AN63787

EZ-USB® FX2LP™ GPIF and Slave FIFO Configuration Examples Using 8-bit Asynchronous Interface

Author: Gayathri Vasudevan
Associated Project: Yes
Associated Part Family: CY7C68013A/14A/15A/16A
Software Version: Keil µVision2
Related Application Notes: None

AN63787 discusses how to configure the general programmable interface (GPIF) and slave FIFOs of EZ-USB FX2LP™, in both manual mode and auto mode, to implement an 8-bit asynchronous parallel interface. This application note is tested with two FX2LP development kits connected in back-to-back setup, the first one acting in master mode and the second in slave mode.

Contents

Overview

Cypress FX2LP is one of the most popular programmable high-speed USB controllers in the industry. This application note demonstrates the implementation of manual mode and auto mode operation in both GPIF and slave FIFOs of EZ-USB FX2LP. The FX2LP to FX2LP back-to-back setup is used, with one FX2LP operating in master (GPIF) mode and the other in slave mode. The GPIF is a programmable 8- or 16-bit parallel interface that reduces system costs by providing a glueless interface between the EZ-USB FX2LP and different types of external peripherals. This application note describes the manual mode and auto mode operation using this setup:

- **In manual mode:** Two FX2LP chips are interfaced with each other; one of the chips functions in GPIF manual mode and the other functions in slave FIFO manual mode. In certain applications, the EZ-USB CPU must be present in the data path to interpret or modify the data before it is passed to the external device or host computer. In this section, a bidirectional parallel interface is implemented to show how the master CPU (FX2LP in GPIF manual mode) and slave CPU (FX2LP in slave FIFO manual mode) perform data modification.
- **In auto mode:** Two FX2LP chips are interfaced with each other; one of the chips functions in GPIF auto mode and the other in slave FIFO auto mode. The EZ-USB CPU is usually removed from the data path to maximize bandwidth.

For more information on these modes, see the chapters on Slave FIFOs and General Programmable Interface in the EZ-USB Technical Reference Manual. See application note EZ-USB FX2LP™ GPIF Design Guide – AN66806 for more details on FX2LP GPIF design.

System Requirements

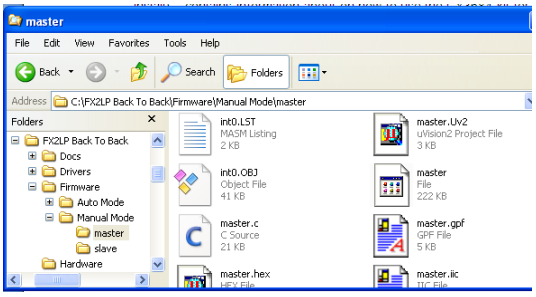
Hardware

Two FX2LP Development Kit (CY3684) boards are used as the development and testing platforms for this example. A detailed schematic of the development kit (DVK) is provided in the 'hardware' folder, included in the attachment. More information about the board is available in the 'EZ-USB Advanced Development Board' section of the 'EZ-USB_GettingStarted' document, available at `C:\Cypress\USB\doc\General` (after DVK install).

Software

- Control Center - Available through Suite USB 3.4.
- Keil uVision 2 – The 4-Kbyte evaluation version is available with the CY3684 DVK. For the full version, contact Keil.

Project Directory Structure



FX2LP Back-to-Back

This is the project folder included with this application note and contains the following:

- Docs
- Drivers
- Firmware
- Hardware

Documentation

Address: FX2LP Back To Back\Docs

Contains the related datasheets and the Technical Reference Manual for EZ-USB FX2LP

Drivers

Address: FX2LP Back To Back\Drivers

Contains the *CyUSB.inf* and *CyUSB.sys* files for FX2LP; use *CyUSB.inf* in the path shown in the following table for the operating system you are using.

Operating System	Folder Path
Windows XP 32-bit	wxp\x86
Windows XP 64-bit	wxp\x64
Windows 7 32-bit	wlh\x86
Windows 7 64-bit	wlh\x64
Windows Vista 32-bit	wlh\x86
Windows Vista 64-bit	wlh\x64

Firmware

Address: FX2LP Back To Back\Firmware

Contains the following folders:

- Manual Mode
- Auto Mode

Each of these folders contains the Master and Slave subfolders.

Master

Address:
FX2LP Back To Back\Firmware\Manual Mode\master and FX2LP Back To Back\Firmware\Auto Mode\master

Contains the firmware needed for the master FX2LP. *Master.uv2* is the project file. This folder also has the *master.hex* and *master.iic* files to be loaded into the RAM and large EEPROM, respectively.

Slave

Address:
FX2LP Back To Back\Firmware\Manual Mode\slave and FX2LP Back To Back\Firmware\Auto Mode\slave

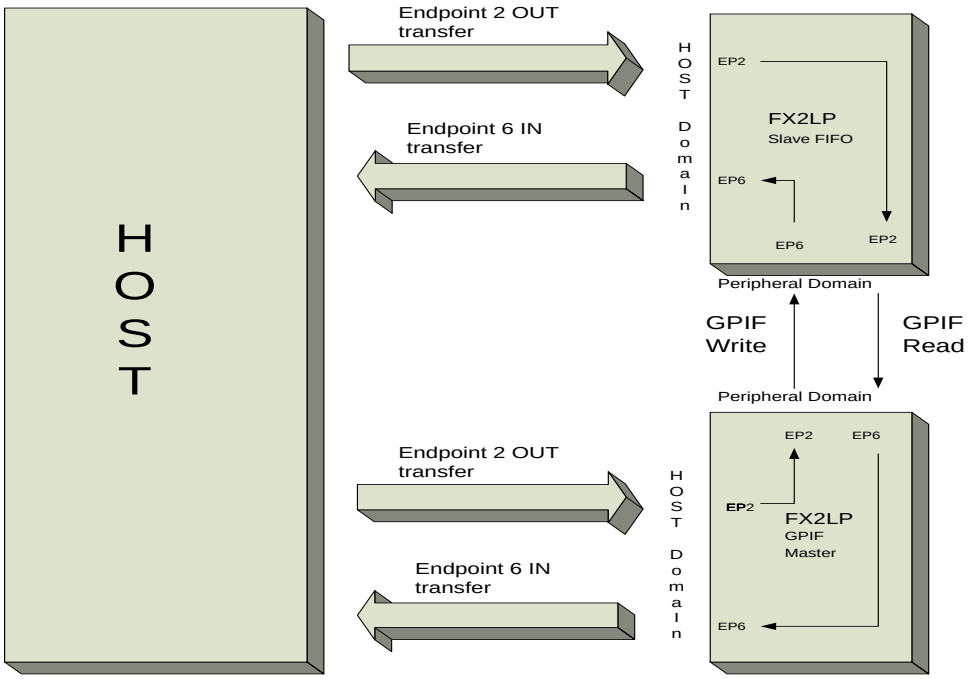
Contains the firmware needed for the slave FX2LP. *Slave.uv2* is the project file. This folder also has the *slave.hex* and *slave.iic* files to be loaded into the RAM and large EEPROM, respectively.

