Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.



THIS SPEC IS OBSOLETE

Spec No: 001-63620

Spec Title: CONFIGURING A XILINX SPARTAN-3E FPGA OVER USB USING EZ-USB FX2LP(TM) - AN63620

Replaced by: 001-61345

Configuring a Xilinx Spartan-3E FPGA Over USB Using EZ-USB FX2LP™

AN63620

Author: Prajith Cheerakkoda
Associated Project: Yes
Associated Part Family: CY7C68013A/CY7C68014A/CY7C68015A/CY7V68016A
Software Version: None
Associated Application Notes: AN61345, AN6077

OBSOLETE

Application Note Abstract

This application note demonstrates a technique for dynamically configuring a Xilinx Spartan-3E Field Programmable Gate Array (FPGA) over USB using EZ-USB FX2LP™, a high-speed USB peripheral controller. After the FPGA is configured, FX2LP can act as a high-speed data path between the USB host and the FPGA. This capability of FX2LP enhances FPGA-based USB applications such as logical analyzers, oscilloscopes, image processing, and high-speed data acquisition.

Introduction

FX2LP is an excellent solution for adding high-speed USB functionality to FPGA-based solutions. AN61345 demonstrates the implementation of a high-speed interface between FX2LP and a Xilinx Spartan-3E FPGA with FX2LP acting in Slave FIFO mode and FPGA acting as the master to it.

For an advanced FPGA-based application that requires high-speed USB connectivity, configuring the FPGA over USB using FX2LP eliminates the need for a dedicated configuration chip (for example, a PROM or a processor) for the FPGA. This method can also act as a replacement for the popular JTAG configuration interface that requires JTAG connectors on the board. Usage of this method reduces cost and board space.

FX2LP can be interfaced to an external system in two modes of operation: GPIF (General Programmable Interface) mode and Slave FIFO mode.

- FX2LP acts as a master to the external system and generates all the necessary control signals to read and write data. This mode is known as General Programmable Interface (GPIF).
- When the external system is intelligent enough to generate the necessary read and write control signals, it can act as the master of the interface and FX2LP can act as the slave device. Here, we configure FX2LP in the Slave FIFO mode.

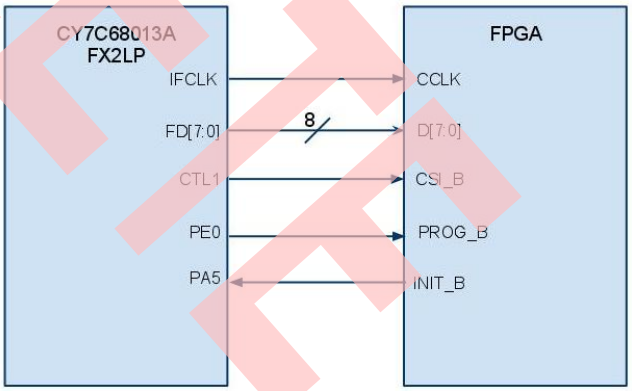
This application note demonstrates the usage of GPIF mode to load the configuration bitstream into FPGA. After the configuration is performed successfully using GPIF, the operation mode of FX2LP is switched to Slave FIFO so that the FPGA can act as a master for the data transfer phase.

The scripting capability of the Cypress generic driver CyUSB.sys can be used for automatically downloading firmware to FX2LP after it is plugged into the host machine. This eliminates the need to store FX2LP firmware on a large EEPROM, so that a smaller, less expensive EEPROM which stores only the device VID/PID can be used.

Hardware Connections

There are various options for configuring a Xilinx® Spartan®-3E FPGA. In this application note, the configuration mode implemented using GPIF is the Non-Continuous SelectMAP mode. The following diagram illustrates the hardware connections required for programming the FPGA using GPIF.

Figure 1. Hardware Connections Diagram



The following table defines the interface between FX2LP and FPGA for the implementation of the SelectMAP configuration mode.

