Project Objective
This code example demonstrates USB data streaming using isochronous endpoints in EZ-USB FX2LP™. The example demonstrates this data streaming in Windows XP 32-bit OS using the CY3684 kit.

Overview
This code example demonstrates data streaming in Isochronous transfer mode using CY3684 development kit for FX2LP. The isochronous data is continuously transferred to or from the endpoint buffers without any processing on the data. EP2 acts as continuously streaming isochronous OUT endpoint and EP6 acts as continuously streaming isochronous IN endpoint.

System Requirements

Software
CyConsole 1.7.0.2, USB Control Center 3.4.4.0 – Available through Suite USB 3.4
Keil uVision 2 – The 4K bytes evaluation version is available with the CY3684 Development kit. (This example can work with 4K version). For the full version please contact Keil.

Firmware
Firmware – Available in “Project\ Isochronous_Streaming” folder of the attached zip file of this example.

Hardware
The FX2LP development kit (CY3684) has been used to develop and test this code example. A detailed schematic of the DVK is provided in the hardware folder in the example code attached.

Prerequisites
The pre-requisites for testing this code example are
1) EZ-USB Advanced Development Kit Users Guide – contains information about EZ-USB Firmware Frameworks.
2) Device Descriptor Table Data – Knowledge base article on DSCR.a51 descriptor file.
3) CY3684 EZ-USB Getting Started Guide – contains information about on how to use the CY3684 kit for the first time.
4) AN65209 – Resources available for FX2LP

Block Diagram
Figure 1 shows the basic block diagram for testing this code example.

Figure 1. Block Diagram: Hardware Connections for Testing the Example
Operation

Isochronous Transfers in USB

In USB specifications, there are four types of transfers defined for varying needs of data communication. The types of USB transfers are:

- Control transfer
- Isochronous transfer
- Bulk transfer
- Interrupt transfer

Isochronous transfer is designed for data communication where latency is a critical factor against error-free communication. For example, audio or video streaming applications require data to be sent with a predefined rate for continuous reproduction of sound or video. These applications produce unwanted glitches if there are retries of data transfer due to losses. On the other hand, even though a frame is dropped or transferred with errors, this is less likely to produce any noticeable defects in the reproduced audio or video.

Isochronous transfers take place at a maximum rate of once per full-speed USB frame and up to three times per high-speed USB microframe. Isochronous transfer can contain up to 1023 bytes/packet in full-speed mode and up to 1024 bytes/packet in high-speed mode. For the guaranteed latency, this slot is kept dedicated on the USB bus for the isochronous transfers.

Further, to maintain the latency, the acknowledgement packet used for handshake in bulk, interrupt or control transfer is excluded in isochronous transfers. This minimizes the communication overhead and retries. Errors in isochronous transfers may be detected by CRC16 field in data packet. But, there is no mechanism for error correction. The erroneous packet may either be dropped or it can be used as it is with the errors.

Figure 2 and Figure 3 show examples of isochronous transfers. An IN transfer is completed in two packets. First packet is 'IN' token packet issued by the host. The device responds with the data packet. As mentioned above, there will not be any acknowledgement packet involved.

Figure 2. Isochronous IN Transfers

![Figure 2. Isochronous IN Transfers](image)

Similarly, an isochronous OUT transaction will have a token packet followed by a data packet. The device will not issue any acknowledgement for the received data. The following figure shows such an example.

Figure 3. Isochronous OUT Transfers

![Figure 3. Isochronous OUT Transfers](image)

As it can be seen in both figures, Data packets do contain toggling Data0 / Data1 fields those can be used for synchronization to a certain extent. Also, the CRC16 field can be used for error detection.
Firmware

This section describes how this example is made. The description is specific to FX2LP. The example is based on the standard framework for FX2LP device. Details about this framework is available in CY3684 Dvk Users Guide.pdf

In FX2LP, EP2, EP4, EP6 and EP8 are configurable as isochronous endpoints. Out of these, EP2 and EP6 are configured as quad buffered isochronous OUT and IN endpoints respectively with maximum packet size of 512 bytes. The other two endpoints, namely, EP4 and EP8 are not used in this example and hence they are disabled.