Prerequisite Information

You should have the following information before testing this application note:

- CY3684 EZ-USB_GettingStarted.pdf and DVK Users Guide.pdf – These documents contain information on how to use the CY3684 kit for the first time and is available at `C:\Cypress\USB\doc\General` (after DVK install).
- Getting Started with FX2LP™ - This document serves as a starting point for the new user to get familiar with FX2LP.

Block Diagram



Two FX2LP chips are interfaced with each other: one of the chips functions in GPIF mode and the other functions in slave FIFO mode. Both the FX2LPs are configured to have the following endpoints:

EP2 – BULK OUT, 512 Byte, double buffered

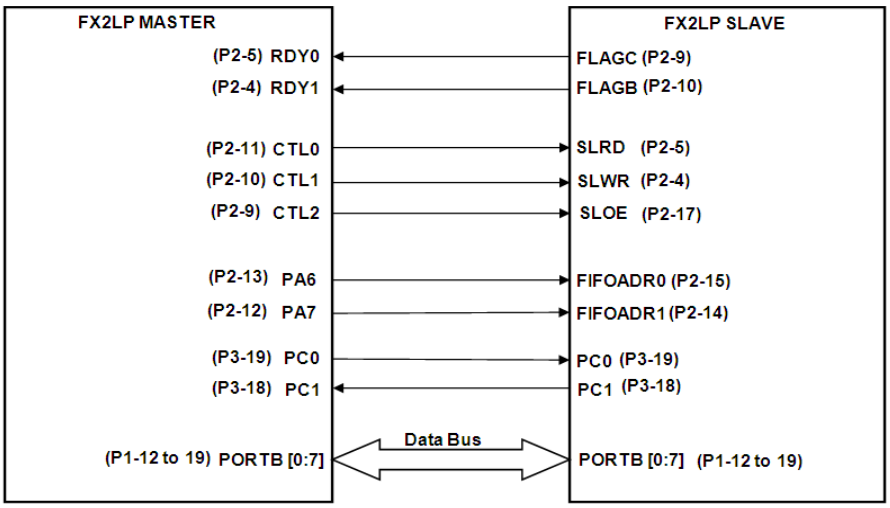
EP6 – BULK IN, 512 Byte, double buffered

The data sent from the host to the EP2 OUT endpoint of the master will be transferred to the EP6 IN endpoint of the slave. Similarly, the data transferred from the host to EP2 OUT of the slave will be transferred to EP6 IN of the master. Thus, the data read from EP6 IN of the slave (master) will be the same as that transferred to EP2 OUT of the master (slave).

To demonstrate the difference between auto and manual modes, the data is modified with both the master FX2LP and slave FX2LP in manual mode.

Hardware Connections

Figure 1. Hardware Connections (both master and slave are of 128-pin package)



This section discusses the required hardware interconnect between the two FX2LPs. In Figure 1, markings inside brackets denote the name of the headers in the FX2LP DVK. The data bus is eight bits wide and the interface is asynchronous. For the slave FX2LP:

- FLAGB and FLAGC are used to report the status of the slave FIFOs.

  FLAGC – EP2EF

  FLAGB – EP6FF

- The slave FIFO 'control' pins used are: SLOE (Slave Output Enable), SLRD (Slave Read), SLWR (Slave Write), and FIFOADR[1:0] (FIFO Select).

- The FIFOADR[1:0] pins select which of the four FIFOs is connected to the data bus (controlled by the external master).

The master FX2LP uses three control signals (CTL0, CTL1, and CTL2) and two ready signals (RDY0 and RDY1) to interface with the slave FX2LP, as shown in the figure. The master FX2LP uses Port A pins [6, 7] to select the FIFOs.

For manual mode operation, two additional lines are required for a handshake between the slave and master devices to implement the master-out-slave-in transfer. PC0 and PC1 are used for this purpose. More details on the use of these pins are given in later sections.

For more information, see the Slave FIFOs and General Programmable Interface sections in the EZ-USB Technical Reference Manual.

