1.3.4

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

Contents

1 Introduction 4

2 USB Troubleshooting 5
2.1 USB Enumeration Failures 5
2.2 Unexpected Connection Failures 6
2.3 USB Data Transfer Failures 6
2.4 Low USB transfer performance 8
2.5 USB Configuration Errors 9
2.6 Isochronous transfer failures at hi-speed 9
2.7 USB 2.1 LPM-L1 Interoperability Failures 10

3 Data Streaming Troubleshooting 11
3.1 Flow Control Issues 11

4 GPIF II Troubleshooting 13
4.1 Data Transfer Failures 13

5 SDK Tools Troubleshooting 14
5.1 Elf2img converter issues 14
5.2 Eclipse IDE issues 15

1 Introduction

This document provides hints on troubleshooting system errors that are seen when using the EZ-USB FX3 device and the FX3 SDK. It also serves as a FAQ database for the FX3 SDK.

The hints provided here are based on the learning from multiple FX3 based system designs, and will be updated on a periodic basis. This document should be used in conjunction with the FX3 Programmer's Manual and API Guide documents that are part of the SDK package itself.

2 USB Troubleshooting

2.1 USB Enumeration Failures

I. FX3 does not enumerate as a USB 3.0 device after firmware is downloaded.

In most cases, USB 3.0 enumeration failure is due to poor signal integrity on the USB 3.0 lines of the FX3 device. Please ensure that you are following the FX3 board design guidelines from Cypress. Refer to AN70707: EZ-USB® FX3™/FX3S™ Hardware Design Guidelines and Schematic Checklist for details.

The USB driver in the SDK is capable of logging USB related events and state changes into a user provided memory buffer. This buffer content can later be read back through USB or UART to identify the event sequence that led to the failed enumeration.

Refer to the CyU3PUsbInitEventLog() API in the FX3 API guide for information on how to initiate these driver logs. The USBBulkSourceSink example in the SDK includes reference code that makes use of this API and provides two ways of reading the event logs out:

- Using a vendor specific USB control request.
- Through UART debug prints

The contents of the event log buffer are single bytes that are to be interpreted using a set of constants defined in the cyu3usb.h header file. Refer to the macro definitions for CYU3P_USB_LOG_VBUS_OFF onwards.

II. FX3 fails to enumerate as a Hi-Speed device when SuperSpeed is enabled

This can happen if the FX3 device detects SuperSpeed receiver terminations on the host side, and then fails to receive the Polling LFPS signaling from the host; it will enter the USB 3.0 compliance state. The device can only exit from the compliance state if a USB 3.0 Reset is received or if Vbus is turned off. New USB events such as `CY_U3P_USB_EVENT_SS_COMP_ENTRY` and `CY_U3P_USB_EVENT_SS_COMP_EXIT`, are added which can be used to identify whether the device is entering to exiting the compliance mode.

Please verify that the USB host port does not offer receiver terminations without the USB 3.0 host being available.

III. FX3 falls to a Hi-Speed connection after the host resumes from a sleep/hibernate state

The USB driver for FX3 requires that the device be able to detect Vbus voltage changes. If a Vbus change is not detected by the device when it is disconnected from the host; it will only connect as a hi-speed device on a subsequent USB connection.

2.2 Unexpected Connection Failures

I. The SuperSpeed USB connection from FX3 fails abruptly during data transfer, or the device re-enumerates.

- The USB driver in the FX3 SDK monitors the number of USB link errors that are detected by the hardware; and causes a re-enumeration when the number of errors crosses a threshold value (about 64 errors) within a 1 second period. This code is not expected to come into play on a functional link, because there will be not more than one or two errors per second happening. If the device is re-enumerating, it is likely that there is a bad USB link due to bad interconnect cables or traces.
- Another reason for USB 3.0 connection errors is link errors that happen around link power state transitions. The best option to avoid such problems is to prevent USB 3.0 link state transitions, by having the FX3 device systematically reject any low power requests.
- It is recommended to disable the low power mode transitions by calling CyU3PUsbLPMDisable() API when the data transfers are active because there may be cases where the host performs very fast U1 Entry/Exit (Entry to Exit duration < 5 µs) and the firmware is not capable of responding to such fast low power mode transitions.

  The FX3 device can be placed in this mode by making use of the ***CyU3PUsbLPMDisable()*** API. Please note that using this API can also help improve the USB data transfer performance because of increased link efficiency.

