Using the OSR USB FX-2 Learning Kit V2.0

Board Rev: 00
Firmware Revision: 3.5
Document Revision: 2.1

This document describes how to use the OSR USB FX-2 Learning Kit, V2.0. This USB device was designed and built by OSR specifically for use in teaching software developers how to write drivers for USB devices. This document describes the basic functionality available on the board. The target audience for this document is Windows device driver writers who want to use or implement a driver for the board.

1 Configurations

The board is configured with the following VID, PID, and version:

Parameter	Value	Assigned To
Vendor ID	0x0547	Cypress Semiconductor (actually Anchor Chips, Inc).
Product ID	0x1002	Sample device
Version ID	0	

The vendor and product IDs used by the board are the same as those used by the development board supplied with the Cypress EZ-USB FX2 Development Kit (CY3681).

The board supports a single configuration. The board automatically detects the speed of the host controller, and supplies either the high or full speed configuration based on the host controller's speed. The alternate speed is also supplied as an "other speed configuration descriptor."

2 Interfaces

In both high-speed and full-speed mode, the board implements Interface 0.

3 Endpoints

The firmware supports 3 endpoints, as follows:

Endpoint Number	EP Type and Use	Packet Size in FULL SPEED	Packet Size in HIGH SPEED
1	Interrupt, IN from board to host	1 byte	1 byte
6	Bulk, OUT from host to board	64	512
8	Bulk, IN from board to host	64	512

Endpoint number 1 is used to indicate the state of the 8-switch switch-pack on the OSR USB-FX2 board. A single byte representing the switch state is sent (a) when the board is first stated, (b) when the board resumes after selective-suspend, (c) whenever the state of the switches is changed.

Endpoints 6 and 8 perform an internal loop-back function. Data that is sent to the board at EP6 is returned to the host on EP8.

Additional information about the how each endpoint works is provided in the following sections.

3.1 Endpoint 1 – Interrupt Endpoint Packet Format

The board sends an 8-bit value that represents the state of the switches on the switch pack from EP1. No more than 1 byte of data is ever sent at one time. The switch pack value is sent on startup, resume from suspend, and whenever the switch pack setting changes. Note that the firmware *does not de-bounce* the switch pack, so unless the switches are very quickly and firmly switched, one switch change can result in multiple bytes being sent. http://www.labbookpages.co.uk/electronics/debounce.html

Note that the bits indicating the switches are backwards with respect to the numbers on the switch pack. That is, the value that's sent in bit 0 by the firmware reflects the switch that's labeled "8" on the switch pack.

Endpoint one is single buffered. That is, the firmware will buffer the information from exactly one switch pack transition, and the switch pack's state is not queried again until the buffered transition data is read by the client driver.

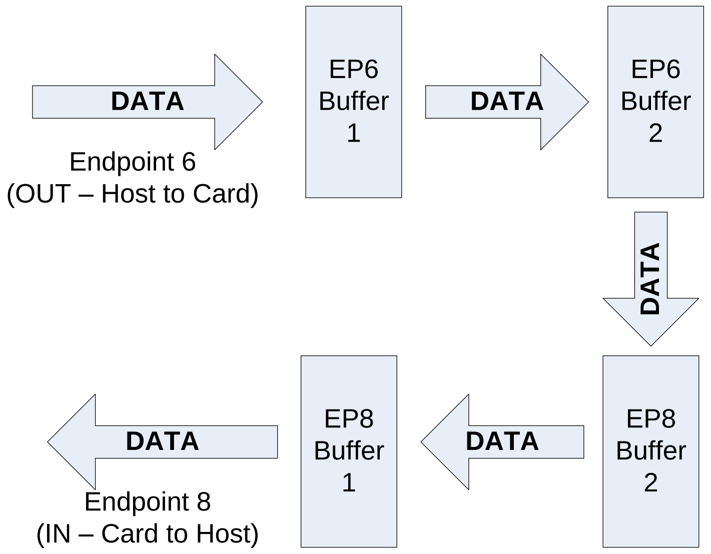
The best strategy to use in implementing client driver support for this feature is to always have a small number of reads for EP1 data queued.

3.2 Endpoint 6 and 8 Bulk Packet Format

The board receives data on EP6 and sends the same data back on EP8. The board does not change the values of the data that it receives on EP6, and does not internally create any data on EP8. Thus, the board performs a pure EP6 to EP8 loop-back operation.

The maximum packet size accommodated by EP6 and EP8 depends on the speed at which the board is operating. When operating in full speed, the maximum packet size is 64 bytes. When operating in high speed, the maximum packet size is 512 bytes.