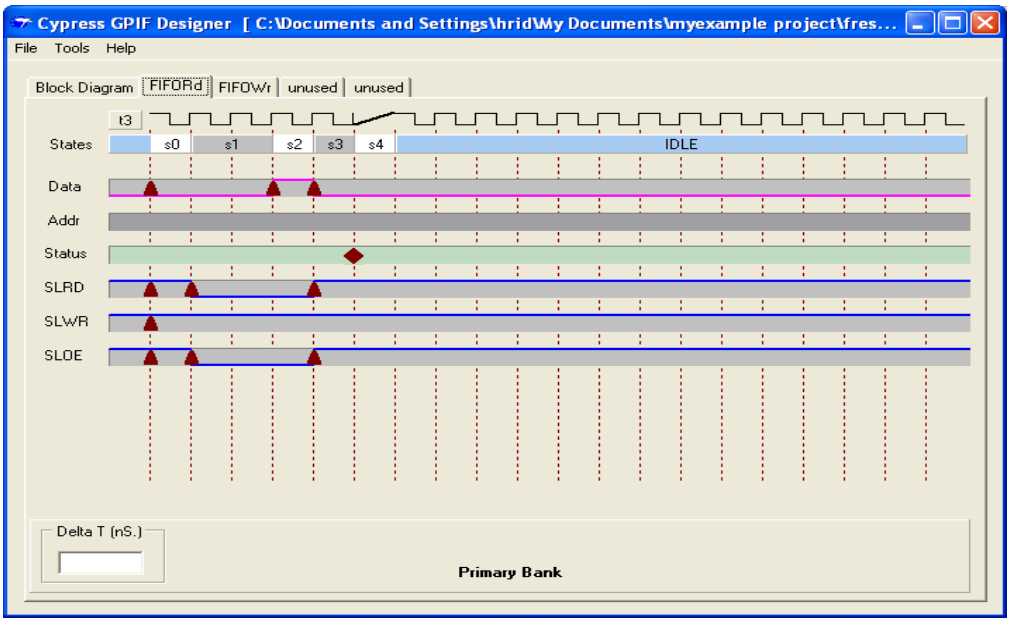
GPIF Waveforms

The GPIF Designer utility is used to create the waveform descriptors to read from and write into the slave FX2LP. First, you must define the interface and then create the waveforms using the utility. After the interface is configured, create the read and write waveforms using the communication that takes place over the interface. Both the GPIF read and write waveforms follow the logic of passing through N iteration (S1 - S2 -...- Idle), where N is the transaction count specified by loading into the registers GPIFTCB3:0 with the desired number of transactions. This application note uses two waveforms, one to read and the other to write into the slave FX2LP; these are shown in the screenshots in the following pages.

Read Waveforms

Read waveforms are designed to read data from the OUT Endpoint (EP2) of the slave FX2LP into the IN Endpoint (EP6) of the master FX2LP. They must satisfy the timing requirements of the various signals involved in the read cycle of the FX2LP.

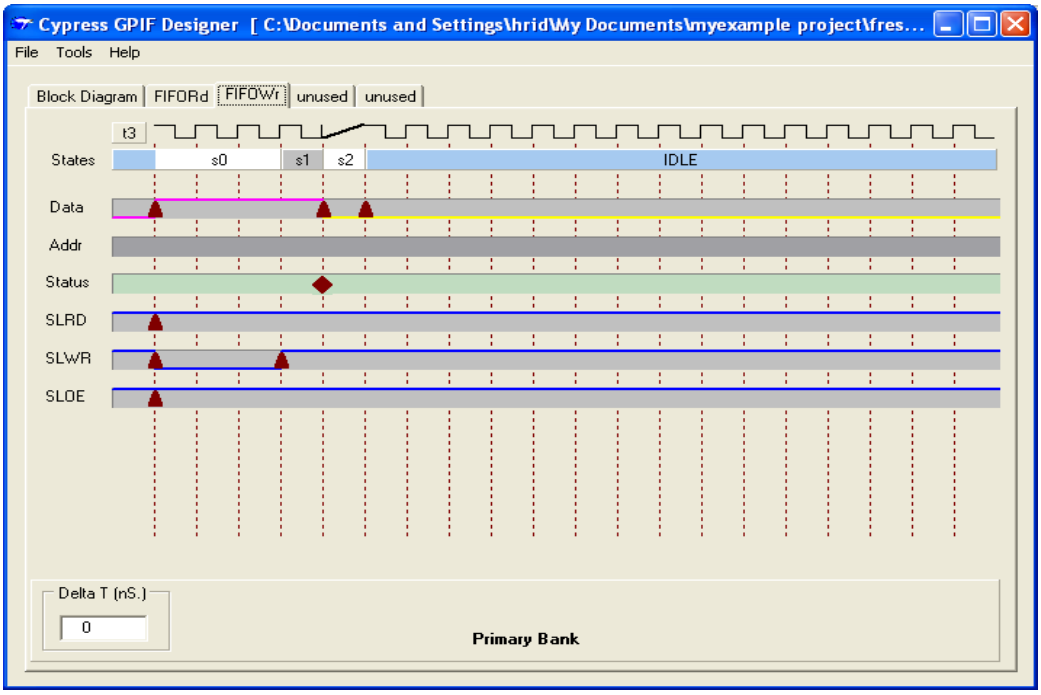
Figure 2. FIFO Read Waveform



Write Waveforms

Write waveforms are designed to write data from the OUT Endpoint (EP2) of the master FX2LP into the IN Endpoint (EP6) of the slave FX2LP. They must satisfy the timing requirements of the various signals involved in the write cycle of the FX2LP.

Figure 3. FIFO Write Waveform



Exporting GPIF Waveforms

To export the waveforms to a C file and include it in the firmware project, follow these steps. In the GPIF Designer utility:

1. Select **Tools > Export to gpif.c File**
2. Save the file as gpif.c in the project directory

The gpif.c file is added into your project (Master.uv2). For more information about using the GPIF Designer utility, see Interfacing SRAM with FX2LP over GPIF.

Manual Mode Operation of FX2LP in GPIF and Slave FIFO Configuration

Firmware

To view the code for the master, open *master.uv2* from "FX2LP Back To Back\Firmware\Manual Mode\master\". To view the code for the slave, open *slave.uv2* from "FX2LP Back To Back\Firmware\Manual Mode\slave\".