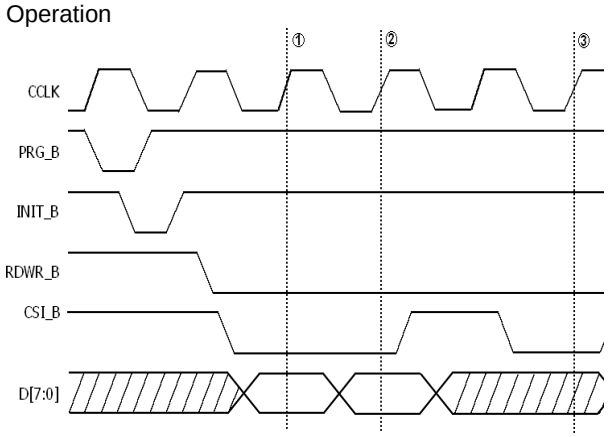
Table 1. Assignment of FX2LP Pins to FPGA Configuration Signals

FX2LP Pin	Xilinx Spartan-3E FPGA signal	Description
IFCLK	CCLK	IFCLK is connected to configuration clock (CCLK). The IFCLK output can be configured to either 30 MHz or 48 MHz.
CTL1	CSI_B	When CSI_B is asserted, the FPGA samples the configuration data on each rising CCLK edge.
PE3	RDWR_B	The state RDWR_B pin decides whether the FPGA bus is being read or written into. When RDWR_B is asserted low, a write operation is to be performed to the FPGA. When it is high, this indicates a read operation. Because you are writing into the FPGA, you can permanently keep this pin low to indicate a data write condition.
FD[7:0]	D[7:0]	Byte-wide configuration data bus
PE4	PROG_B	Drive PROG_B low and release to reprogram the FPGA. Holding the PROG_B pin low clears the configuration memory of the FPGA.
PA5	INIT_B	After the PROG_B signal is de-asserted by FX2LP, it should wait until the FPGA asserts the INIT_B signal high. This indicates that the FPGA is ready to start the configuration process. If the FPGA drives the INIT_B signal low at the end of the configuration, it indicates a configuration error.
PE1	DONE	This pin is low during configuration and asserted high by the FPGA once the configuration is completed successfully.

The 56-pin package can be chosen for this application because the size of the included firmware does not exceed the size of the on-chip RAM.

The following diagram demonstrates the timing diagram for the slave SelectMAP method of configuration. At instants 1, 2, and 3, a byte of configuration data each is clocked into the FPGA. The configuration timing specifications for each FPGA family can be found in the respective family datasheet.

Figure 2. Timing Diagram for Slave SelectMAP Mode of Operation

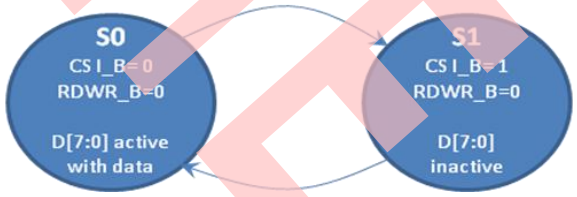


Usage of GPIF Designer to Create the Configuration Waveform

GPIF Designer is a graphical user interface from Cypress to help developers in the creation of GPIF waveforms. Familiarity with the General Programmable Interface chapter of the EZ-USB Technical Reference Manual is necessary. Extensive documentation of the general GPIF functionality and waveform programming is also available in the Help menu of the GPIF Designer tool. After the block diagram is specified and waveforms are correctly configured, the GPIF Designer is ready to generate the GPIF waveform descriptors. The utility performs the generation of the data structures when the user selects the *Export to GPIF.c* function in the **Tools** menu.

The state diagram for the configuration waveform is illustrated in the following figure. In s0, CSI_B is asserted and a byte of configuration data is placed on the data bus. In state S1, CSI_B is de-asserted and the data bus stays inactive.

Figure 3. State Diagram for Waveform Creation



The block diagram should be modified to indicate that CTL1 is the only control line to be used in the design and it is named as CS. There are no ready lines used for this design. After the waveform is triggered, the waveform terminates only when the transaction count (TC) expires. GPIF checks the transaction count each time the waveform passes through the idle state.

The following figure shows the block diagram view of the design in GPIF Designer. The IFCLK is inverted so that the setup time requirement for the FPGA configuration data is easily met.