Make the following changes in the high-speed descriptor section in the dscr.a51 file

```assembly
;; Interface Descriptor
db   DSCR_INTRFC_LEN    ;; Descriptor length
db   DSCR_INTRFC        ;; Descriptor type
db   0                  ;; Zero-based index of this interface
db   0                  ;; Alternate setting
db   2                  ;; Number of end points
db   0ffH               ;; Interface class
db   00H                ;; Interface sub class
db   00H                ;; Interface sub sub class
db   0      ;; Interface descriptor string index

;; Endpoint Descriptor
db   DSCR_ENDPNT_LEN    ;; Descriptor length
db   DSCR_ENDPNT        ;; Descriptor type
db   02H                ;; Endpoint number, direction – EP 2 as out endpoint
db   ET_ISO             ;; Endpoint type
db   00H                ;; Maximum packet size (LSB)
db   02H                ;; Max packet size (MSB)
db   01H                ;; Polling interval

;; Endpoint Descriptor
db   DSCR_ENDPNT_LEN    ;; Descriptor length
db   DSCR_ENDPNT        ;; Descriptor type
db   86H                ;; Endpoint number, direction - EP6 as in endpoint
db   ET_ISO             ;; Endpoint type
db   00H                ;; Maximum packet size (LSB)
db   02H                ;; Max packet size (MSB)
db   01H                ;; Polling interval
```

Polling interval is defined as 1 because an isochronous transfer is scheduled to take place every frame or microframe according to the high-speed or full-speed USB operation respectively. The possible values of polling interval range from 1 to 16. If we specify “n” as polling interval value, the actual polling will take place every 2^(n-1) frames or microframes. Thus; here we have n = 1 which corresponds to 2^(0-1) = 2^0 = 1. Therefore, these endpoints will have a transfer in every frame or microframe.

In this example, the basic FX2LP program framework provided by cypress is modified in order to make the continuous isochronous streaming possible. The initialization routine defines the endpoint EP2 as isochronous quad buffered OUT endpoint and EP6 as isochronous quad buffered IN endpoint. Both the endpoints have 512-byte buffers. A valid data is written to all the four buffers of the IN endpoint EP6. All buffers of EP2 are armed and all buffers of EP6 are committed with byte count equal to 512 bytes. Apart from this, CPU clock is set to internal 48 MHz clock source and auto pointers are enabled.

In addition to these initializations, there is an additional parameter to be set for Isochronous IN endpoints in high-speed USB mode. The number of IN packets transferred per high-speed microframe can be set individually for each isochronous endpoint. USB 2.0 specification allows this to vary from one to three packets per microframe. In FX2LP, only EP2 supports all of the three options of sending one, two or three packets per microframe when configured as isochronous IN endpoint. EP6 may transfer one or two packets whereas EP4 and EP8 can transfer only one packet per USB microframe. This value is set by writing to EPxISOINPKTS register’s first and 0th bit. These bits are also called IPPF (input packets per frame) and contain directly the number of packets to be transferred per microframe. In this example, EP6 is configured to send single packet per microframe. Following is the part of initialization routine of the example.
EP2CFG = 0x90;  //Quad buffered Iso out endpoint. Buffer size- 512 bytes
SYNCDELAY;
EP4CFG &= 0x7F;  //EP4 Disabled
SYNCDELAY;
EP6CFG = 0xD0;  //Quad buffered Iso in endpoint. Buffer size- 512 bytes
SYNCDELAY;
EP8CFG &= 0x7F;  //EP8 Disabled
SYNCDELAY;

EP6ISOINPKTS = 0x01; //Set 1 IN packet per microframe for EP6IN
SYNCDELAY;

// out endpoints do not come up armed
// since EP2 is quad buffered we must write dummy byte count four times

EP2BCL = 0x80;    // arm EP2OUT by writing byte count w/skip.
SYNCDELAY;
EP2BCL = 0x80;
SYNCDELAY;
EP2BCL = 0x80;
SYNCDELAY;
EP2BCL = 0x80;
SYNCDELAY;

// enable dual autopointer feature
AUTOPTRSETUP |= 0x01;