Configuring FX2LP in GPIF Manual Mode

- Configure the REVCTL.0 bit to '1'. Therefore, both IN and OUT packets can be edited, sourced, skipped, and committed.
- Set REVCTL.1 bit to 1; this disables the auto arming of OUT endpoints if it transitions from AUTOUT=0 to AUTOOUT=1.

```
REVCTL = 0x03;      // CPU can source and edit both IN and OUT packets
SYNCDELAY;
```

- Set EP2FIFOCFG and EP6FIFOCFG to '0'; this configures the endpoints in 8-bit manual mode.

```
EP2FIFOCFG = 0x00; // manual out mode, 8 bit data bus
SYNCDELAY;
EP6FIFOCFG = 0x00; // manual in mode, 8 bit data bus
SYNCDELAY;
```

- Configure PC0 (Txn_Over) as an output pin and PC1 (Pkt_Committed) as an input pin.

```
/* #define statements before TD_INIT()*/
#define Txn_Over PC0
#define Pkt_Committed PC1

/* code snippets from TD_INIT() */
PORTCCFG = 0x00;  //configure port C as an I/O port
OEC= 0xFD;        //Txn_Over configured as O/P, Pkt_Committed configured as I/P
IOC = 0xFD;
```

To implement the slave FIFO in "manual in" mode, two lines are required to handshake between the slave and the master. Name the two lines as Txn_Over and Pkt_Committed. Txn_Over is asserted by the master when a GPIF transaction is completed. Pkt_Committed line is toggled by the slave whenever it commits a packet.

- Txn_over the de-asserted state: Txn_Over = 1, indicates the master can start next GPIF write transaction (writing from EP2 OUT of the master to EP6 IN of the slave).
- Txn-over the asserted state: Txn_Over = 0, indicates that a GPIF write transaction has been completed by the master and that the slave can now start reading from its IN endpoint (EP6).
- Pkt_Commited: Every toggle of this signal means that the previous packet sent by the GPIF master has been processed and committed by the slave, it is now ready to accept another packet.

GPIF Manual OUT Mode

- In the master, inside Td_Poll(), it is continuously checked to see whether the Pkt_Committed pin is toggled.

```
if( Pkt_Committed == ~b)
{
    b = Pkt_Committed;    // store the current state of Pkt_Committed in variable b so
                          // that the next toggle can be detected
    Txn_Over = 1;
}
```

- When a toggle is detected (that is, when the just-received packet in the IN endpoint of the slave is committed to its USB domain), Txn_Over is de-asserted, indicating that the master can start the next GPIF transaction.

```
      if( (!( EP2468STAT & 0x01 )) )   //if EP2 not empty, modify packet and commit it to
peripheral
      {                                //domain
           SYNCDELAY; //
          EP2FIFOBUF[0] = 0x01;  // editing the packet
          SYNCDELAY;
          EP2FIFOBUF[1] = 0x02;
          SYNCDELAY;
          EP2FIFOBUF[2] = 0x03;
          SYNCDELAY;
          EP2FIFOBUF[3] = 0x04;
          SYNCDELAY;
          EP2FIFOBUF[4] = 0x05;
          SYNCDELAY;
          EP2BCH = 0x02;
          SYNCDELAY;
          EP2BCL = 0x00;       // commit edited pkt. to interface fifo
          SYNCDELAY;
      }
      if ( ! (EP24FIFOFLGS & 0x02) )
      {
          if((SLAVENOTFULL) && (Txn_Over == 1))
          {
             if( GPIFTRIG & 0x80 )               // if GPIF interface IDLE
                   {
                 PERIPH_FIFOADR0 = 0;            // FIFOADR[1:0]=10 - point to peripheral
EP6
                 PERIPH_FIFOADR1 = 1;
                 SYNCDELAY;
                 if(enum_high_speed)
                 {

                    SYNCDELAY;
                    GPIFTCB1 =   0x02;        // setup transaction count
                     SYNCDELAY;
                    GPIFTCB0 =   0x00;
                    SYNCDELAY;
                 }
                 else
                 {
                     SYNCDELAY;
                     GPIFTCB1 = 0x00;            // setup transaction count
                     SYNCDELAY;
                     GPIFTCB0 = 0x40;
                     SYNCDELAY;
                 }
                 SYNCDELAY;
                 GPIFTRIG = GPIFTRIGWR | GPIF_EP2;  // launch GPIF FIFO WRITE Transaction
from EP2                        SYNCDELAY;
                 while( !( GPIFTRIG & 0x80 ) )      // poll GPIFTRIG.7 GPIF Done bit
                       {
                      ;
                 }
                  SYNCDELAY;
                 Txn_Over = 0;    //assert Txn_Over line to indicate that packet has been
transmitted

             }
          }
      }
```

- If there is a packet in the USB domain of the EP2 OUT Endpoint, the first five bytes of the packet are modified and then committed.
- Then, if there is space in the +EP6 IN endpoint of the slave FX2LP and if the Txn_Over is not asserted, then the GPIF write transaction is triggered.
- GPIF Write: Writing from EP2 OUT of the master FX2LP to EP6 IN of the slave FX2LP.
- After that transaction is over (determined by polling the 'Done' bit in the GPIFTRIG register), Txn_Over is asserted to give an indication to the slave that one transaction is over and it can start reading that packet.

GPIF Manual IN Mode

```
  if ( GPIFTRIG & 0x80 )              // if GPIF interface IDLE - triggering gpif IN
transfers
  {
    PERIPH_FIFOADR0 = 0;
    PERIPH_FIFOADR1 = 0;              // FIFOADR[1:0]=00 - point to peripheral EP2
    SYNCDELAY;
    if ( SLAVENOTEMPTY )                // if slave is not empty
    {
      if ( !( EP68FIFOFLGS & EP6FULL ) )     // if EP6 FIFO is not full
      {
        if(enum_high_speed)
        {
          SYNCDELAY;
            GPIFTCB1 = 0x02;                // setup transaction count
            SYNCDELAY;
            GPIFTCB0 = 0x00;
            SYNCDELAY;
       }
       else
       {
          SYNCDELAY;
          GPIFTCB1 = 0x00;                // setup transaction count
            SYNCDELAY;
          GPIFTCB0 = 0x40;
          SYNCDELAY;
       }

      GPIFTRIG = GPIFTRIGRD | GPIF_EP6; // launch GPIF FIFO READ Transaction to EP6 FIFO
      SYNCDELAY;
```