Figure 4. GPIF Designer Block Diagram View

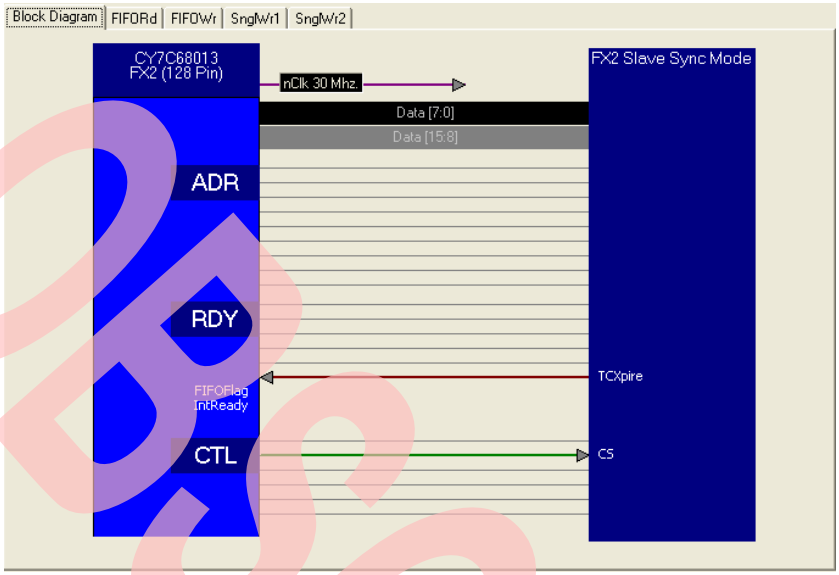
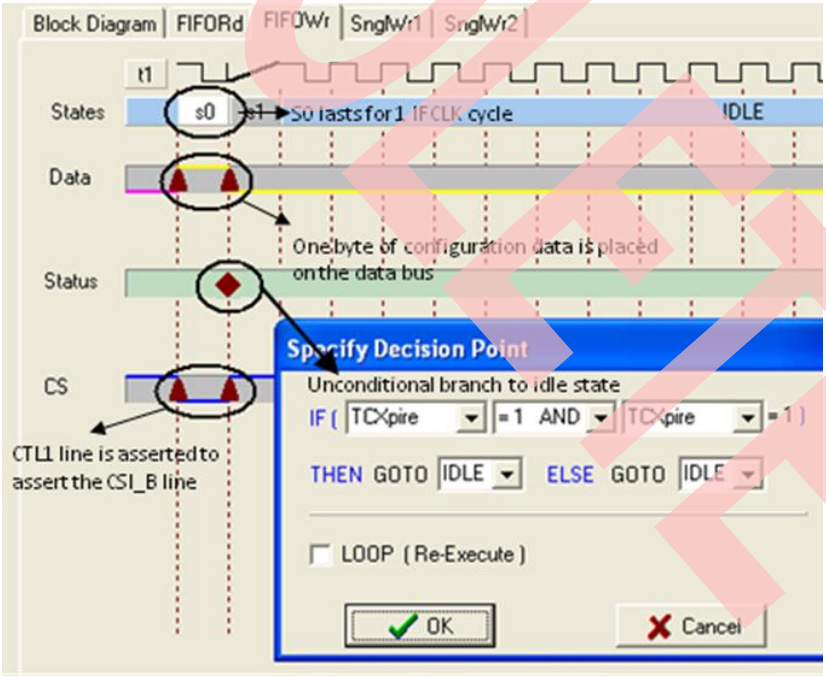