2.3 USB Data Transfer Failures

I. FX3 stops sending data on all USB IN endpoints abruptly.

- It is possible that the endpoint memory block on the FX3 device locks up if the DMA channel for that endpoint is reset or aborted while it is ready from the DMA buffers. The same applies to endpoint reset and flush (***CyU3PUsbResetEp and CyU3PUsbFlushEp***) API calls.

  The caller should ensure that the endpoint is idle before performing any reset or flush operations on the DMA channel or the endpoint. This can be done by placing the endpoint in NAK/NRDY mode where it does not try to send data to the host. This sequence is shown in the code snippet below.

```
CyU3PUsbSetEpNak (epNum, CyTrue);       // NAK the endpoint.
CyU3PBusyWait (100);                    // Insert a delay
// Do clean-up on the channel and EP now.
CyU3PDmaChannelReset (&epChannel);
CyU3PUsbFlushEp (epNum);
// Finally release the EP from NAK mode.
CyU3PUsbSetEpNak (epNum, CyFalse);
```

- It has also been seen that the use of the CyU3PUsbEnablePrefetch() API is causing the endpoints to get stuck, because they try to read very aggressively from the DMA channels. Removing this API call will fix these issues.

  It was earlier recommended that this API be used to prevent data corruption errors during high speed data transfers. The device settings that are required to prevent data corruption which were earlier being made as part of this API call; have now

been made the default settings for the driver. Therefore, there are no known problems associated with removing this API call.

II. Transfer freeze on a burst enabled IN endpoint

- The FX3 device has an errata that causes it to not treat a ZLP (Zero Length Packet) as an end of burst packets in some cases. This error happens intermittently and is caused by a race condition in the device design.

  The work-around for this issue is to ensure that DMA channel corresponding to the endpoint is suspended as soon as it has committed any ZLP to the endpoint. This can be achieved by making use of the CY_U3P_DMA_SCK_SUSP_EOP option to suspend the consumer socket on the DMA channel. The code snippet for doing this is shown below:

```
/* DMA callback that handles the channel suspension. */
static void
AppDmaCallback (
   CyU3PDmaChannel   *chHandle,
   CyU3PDmaCbType_t   cbType,
   CyU3PDmaCBInput_t *cbInput)
{
   /* Having the channel suspend is sufficient delay
      to prevent errors. Resume the DMA channel immediately.
    */
   CyU3PDmaChannelResume (chHandle, CyFalse, CyTrue);
}

{
   /* This code is to be inserted after the DMA channel
      is created. */
   CyU3PDmaChannelSetSuspend (chHandle,
         CY_U3P_DMA_SCK_SUSP_NONE, CY_U3P_DMA_SCK_SUSP_EOP);
}
```

III. Control Request Handling Errors

- The USB driver in the FX3 firmware passes all control requests addressed to an interface (including SET_INTERFACE/GET_INTERFACE) to the Setup callback function registered. If the application does not need to track and handle these requests in a customized manner, the setup callback can just return CyFalse to indicate that the request has not been handled. In this case, the USB driver will take the default action (SET_INTERFACE will be acked, and a value of 0x00 will be returned for GET_INTERFACE).

  If the setup callback returns CyTrue, the driver assumes that the application has handled the request and takes no further action (including trying to stall EP0). Please ensure that the setup callback returns CyTrue if and only if the control request has been handled.

  Any control requests that are deferred to a user thread need to be completely handled by the application logic. As the USB driver uses the return value from the callback to identify whether the request has been handled by the application, it is not possible to defer the request handling to a thread; and then hand it back to the driver for handling.

The Control requests cannot be stalled once the data is received from the Control endpoint pipe by calling CyU3PUsbGetEp0Data() API. There is no workaround for this issue.

IV. USB Data Corruption

- If FX3 is operating on a USB 3.0 link with poor signal quality, and there are a number of protocol level CRC errors and retries happening; the device could get into a state where it sends incorrect data on any of the IN endpoints (including the control endpoint). This happens due to a known defect in the way the FX3 device handles a pre-mature burst transfer termination by the host.

  SDK versions 1.3.3 and later introduce a firmware work-around to prevent this kind of failure. The USB driver in the libraries internally manages (suspends and then resumes) DMA transfers and performs an Endpoint Memory reset when potential error conditions are seen.