- In GPIF manual IN mode, check the slave's 'Empty' flag to verify that there is a packet to be read from the slave device. If the empty flag is de-asserted, then a FIFO read transaction to read 512 bytes from the slave is initiated.
- GPIF Read: Reading from EP2 OUT of slave FX2LP into EP6 IN of the master FX2LP.
- The GPIFTRIG.7 bit is continuously polled to wait until the GPIF transaction has ended.

```
    while( !( GPIFTRIG & 0x80 ) )     // poll GPIFTRIG.7 GPIF Done bit
      {
        ;
      }
      EP6FIFOBUF[ 4 ] = 0x05; //edit the last five packets before committing
    EP6FIFOBUF[ 3 ] = 0x04;
    EP6FIFOBUF[ 2 ] = 0x03;
    EP6FIFOBUF[ 1 ] = 0x02;
    EP6FIFOBUF[ 0 ] = 0x01;
    SYNCDELAY;
    SYNCDELAY;
    EP6BCH = 0x02;            //committing the packet
```

```
      SYNCDELAY;
      EP6BCL = 0x00;
      SYNCDELAY;
    }
  }
```

- When the transaction ends, the first five bytes of the packet are modified and then committed to the host domain by writing the number of bytes to be committed into the EP6BCH/BCL registers.

Configuring FX2LP in Slave FIFO Manual Mode

TD_Init() configures endpoint 2 as an OUT endpoint and endpoint 6 as an IN endpoint, both functioning in 8-bit manual mode.

Slave FIFO Manual IN Mode

```
      if (PC0 == 0 &&(!(EP68FIFOFLGS & 0x02)))
      {
            EP6FIFOBUF[ 507 ] = 0x05; //edit the last five packets before committing
            EP6FIFOBUF[ 508] = 0x04;
            EP6FIFOBUF[ 509 ] = 0x03;
            EP6FIFOBUF[ 510 ] = 0x02;
            EP6FIFOBUF[ 511 ] = 0x01;
            SYNCDELAY;
            SYNCDELAY;
            EP6BCH = 0x02;            //commiting the packet
            SYNCDELAY;
            EP6BCL = 0x00;
            SYNCDELAY;
            PC1 = ~PC1;               //toggle PC0 to indicate that the buffer has been
passed
            while(  PC0 != 1);        //wait for PC0 to become high again. This is to
prevent
                                    //committing multiple packets at a single assertion of
PC0
      }
```

- In Td_Poll(), PC0 (which is tied to the Txn_Over of the master FX2LP) is continuously checked to verify that Txn_Over is asserted to indicate the end of a FIFO write transaction from the master. This check is necessary; otherwise, the data in the IN FIFO of the slave FX2LP can get committed to the USB domain before the entire packet is transferred into it from the master. When this happens, the host will see data being split into a number of smaller packets.
- When the data packet is committed, the slave toggles the PC1 (which is tied to the Pkt_Committed line of the master FX2LP) to let the master device know that the packet is committed and the next packet transaction can now start. The master also de-asserts the Txn_Over line when it finds that the Pkt_Committed line has been toggled.
- When PC0 (Txn_Over) is asserted and when EP6 IN is not empty, then the last five bytes of the packet is modified in the slave FX2LP and then committed to the host domain.

Slave FIFO Manual OUT Mode

```
      if( !( EP2468STAT & 0x01 ) )   //if EP2 not empty, modify packet and commit it to
peripheral
      {                            // domain
           SYNCDELAY;
           EP2FIFOBUF[511] = 0x01; // editing the packet
           SYNCDELAY;
           EP2FIFOBUF[510] = 0x02;
           SYNCDELAY;
           EP2FIFOBUF[509] = 0x03;
           SYNCDELAY;
           EP2FIFOBUF[508] = 0x04;
           SYNCDELAY;
           EP2FIFOBUF[507] = 0x05;
           SYNCDELAY;
```

```
        EP2BCH = 0x02;
        SYNCDELAY;
          EP2BCL = 0x00;       // commit edited pkt. to interface fifo
          SYNCDELAY;

    }
```

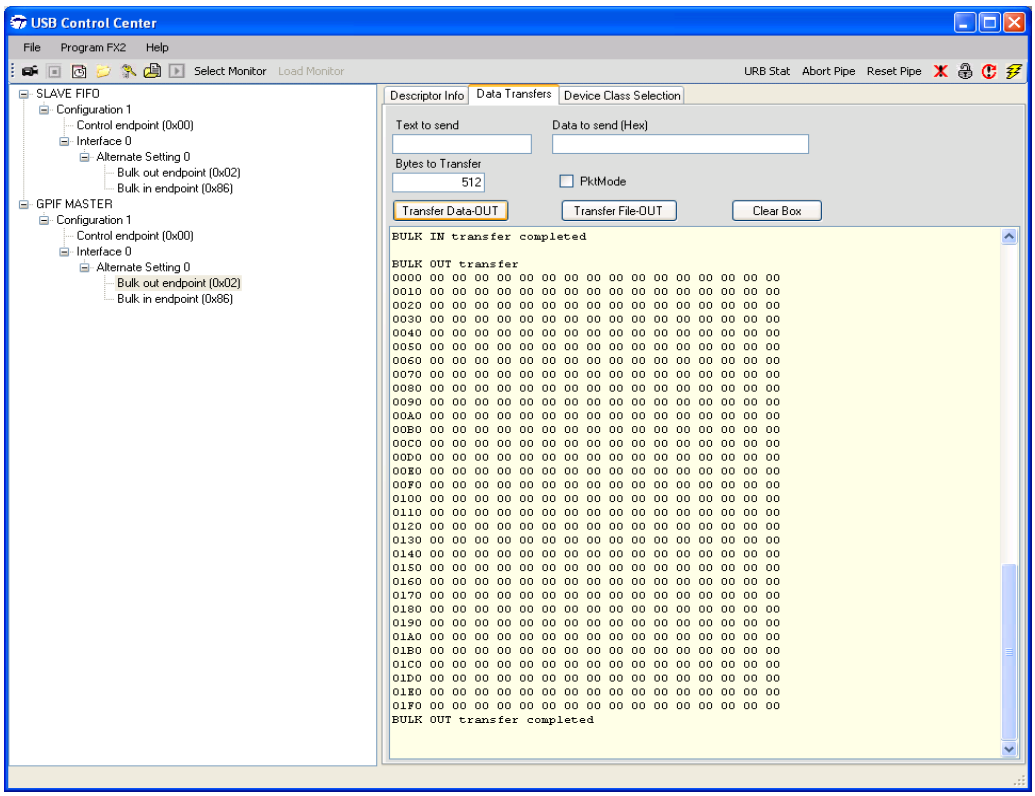
The slave FIFO manual OUT mode firmware checks if the 'Empty' flag of the OUT endpoint is de-asserted. If it is de-asserted, then the last five bytes of the packet in the EP2 OUT endpoint of the slave FX2LP are edited and then the packet is committed to the peripheral domain.