Endpoint 6 and endpoint 8 are always double buffered. This means, for example, that up to 4 data packets (of any size, up to the maximum packet size) can be sent to EP6 before any data is read on EP8. The buffering arrangement that makes this possible is illustrated in Figure 1.



3.3 Figure 1 – How EP6 and EP8 Buffers Are Used

If you send 4 data packets to EP6 without issuing any reads to EP8, the first two data packets sent to EP6 will be accommodated by EP8's output buffers, and the last two data packets sent to EP6 will be buffered in EP6's input buffers. If you attempt to send a 5th data packet from the client driver to EP6 without sending any reads to EP8, the write request will wait in the Windows host (bus) driver until EP6 buffer space is available on the board. This buffering scheme applies regardless of the number of bytes that you send per packet. Even a packet that's one byte long will consume one complete endpoint buffer.

Continuing with this example, if you have sent 4 data packets to EP6, and then you issue a read to EP8, the contents of the first data packet sent to EP6 will be returned on EP8. As a result of the read to EP8, one endpoint buffer is again available on EP6.

4 Board LED Displays

The OSR FX2 firmware uses the LED displays on the board as described in this section.

4.1 7-Segment LED display

During initialization, the board displays the firmware major and minor revision numbers in the LED, 1 digit at a time, with a 500ms wait between digits.

After the firmware revision has been displayed, the firmware puts an "F" or "H" in the display to indicate whether the board firmware is running in full or high speed mode, respectively

Whenever the device is put into a selective suspend state by the USB Driver, the firmware will display a “S” in the 7-Segment display. When the board is returned to active state from suspended state, the firmware displays an “A” (for “active”) in the 7-segment display.

The 7-Segment display can also be set or read by the client driver using vendor-commands described later in this document.

4.2 LED Bar Graph Display

The LED bar graph display is used by default to indicate bulk packet activity. One light is lit on the LED bar graph for each 10 bulk packets that the board transmits. From 0 to 8 LED segments are displayed (note that top 2 LED segments are never lit, because they’re not connected to anything on the board). After all 8 LEDs have been lit, the display wraps-around to no LEDs being lit.

The contents of the bar graph can also be set or read by the client driver using vendor commands described later in this document.

5 Vendor Commands

The firmware supports a number of vendor specific commands which allow a driver to query the state of LEDs and switches as well as modify the state of LEDs.

Below, Figure 2 gives a general layout of the device, its various LEDs, Switches and Segmented LEDs. The reader should note the numbering on the various LEDs, Switches and Segmented LED because the input and returned bitmasks represent their placement on the device.

Three cases are specifically worth noting:

- The numbering of the LEDs in the bar graph block is not intuitive. When reading the state of the LEDs a returned value of 0x08, indicates that “LED 1” is on, whereas a returned value of 0x10, indicates that “LED 2” is on. The numbering shown in Figure 2 should make this clear. In case you didn't see it mentioned above, the top to LEDs in the bar graph aren't connected to anything... so you won't be able to light them.

- The switch numbers, as addressed by the device firmware, are reversed from the number printed on the switch pack on the board. For example, a returned value of 0x01 indicates that Switch 1 is toggled, even though it is numbered “8” on the switch pack on the board. Again, the numbering shown in Figure 2 shows the actual number used by the firmware.

- For the 7-segment display, the hex values shown in Figure 2 correspond to the value to be passed to the board to light the indicated segment of the display. For example, to display a “1” in the display, you’d want to light the two right-most vertical display segments. To light these two segments, send the value 0x6, which is 0x02 (to light the top right-most vertical bar) OR’ed with 0x04 (to light the bottom right-most vertical bar). If you're wondering what the funny little block with the value 0x08 is on diagram, it's for the decimal point. So, actually, it's an 8-segment display, isn't it.