  As an endpoint memory reset will result in some loss of data on the in-flight endpoint; it is required that the firmware application perform an error recovery on the corresponding endpoint. This can be done by registering for the CYU3P_USBEP_SS_RESET_EVT event on all IN endpoints, and performing an endpoint specific recovery when the event is received. The recommended recovery procedure is to STALL the endpoint, and then stop and restart the DMA data path when the CLEAR_FEATURE request is received.

  Please refer to the GpifToUsb firmware example for an implementation of this sequence.

```
if (evtype == CYU3P_USBEP_SS_RESET_EVT)
{
    /* Stall the endpoint on which error occurred. */
    CyU3PUsbStall (epNum, CyTrue, CyFalse);
}
```

- When FX3 is operating on a Hi-Speed link, and performing concurrent IN transfers on any Bulk-IN endpoint along with the EP0-IN, it is possible that the data sent on the control endpoint gets corrupted. This happens due to the endpoint block on FX3 fetching data prematurely from the DMA channel.

  A work-around for this problem has been implemented in SDK versions 1.3.3 and later, where all BULK-IN DMA channels are suspended for a short duration while the EP0-IN transfer is being completed. This is done within the USB driver in the SDK and no action is required on the application side.

  The error exists only on a USB 2.0 link, and therefore this work-around is not applied when functioning as a SuperSpeed device. So, there will not be a performance impact due to the use of this work-around.

2.4 Low USB transfer performance

I. Poor system performance is seen with specific USB hosts like the Intel USB 3.0 host

- The Intel USB 3.0 host is more aggressive in the usage of USB link power saving as compared to other USB 3.0 hosts like Renesas or ASMedia. This can cause poor data transfer performance because the FX3 device is not capable of

automatically exiting a low power state (U1/U2) when it has data to be sent to the host.

The FX3 device requires firmware intervention to initiate a state transition from U1/U2 back to U0. This state change is initiated by making the following API call.

```
CyU3PUsbSetLinkPowerState (CyU3PUsbLPM_U0);
```

The transfer performance can be improved by using the ***CyU3PUsbLPMDisable()*** API to completely disable low power state transitions; or by periodically calling ***CyU3PUsbSetLinkPowerState()*** to ensure that the link is reverted to U0.

II. Poor data transfer on the IN data path with burst enabled endpoints

- In default mode, the FX3 device does not try to combine data from multiple DMA buffers into a single burst transfer on the USB side. This can lead to some performance drop, particularly when using small sized DMA buffers on the data path.

  The device can be configured to allow data from multiple buffers to be combined into a single burst, by calling the ***CyU3PUsbEPSetBurstMode()*** API.

2.5 USB Configuration Errors

I. Device with high bandwidth Isochronous endpoints fails to start up and function properly

- High bandwidth Isochronous endpoints are those that support transfers of more than one packet of data at USB Hi-Speed and one burst of data at USB SuperSpeed, per micro-frame. The FX3 device supports high bandwidth Isochronous transfers only on the Endpoints 3 and 7 (both IN and OUT). Please ensure that there endpoints are selected whenever high bandwidth Isochronous data transfers are used.
- Some USB hosts may have limitations with respect to the amount of bandwidth they can reserve for Isochronous data transfers. If the bandwidth required by the device exceeds these limits, the host will not select the device configuration. It is recommended that the application implement multiple bandwidth settings by varying the burst length and number of bursts per micro-frame. The host can then select the best possible bandwidth setting that it can support.

2.6 Isochronous transfer failures at hi-speed

II. Significant loss of data is seen when using high bandwidth Isochronous endpoints at USB hi-speed

- As per the USB 2.0 specification, a device should use the DATA0 PID when sending a single packet of data during a micro-frame, regardless of the MULT setting for the isochronous endpoint.

  The FX3 device has an errata item where the PID on a ISO data packet is always based on the MULT setting alone. i.e., The device uses the DATA2 PID if MULT is set to 2, and the DATA1 PID if MULT is set to 1.

  Sending data with the wrong PID causes the USB host to drop the data packet, leading to loss of data at the system level.

The work-around for this issue is to perform the following operations at the beginning of each micro-frame. The beginning of a micro-frame can be identified by registering for a CY_U3P_USB_EVENT_SOF_ITP event, or by configuring a GPIO timer to fire every 125 us.