Testing the Project (Manual Mode)

1. Download and install Cypress SuiteUSB 3.4. This installs a utility named Control Center.
2. Connect both the CY3684 boards to the PC, as shown in Figure 1. They enumerate with the default internal descriptor. Use the *CyUSB.inf* file in the Drivers folder to bind with the device. For help with binding the driver, see *MatchingDriverToUSBDevice.htm* in the Drivers folder.
3. Download *slave.iic* in the attachment along with the code example, available in "\FX2LP Back To Back\Firmware\Manual Mode\slave" to the large EEPROM of the slave FX2LP using the Control Centre utility. Reset the device.
4. Pop-up windows appear asking to bind the driver. Use the correct *CyUSB.inf* file (according to the OS used) located in the Drivers folder to bind.
5. Download *master.iic* in the attachment, available in "\FX2LP Back To Back\Firmware\Manual Mode\master", to the large EEPROM of the master FX2LP using the Control Centre utility. Reset the device.
6. Pop-up windows appear asking to bind the driver. Use the same *CyUSB.inf* file located in the Drivers folder to bind.

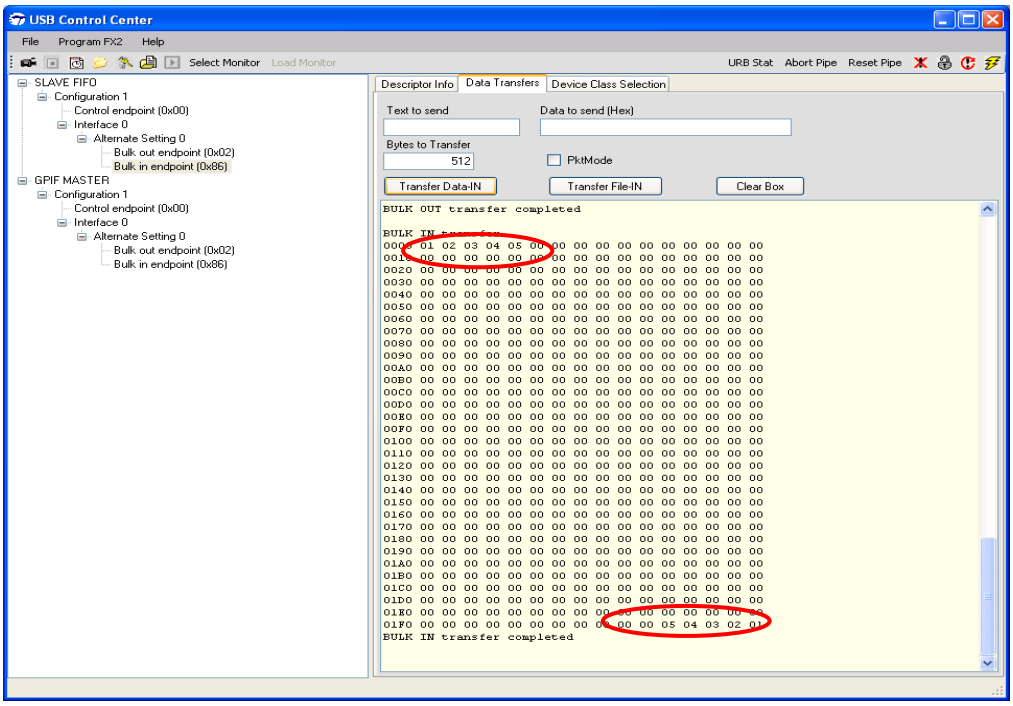
Use Control Centre to send 512 bytes into EP2 of the master FX2LP.