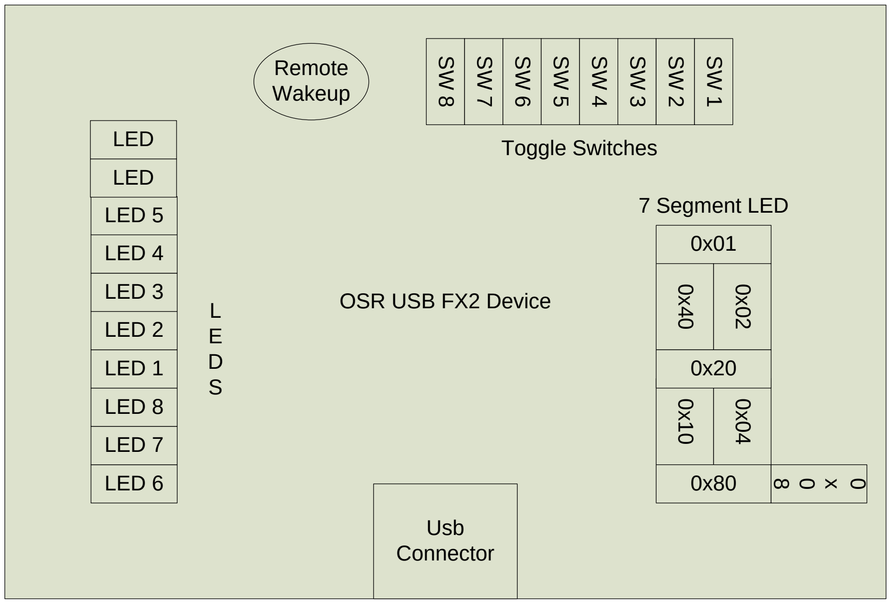


Figure 2 - OSR FX2 Device Layout

The Vendor commands are built, by the driver, using the **UsbBuildVendorRequest** routine. The routine is defined as follows:

```
VOID
UsbBuildVendorRequest(
   IN PURB     Urb,
   IN USHORT   Function,
   IN USHORT   Length,
   IN ULONG    TransferFlags,
   IN UCHAR    ReservedBits,
   IN UCHAR    Request,
   IN USHORT   Value,
   IN USHORT   Index,
   IN PVOID    TransferBuffer  OPTIONAL,
   IN PMDL     TransferBufferMDL  OPTIONAL,
   IN ULONG    TransferBufferLength,
   IN PURB     Link  OPTIONAL,
);
```

All vendor commands are issued with the USB function **URB_FUNCTION_VENDOR_DEVICE**, the commands and their requirements are listed below:

Vendor Command	Driver Input Requirements	Driver Output
0xD4 – READ 7 SEGMENT DISPLAY Placed in Request Field. Reads the current state of the 7-segment display	None.	Transfer Buffer = segmented LED bits, returned as a UCHAR value TransferBufferLength = size of Buffer containing State
0xD6 – READ SWITCHES Placed in Request Field. Reads the state of all switches on the board and returns a state bitmask.	None.	Transfer Buffer = switch bitmask, returned as a UCHAR value. TransferBufferLength = size of Buffer containing State
0xD7 – READ BARGRAPH DISPLAY Placed in Request Field. Reads the state of the LEDs in the bargraph display on the board and returns a state bitmask.	None.	Transfer Buffer = LED bitmask, returned as a UCHAR value. TransferBufferLength = size of Buffer containing State
0xD8 – SET BARGRAPH DISPLAY Placed in Request Field. Sets the state of all LEDs on the board via an input bitmask.	Value = LED BitMask (UCHAR Value) Bits 0 – 7, corresponding to segments of display to light. See Figure 1.	None.
0xD9 – IS HIGH SPEED Placed in Request Field. Returns the state of the device, for use on early Win2K systems that do not support bus driver's standard feature.	None.	Transfer Buffer = Speed State: 1 = HighSpeed, 0 = Full Speed, returned as a UCHAR value. TransferBufferLength = size of Buffer containing State
0xDB – SET 7 SEGMENT DISPLAY Placed in Request Field. Sets the state of the 7-segment LED display on the board via an input bitmask.	Value = LED BitMask, (UCHAR Value) Bits 0 – 7, corresponding to segments to light (see Figure 1).	None.

6 Advanced Usage Information

This section describes advanced information about how the board can be used. You do not have to read this section if you are only interested in knowing the information that's required to write a driver for the OSR USB FX2.

The OSR USB FX2 device is loosely based on the development board supplied with the Cypress EZ-USB FX2 Development Kit (CY3681). You can optionally use the Cypress demo driver EZUSBDRV.SYS to control the OSR USB FX2 board, because the board uses the Cypress USB FX2 chipset and the OSR board's vendor ID and product ID are identical to that of the Cypress development board.

Important Note
Using Cypress EZUSBDRV.SYS and Other Cypress Software

OSR does not endorse, nor do we recommend, that you use the Cypress demonstration driver with this board. Our experience with the EZUSBFX2 driver has not been very good, and we have experienced numerous system crashes when using it.

OSR does not provide support for, or assistance with using, any Cypress software including the EZUSBDRV or its associated applications.

Because the board is compatible with the Cypress USB driver, you can also use it with the test applications provided in the Cypress EX-USB Development Kit. This includes the Cypress EZ-USB control panel application, and the "bulk loop" sample application from the kit.