//Write a valid data in quad buffered IN endpoint EP6 and commit the buffers
for (cnt = 0; cnt < 4; cnt++)
{
    APTR1H = MSB( &EP6FIFOBUF );
    APTR1L = LSB( &EP6FIFOBUF );

    for (i = 0x0000; i < 0x0200; i++)
    {
        EXTAUTODAT1 = 0x55; //Write data in EP6IN
    }

    EP6BCH = 0x02;
    SYNCDELAY;
    EP6BCL = 0x00; // commit EP6IN by writing bytecount
    SYNCDELAY;
}

In the polling routine of the framework, no data processing is done. The status of buffers is continuously monitored. When the IN endpoint buffer is found not to be full, it is immediately committed back without changing the buffer contents. This is done by writing byte count values. Similarly, when the OUT endpoint buffer is found not to be empty, it is immediately armed to the SIE without reading the incoming data. This is done by writing dummy byte count. This policy is adopted in order to allow continuous streaming of isochronous data to or from the device. Otherwise, an OUT packet may get dropped due to unavailability of buffer space without being noticed by the host. Also, an IN request may not be served at all by the device if no buffer is committed to SIE. In case of FX2LP, instead of not servicing an IN request at all, we can alternatively set the flag to ask SIE to send zero length packets when the buffers are not committed. The polling routine of the example takes care of continuous valid isochronous streaming. Following is the polling routine.
while (!(EP2468STAT & bmEP2EMPTY)) //Check if EP2OUT is not empty
{
    EP2BCL = 0x80; //Arm EP2OUT for next Iso transfer
}

while (!(EP2468STAT & bmEP6FULL))  //Check if EP6IN is not full
{
    EP6BCH = 0x02;
    SYNCDELAY;
    EP6BCL = 0x00; //Commit EP6IN by writing bytecount
    SYNCDELAY;
}

**Hardware Connections**

For this example, all the jumpers on CY3684 should be left to their default states. You can refer to the default jumper settings of the DVK in the Table 1 of EZ-USB_GettingStarted.pdf

The Schematic of CY3684 development board can be found inside the folder Hardware in the code example. The datasheets of components used like LT1763CS8-3.3, PCF8574T and 24LC128 can be found under the folder Docs\Datasheets.

**Testing the Example**

While testing this example, customers should be aware of some limitations of lower level windows USB drivers. These drivers allow committing the data for isochronous transfers only in the multiples of 8 times the maximum packet size of the endpoint. Therefore, for the 512 byte endpoints defined in the example, the transfers should always be of size equal to or multiple of 4096 bytes, approximately 8 times the packet size.

The example can be tested using CY3684 development board for FX2LP. CyConsole GUI that comes with CY3684 development board and SuiteUSB Control Center can be used for testing the example.

**Testing the Example Using Control Center**

1. The Control Center GUI that comes with SuiteUSB 3.4 can be used to initiate host side commands for USB devices.
2. Connect the CY3684 FX2LP development board to the PC with EEPROM ENABLE switch in NO EEPROM position.
3. Now the board enumerates with the default internal descriptor. Use the correct CyUSB.inf available in the link Drivers for FX1/FX2LP to bind with the device.
4. Program the FX2LP RAM by clicking RAM option in the Program FX2 menu of Control Center. The Isochronous_Streaming.hex under the folder “Project\Isochronous_Streaming” in the code example attachment should be selected to program the RAM.
5. After the RAM is programmed, the FX2LP remunerates with descriptors as defined in the example.
6. Then Windows should pop up asking to bind the driver. Use the correct CyUSB.inf available in the link Drivers for FX1/FX2LP to bind with the device.
7. After the device is properly bound to the drivers, you may notice the Device Tree portion getting modified according to the descriptors as defined in the dscr.a51 file of the example. You can select EP2 for testing an OUT isochronous streaming.
8. Go to Data Transfer tab page. You can enter any data that you want to transfer into either the Text or Data field of this tab. The Bytes to transfer field gets updated as you go on writing the data. When you have finished entering the data, this field shows total number of characters (bytes) entered. But, to meet the OS driver requirements, you should change this value manually to multiples of 4096 bytes. The Control Center utility automatically appends your data with appropriate number of zeros to make the transfer length equal to the value in Bytes to transfer field. Click Transfer Data-OUT to initiate the isochronous OUT streaming.
Figure 4. Testing Isochronous OUT transfer using Control Center

Minimum Length of Isochronous Transfer is 8 times max packet size

**Note** Any value other than multiples of 4096 written in the **Bytes to Transfer** field of **Data Transfer** tab throws an error due to driver limitations.