Figure 4. GPIF Master OUT Data Transfer



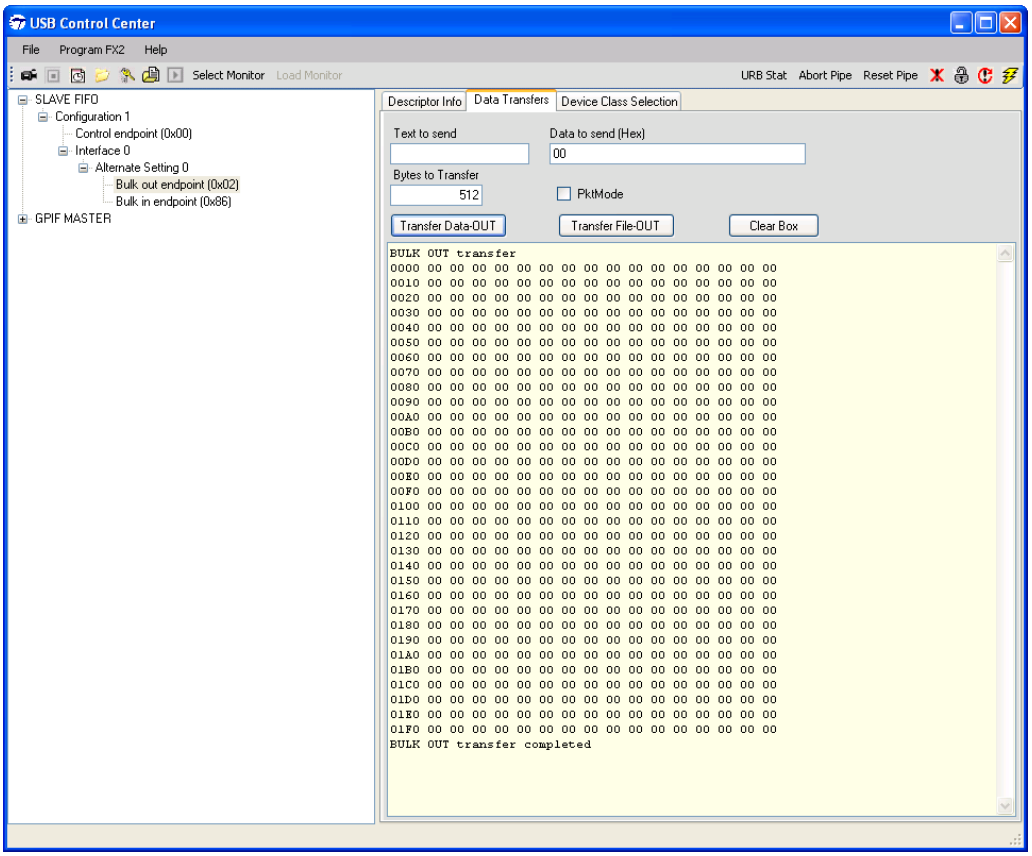
7. Issue BULK IN transfer of 512 bytes from EP6 of the slave FX2LP. The data received should be the same as that sent to EP2 OUT of the master FX2LP, except for the first and the last five bytes modified. The master modifies the first five bytes and the slave modifies the last five bytes.

Figure 5. Slave FIFO IN Transfer



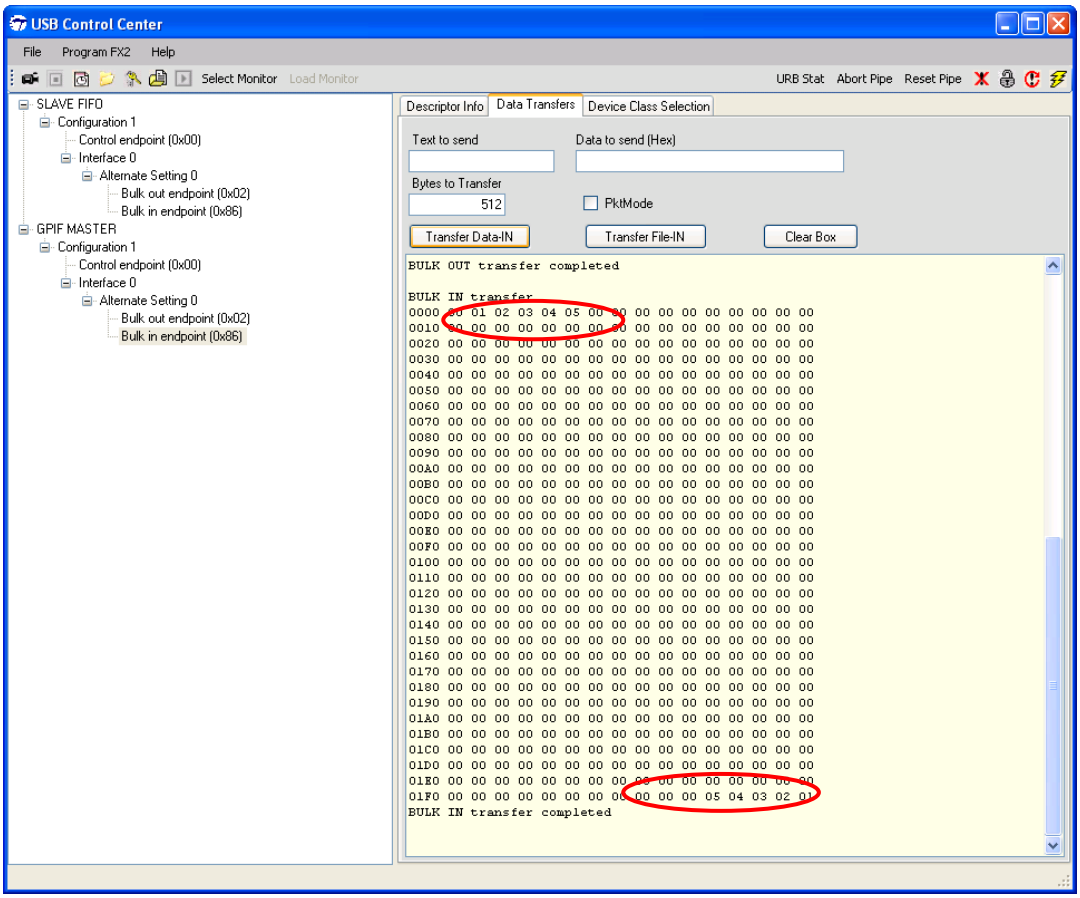
8. Now, send 512 bytes of data from EP2 of the slave FIFO.

Figure 6. Slave FIFO OUT Transfer



9. Issue Bulk IN transfer of 512 bytes from EP6 of the master FX2LP. In the read data, the first five bytes and last five byes are modified. The master modifies the first five bytes and the slave modifies the last five bytes.

Figure 7. GPIF Master IN Transfer



Auto Mode Operation of FX2LP in GPIF and Slave FIFO Configuration

Two FX2LP chips are interfaced with each other; one of the chips functions in GPIF auto mode and the other functions in slave FIFO auto mode.

Firmware

To view the code written for the master, open *master.uv2* from “FX2LP Back To Back\Firmware\Auto Mode\master”. To view the code for the slave FX2LP, open *Slave.uv2* from “FX2LP Back To Back\Firmware\ Auto Mode\slave”.