The OSR USB FX2 Learning Kit was specifically designed to allow different firmware to be downloaded to the device. If you're using the Keil uVision tools to build firmware for the device, we recommend setting the Code Range option to 0x80-0xFFF and the Xdata Range option to 0x1000 in the BL51 Locates tab of the Target Options dialog.

The OSR USB FX2 device does not support in-circuit reprogramming of the EEPROM. However, because the EEPROM is socketed, you can easily replace the EEPROM with one that you program yourself using an EPROM programmer.

Please do not even think about using the OSR USB FX2 board as a reference for any sort of hardware design. We built this board with no other goals in mind other than (a) creating a board that works for the purposes of learning how to write Windows drivers and (b) keeping the board as inexpensive as possible. In so doing, we've violated many, many, of Cypress' design recommendations.

7 Device Problems or Questions

OSR Open Systems Resources, Inc. warrants that this board will be free of defects in materials or workmanship for a period of 30 days from purchase. We know that's not very long, but this is a development/test board, and we're selling at our cost.

Answers to questions about how to use this board that aren't answered in this document can be directed to OSR via email to hwsupport@osr.com.

General questions about how to write device drivers, or how to debug a driver you're writing, should be directed to the NTDEV list that OSR administers. Visit the list server section of www.osronline.com to join the list or search the archives.

We don't want to sound cruel, BUT: Unless you're one of our current or former students (who are *always* welcome to send us questions on *any* topic), please don't send us random questions asking us to help you write or debug your driver. There are thousands of you out there and only a few of us, so we can't help everybody. Please use the NTDEV list. You'll find people on that list who know more about USB drivers than we do, anyways.

8 Frequently Asked Questions

Since release of this device, we've gotten many questions sent to hwsupport@osr.com, and we see some of them frequently enough that they qualify for inclusion here:

Q: *Will you guys send me the source code to the firmware on the device?*

A: No. It's not very different from the FX2 BULKLOOP sample in the Cypress Development Kit. It's not that we're trying to be difficult, but we just didn't write the code for external consumption, and we'd have to take all the bad words out of it first.

Q: *PLEASE?*

A: Sorry. No, we will not accept money (a rare statement from anyone here at OSR).

Q: *Will you guys send me the schematic for the device or explain the pin-outs from the chip? It's perfect for what I need, except for mumble.*

A: No. We didn't create this device as a cheap alternative to the Cypress USB-FX2 Development Kit. We created it as a cheap way for folks to learn how to write Windows drivers. Anyhow, if you can't figure out what pints connect to what from looking at a two-sided board and reading the chip spec sheet, you really don't have any business building your own hardware.

Q: *Do you have a Linux driver for the board?*

A: WHO do you think you're talking to? You want a Linux driver, call the guy in Finland. But seriously, tons of people **have** told us that they've written Linux drivers for the board. Just not us and, no, we don't remember who they are so please don't ask.

Q: *Is there someplace I can download a Windows WDF/KMDF driver for the board?*

A: No. The example KMDF driver is in the WDF DDK.

Q: *I'd like to buy xxx boards for the folks I work with. Can I get a volume discount?*

A: Did you miss the part about us selling the boards *at our cost*? So, if we were to give you a volume discount, it would cost us money. We might be able to work something out on shipping, though, if you want us to ship them via ground transportation and in one big box. One of our clients has purchased close to 200 of these boards so far, and they pay the price listed on the web site with an adjustment for shipping. We'll do the same for you if you'd like to buy, say, 25 or more boards at one time.

Q: *Hey, you guys screwed up! The bits on the switch pack are numbered backwards relative to the way they're reported by the firmware.*

A: Actually, this is intentional. Drivers are frequently called-on to "fix up" things that aren't perfect in the hardware. We didn't want to build a device that was artificially simple to use.

Q: *Are you guys too dumb to write firmware to de-bounce the switch pack?*

A: Yes, obviously, that's it. Please read the answer to the previous question.

9 Differences Between OSR USBFX-2 V1.0 and V2.0

The kits are functionally identical and are programmed the same way. The major changes we made between V1.0 and V2.0 were (a) we change the board layout to use surface-mount components wherever practical, (b) we added more copper to serve as a slightly better ground plane, (c) we change the BOM to make the board ROHS compliant.

10 OSR Thanks....

If you're using this board, you owe a major debt of thanks to John Nesselhauf, who donated his valuable, expert, time designing this board and laying it out, to make the board a reality. Thanks also go to (in alphabetical order) Randy Aull, Doron Holan, Nar Ganapathy, and Eliyas Yakub for feedback, ideas, and encouragement. And, as always, OSR remains grateful to its engineering team, without whose enthusiasm this project would have died. Again.