8. To check an IN isochronous streaming, select EP6 from the device tree. Select **Bytes to Transfer** field in the **Data Transfer** tab to be a multiple of 4096 bytes. Click **Transfer Data-IN** to initiate an isochronous IN transfer. **Control Center** displays the data that was written into these endpoint buffers in the **TD_Init()** routine. The same data appear multiple times. This can be expected since the polling routine in the program commits these buffers without modifying the data.
Figure 5. Testing Isochronous IN Transfer using Control Center

Test the Example Using CyConsole

1. The CyConsole utility comes with SuiteUSB 3.4 installation.
2. Connect the board to the PC with EEPROM ENABLE switch in NO EEPROM position.
3. Now the board enumerates with the default internal descriptor. Use the correct CyUSB.inf available in the link Drivers for FX1/FX2LP to bind with the device.
4. Download the Isochronous_Streaming.hex under the folder “Project\Isochronous_Streaming” in the code example attachment to the RAM using the download button of the CyConsole utility.
5. After the code is downloaded, the FX2LP remunerates with descriptors as defined in the example.
6. Then windows should pop up asking to bind the driver. Use the correct CyUSB.inf available in the link Drivers for FX1/FX2LP to bind with the device.
7. Once the device is bound to the drivers, you may see the endpoints appearing in the Iso Trans toolbar’s Pipe dropdown menu. You can select EP2 for testing an OUT isochronous streaming.
8. Select Length in Iso Trans toolbar as multiples of 4096 bytes. Click Iso Trans to initiate streaming of the isochronous data into the selected endpoint. By default, CyConsole sends an incrementing hex values as the isochronous data being displayed on CyConsole.

Note Any value other than multiples of 4096 written in the Bytes to Transfer field of Data Transfer tab throws an error due to driver limitations.

9. The code can also be tested using the Isochronous_Streaming.iic under the folder “Project\Isochronous_Streaming” by loading the .iic file to EEPROM 24LC128 using the 64 KB EEPROM option in the Program FX2 menu of Control Center. Before programming the EEPROM using Control Center, make sure that the EEPROM ENABLE switch is enabled and EEPROM SELECT switch is in LARGE EEPROM position.
Note Any value other than multiples of 4096 written in the Length field of Iso Trans toolbar throws an error due to low-level Windows driver limitations.

9. To check an IN isochronous streaming, select EP6 from the Pipe dropdown menu in Iso Trans toolbar. Select Length to be a multiple of 4096 bytes. Click Iso Trans to initiate an isochronous IN transfer. Control Center displays the data that was written into these endpoint buffers in the TD_Init() routine. The same data appear multiple times. This can be expected since the polling routine in the program commits these buffers as they are without modifying the data.

Note Any value other than multiples of 4096 written in the Length field of Iso Trans toolbar throws an error due to driver limitations.

10. The code can also be tested using the Isochronous_Streaming.iic under the folder "Project\ Isochronous_Streaming" by loading the .iic file to EEPROM 24LC128 using the CyConsole utility. Before programming the EEPROM using CyConsole utility make sure that the EEPROM ENABLE switch is enabled and EEPROM SELECT switch is in LARGE EEPROM position.
Throughput Measurement

To measure the data rate of the USB communication, Screamer application that comes with SuiteUSB installation can be used. In order to use this application for measurement of USB throughput, VID and PID should be declared as 0x04B4 and 0x1003 respectively.

To test the example throughput, keep the maximum possible values of Packets per Xfer and Xfers to Queue fields (Both 128). This shows the throughput of continuous streaming transfers.

Following are the figures of the throughput for EP2 OUT and EP6IN. The OUT endpoint has a throughput of 4.6 MB/s and IN endpoint has a throughput of 4 MB/s.

Figure 8. Measuring Throughput of Example using Screamer Application

Related Documents

Designing for a streaming application: Streaming Data Through Isochronous/Bulk Endpoints on EZ-USB FX2(TM) and EZ-USB FX2LP(TM) - AN4053