1) Use a MANUAL DMA channel instead of an AUTO channel, with each buffer capable of holding the maximum amount of data that can be transferred in a micro-frame (2KB or 3KB based on MULT value).
2) Use the CyU3PDmaChannelGetBuffer() API to identify the amount of data that is available to send to the host.
3) Set the ISO MULT value for the endpoint based on the actual amount of data to be sent. The CyU3PsetEpConfig API can be called repeatedly for doing this.
4) Commit the data to the host by calling CyU3PDmaChannelCommitBuffer.

Refer to the USBVideoClass firmware example for a sample implementation of this scheme.

2.7 USB 2.1 LPM-L1 Interoperability Failures

All USB 3.0 devices are expected to pass the USB 2.1 LPM-L1 inter-operability tests that introduce a new low power mode with reduced resume latencies. The LPM-L1 specification allows a USB 2.1 peripheral to accept or reject the LPM-L1 entry requests that are generated by the USB host.

The FX3 device is not capable of rejecting the LPM-L1 request with a NYET handshake, and can only accept this request with an ACK handshake or ignore the request with no response.

I. Device operation error during L1 interoperability testing

- The USB host repeatedly forces the USB link to the L1 state during the interoperability testing. If the link enters the L1 state while the host is expecting any data from the FX3, the host will expect FX3 to initiate the exit from L1 state. As in the USB 3.0 case, the FX3 hardware does not do this automatically when it is ready for data transfer. The firmware application needs to trigger the resume operation by calling CyU3PUsb2Resume() API. The application can also use the CyU3PUsbLPMDisable() API to request the USB driver to automatically initiate an L1 exit soon after L1 entry.

  Please refer to the USB mass storage (FX3SmassStorage) and the CX3 UVC examples (cycx3_uvc_as0260 or other) for a sample implementation of the LPM handling state machine in firmware.

3 Data Streaming Troubleshooting

Devices in the FX3 family are most often used in high speed data streaming applications, with video streaming being the most common application. Various system level issues can cause such applications to stall or to run slowly, leading to poor user experience.

3.1 Flow Control Issues

I. There is loss of data while streaming between the data source and the USB host.

- Data is lost due to buffer switching latencies on the GPIF-II interface. A single DMA buffer used on FX3 can have a maximum size of 65535 bytes, and switching from one buffer to the next entails an average delay of 50 clock cycles, and any data received in this period will be lost. If the data source (e.g. Image sensor) is not capable of doing flow control, a DMA multi-channel should be used to receive the data. Switching from one DMA buffer to the next should be done based on a counter, and not based on the full flag.

  The CX3 firmware examples as well as the code provided in AN75779 - How to Implement an Image Sensor Interface with EZ-USB® FX3™ in a USB Video Class (UVC) Framework demonstrate how this can be done.

- Host side delays in reading data can cause the streaming to stall. The total buffering available on the FX3 device is limited, and long pauses on the host side can cause these buffers to overflow. Once the buffers are overflowing, some data will be lost. The firmware application needs to be able to handle this case and resume streaming properly.

  This can be done by checking for data commit errors or GPIF-II interface errors and restarting the data streaming when these errors are seen. The CX3 firmware examples have a sample implementation of this logic.

- The firmware application is not capable of handling the rate at which data needs to be forwarded. This typically happens when one or more of the following is true:

  1) The DMA buffers used are very small, resulting in a very high frequency of DMA channel interrupts and callbacks.
  2) The data forwarding is slowed down by other operations such as output debug messages or performing other peripheral accesses.
  3) Firmware application is not optimized during build.

  Please follow the below guidelines to ensure that data flow is not slowed down by the firmware implementation.

  - Build the firmware application using the fx3_release version of libraries, and with optimization set to “Optimize more” (-O2) or “Optimize most” (-O3).

- Avoid the use of data cache unless the application is making large changes to the data being streamed. Addition of a small UVC header does not require the use of the data cache.
- Perform the data update (header addition) and commit operations in a DMA callback instead of in an application thread. Doing commit from a thread context requires the RTOS to do multiple context switches which will slow down the overall operation.
- Avoid all other operations while the high speed data streaming is happening. In particular, avoid calling CyU3PDebugPrint from the main data commit logic.
- Make use of all the buffering space available on FX3. Make sure that at least two buffers can be associated with each DMA socket; and make each DMA buffer as large as possible.