Initialization for FX2LP in GPIF Master Mode

TD_Init() takes care of the entire initialization with the following steps:

1. Configure EP2 and EP6 as double-buffered endpoints with buffer size equal to 512.
2. Reset the FIFOs of the endpoints EP2 and EP6.
3. Configure endpoints EP2 and EP6 FIFOs to 8-bit auto mode.

GPIF Auto OUT Mode

TD_Poll() of GPIF master performs the data loopback from EP2 OUT of the GPIF master to EP6 IN of the slave FIFO. The data is transferred from the master FX2LP to the slave using the FifoWr waveforms of GPIF.

```
  if( GPIFTRIG & 0x80 )              // if GPIF interface IDLE
  {
      if ( ! ( EP24FIFOFLGS & EP2EMPTY ) )  // if there's a packet in the peripheral domain
for EP2
      {
         PERIPH_FIFOADR0 = 0;              // FIFOADR[1:0]=10 - point to peripheral EP6
         PERIPH_FIFOADR1 = 1;
         SYNCDELAY;                        // used here as "delay"

         if ( SLAVENOTFULL )           // if the slave is not full
        {
            if(enum_high_speed)
            {
               SYNCDELAY;
               GPIFTCB1 = 0x02;            // setup transaction count
               SYNCDELAY;
                 GPIFTCB0 = 0x00;
               SYNCDELAY;
            }
            else
            {
               SYNCDELAY;
               GPIFTCB1 = 0x00;            // setup transaction count
               SYNCDELAY;
               GPIFTCB0 = 0x40;
               SYNCDELAY;
            }
            SYNCDELAY;
             GPIFTRIG = GPIFTRIGWR | GPIF_EP2;  // launch GPIF FIFO WRITE Transaction from
EP2 FIFO
             SYNCDELAY;
             while( !( GPIFTRIG & 0x80 ) )      // poll GPIFTRIG.7 GPIF Done bit
            {
               ;
            }
            SYNCDELAY;
        }
      }
  }
```

As soon as the packet comes into EP2 OUT of the master FX2LP, it is auto-committed to the peripheral domain. Thus, inside the TD_POLL() you only have to trigger the GPIF Write transaction whenever there is any data in the EP2 OUT endpoint of the master FX2LP.

GPIF Auto IN Mode

```
  if ( GPIFTRIG & 0x80 )                 // if GPIF interface IDLE
  {
      PERIPH_FIFOADR0 = 0;
      PERIPH_FIFOADR1 = 0;              // FIFOADR[1:0]=00 - point to peripheral EP2
      SYNCDELAY;
      if ( SLAVENOTEMPTY )               // if slave is not empty
      {
         if ( !( EP68FIFOFLGS & EP6FULL ) )     // if EP6 FIFO is not full
         {
            if(enum_high_speed)
           {
              SYNCDELAY;
```

```
                GPIFTCB1 = 0x02;                // setup transaction count
                SYNCDELAY;
                GPIFTCB0 = 0x00;
                SYNCDELAY;

            }
            else
            {
               SYNCDELAY;
               GPIFTCB1 = 0x00;                // setup transaction count
                SYNCDELAY;
               GPIFTCB0 = 0x40;
               SYNCDELAY;
            }
             GPIFTRIG = GPIFTRIGRD | GPIF_EP6; // launch GPIF FIFO READ Transaction to EP6
FIFO
             SYNCDELAY;
             while( !( GPIFTRIG & 0x80 ) )     // poll GPIFTRIG.7 GPIF Done bit
             {
               ;
             }
             SYNCDELAY;
        }
    }
 }
```

The EP6 IN endpoints of both the FX2LPs are in auto mode and the AUTO IN packet length is set to 512. Therefore, if the data transferred is an integral multiple of 512 (bytes), it is auto-committed from the peripheral to the host side and available on the host. This firmware can transfer data only when it is in integral multiples of 512 (bytes).

Slave Auto OUT and IN Mode

Because the packets are auto-committed, there is no need for any code in the slave FIFO apart from the initialization. Thus there is no data-handling code inside the TD_POLL() function for the slave FX2LP.

Testing the Project (Auto Mode)

Testing the project is similar to that of the manual mode. See Figure 1 for the connection diagram. There is no need for the two additional connections between PC0 and PC1 in auto mode; this is required only for handshaking in manual mode configuration.

The code to be downloaded to the master FX2LP and the slave FX2LP is:

- Master FX2LP: *master.iic* included in the attachment and available in "\FX2LP Back To Back\Firmware\ Auto Mode\master".
- Slave FX2LP: *slave.ii* included in the attachment along with this code example and available in "\FX2LP Back To Back\Firmware\Auto Mode\slave"
- The only difference observed with auto mode compared to manual mode is that the data received from the EP6 IN endpoint of the slave (master) FX2LP will be the same as the data that was sent to EP2 OUT of the master (slave) FX2LP.

Additional Resources

- EZ-USB FX2LP ™ GPIF Design Guide – AN66806
- Getting Started with FX2LP™
- Interfacing SRAM with FX2LP over GPIF

About the Author

Name: Gayathri Vasudevan

Title: Applications Engineer

Contact: gaya@cypress.com

Document History

Document Title: AN63787 - EZ-USB® FX2LP™ GPIF and Slave FIFO Configuration Examples Using 8-bit Asynchronous Interface

Document Number: 001-63787

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3016455	CPPK	08/26/2010	New example project.
*A	3172695	CPPK	02/14/2011	Removed all references to FX2 and CY3682 DVK. Template updates.
*B	3383901	GAYA	09/28/2011	Document title changed. Added new section explaining annual mode operation.
*C	3520175	GAYA	02/14/2012	Converted code example to application note
*D	3664694	GAYA	07/03/2012	Fixed the code comments. Fixed VID/PID issue.
*E	3720173	GAYA	08/22/2012	Modified the document title.

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
Optical Navigation Sensors	cypress.com/go/ons
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