4 GPIF II Troubleshooting

4.1 Data Transfer Failures

I. **Data transfers through the GPIF II interface fail when the data bus is 32 bits wide and the interface frequency is 100 MHz (synchronous).**

- If the FX3 device is being clocked using a 19.2 MHz crystal or a 38.4 MHz clock input; the GPIF II hardware will be running at a default frequency of 96 MHz. At this frequency, the block cannot handle data that is incoming at the rate of 32 bits per 10 ns; and will result in a high frequency of overflow errors.

  This can be fixed by adjusting the internal clock dividers so that the GPIF II hardware runs at a frequency greater than 100 MHz. This can be done by setting the setSysClk400 field to CyTrue in the CyU3PSysClockConfig_t structure passed to the CyU3PDeviceInit.

II. **Data transfers through the GPIF II interface fail or get corrupted when the FX3 is driving the clock from the GPIF II interface or when GPIF II is used in asynchronous mode.**

- If the GPIF II interface is driving the clock out or if it is used in asynchronous mode, the data transfers may fail or get corrupted due to mismatch in timing parameters on the GPIF II interface.

  This can be fixed by enabling the DLL (delay locked loop) block of the FX3 device while configuring the GPIF II interface. For more information on how to configure the DLL, please refer to the Knowledge Base Article, Configuring EZ-USB® FX3™ GPIF-II DLL

5 SDK Tools Troubleshooting

5.1 Elf2img converter issues

I. Eclipse post-build step to generate binary image fails

- The Eclipse projects provided as part of the FX3 SDK include a post-built step that uses the elf2img.exe converter utility to generate a loadable firmware binary image. It has been seen that this step fails on some systems and throws error messages such as:

  process_begin: CreateProcess(NULL, elf2img.exe -i USBBulkSourceSink.elf -o USBBulkSourceSink.img -v, ...) failed.

  make (e=2): The system cannot find the file specified.

  This happens due to a Windows path resolution error. This can commonly be fixed by adding the following path to the system PATH environment variable. Restart the EzUsbSuite IDE after adding the following path.

  "C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\util\elf2img"

II. Programming firmware binary to I2C EEPROM fails

- The FX3 firmware binary (.img) format includes information on how to address the I2C EEPROM device when programming as well as during boot operations. This elf2img.exe converter takes in a command line parameter (-i2cconf) through which the user can specify this information.

  The default setting chosen corresponds to Atmel/ST Micro I2C EEPROMs with a capacity of 64 KB or more. This configuration expects that the first 64 KB of EEPROM can be found at I2C device address ‘b000, the next 64 KB at device address ‘b001 and so on. If the EEPROM device(s) being programmed have a different addressing scheme, the –i2cconf parameter should be used to specify the desired addressing scheme.

  Please refer to the $FX3_INSTALL_PATH\util\elf2img\readme.txt document in the SDK installation for information on the format of this parameter.

III. Boot from I2C/SPI is slow

- The FX3 firmware binary (.img) format specifies the speed at which the I2C/SPI interface is to be run during the boot operation. This information is also specified through the –i2cconf parameter to the elf2img.exe converter.

  The default operating speed for booting will be the lowest supported (100 KHz for I2C and 10 MHz for SPI). Please refer to the

$FX3_INSTALL_PATH\util\elf2img\readme.txt document in the SDK installation for information on how the frequency is specified through the i2cconf parameter.

5.2 Eclipse IDE issues

I. CX3 Configuration Project option not available

- If the File → New → Other → Cypress dialog does not list the “CX3 Configuration Project” option under Windows, it could be because the corresponding Eclipse plug-ins are failing to load due to permissions issues.

  This can be fixed by running the ezUsbSuite.exe application in administrator mode. Please refer to https://www.techadvisor.co.uk/how-to/windows/how-run-programs-as-administrator-in-windows-10-3632744/ for an example of how to make the “run as administrator” option persistent for the application.

II. The Problems tab shows errors after building a project

- The C syntax checker associated with the Eclipse IDE might flag errors even in cases where the compilation of a project is successful. This happens because the syntax checker is not able to locate definitions from header files located in an external folder like the FX3 library folder.

  Rebuilding the index for the project will resolve these issues. You can trigger the index build by right-clicking on the project name and choosing the Index → Rebuild option.