Figure 5. FIFO Write Waveform

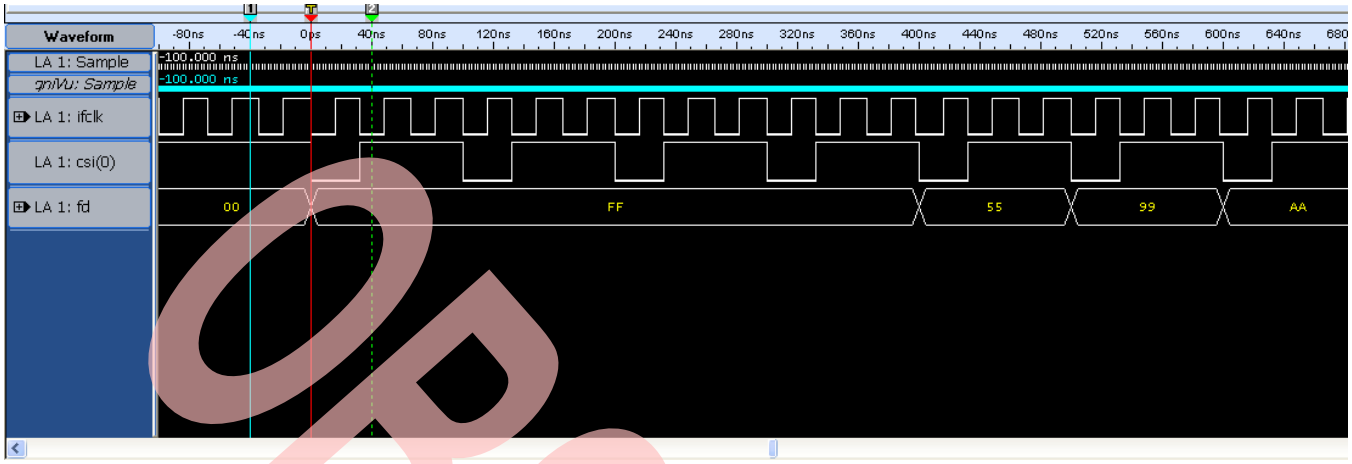


The decision point at S1 unconditionally branches the state-machine to idle state. The waveform is terminated only when the transaction count expires.

OBSOLETE

The following figure shows a logic analyzer trace of the generated gpif waveform.

Figure 6. GPIF Waveform



Exporting GPIF Waveforms

To export the waveforms to a C file and include it in the firmware project, follow these steps:

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

Firmware Implementation

Cypress provides a Firmware FrameWorks that implements 8051 code for EZ-USB chip initialization, USB standard device request handling, and USB suspend power management services for the user. The user only has to provide a USB descriptor table, and code to implement the peripheral function. The FrameWorks provide function hooks and example code to help with this process. The FrameWorks as well as the basic code examples are included in the FX2LP Development Kit CY3684. You can download the kit contents here. You can also find a number of code examples after installing the CY3684 DVK in location C:\Cypress\USB\Examples\FX2LP. The Firmware FrameWorks files can be found in location C:\Cypress\USB\Target\Fw\LP. The CY3684 DVK User guide contains an overview of the Firmware FrameWorks and how to go about using the same for developing application firmware. The user is expected to go through the same for ease of understanding this section.

The TD_Init() section configures the endpoint FIFOs for byte-wide auto mode operation. In auto mode of operation, all the packets received by the FIFO from the host are automatically committed to the peripheral domain without any CPU intervention. AN6077 explains in detail about the auto and manual modes of operation of the EZ-USB FIFOs.

```
void TD_Init(void)             // Called once at startup
{
  CPUCS = 0x10;                // CLKSPD[1:0]=10, for 48MHz operation
  SYNCDELAY;                   // CLKOE=0, don't drive CLKOUT

  GpifInit( );                 // init GPIF engine via GPIFTool output file

  SYNCDELAY;
  EP2CFG = 0xA0;      // EP2OUT, bulk, size 512, 4x buffered
  SYNCDELAY;
  EP6CFG = 0xE0;      // EP6IN, bulk, size 512, 4x buffered
  SYNCDELAY;

  FIFORESET = 0x80;   // set NAKALL bit to NAK all transfers from host
  SYNCDELAY;
  FIFORESET = 0x02;   // reset EP2 FIFO
  SYNCDELAY;
  FIFORESET = 0x06;   // reset EP6 FIFO
  SYNCDELAY;
```

OBSOLETE

```
  FIFORESET = 0x00;  // clear NAKALL bit to resume normal operation
  SYNCDELAY;

  EP2FIFOCFG = 0x00; // allow core to see zero to one transition of auto out bit
  SYNCDELAY;
  EP2FIFOCFG = 0x10; // auto out mode, disable PKTEND zero length send, byte operation
  SYNCDELAY;
  EP6FIFOCFG = 0x08; // auto in mode, disable PKTEND zero length send, byte operation
  SYNCDELAY;

  EP2GPIFFLGSEL = 0x01; // For EP2OUT, GPIF uses emtpy flag
  SYNCDELAY;

  PORTECFG = 0x00;          //Do not enable any of the alternate configurations of PORTE
  PORTACFG = 0x00;          //Do not enable any of the alternate configurations of PORTA
  OEA = 0xC0;               //PORTA_5(INIT) is set as input
  OEE |= 0x18;    //PORTE_1(DONE) is input, PORTE_4(PROG_B) and PORTE_3(RDWR_B) are outputs

  IOE = 0xFE;               // initialize PORTE_4 and PORTE_3 as high

  }
```

The configuration process is triggered when FX2LP receives a vendor command 0xB2 from the host. After the vendor command is received successfully, the configuration process starts after the host sends the configuration bitstream data to endpoint FIFO 2. The vendor command handler for command 0xB2 is as follows:

```
case StartConfig:                   //vendor command 0xb2
    {


        EP0CS |= bmHSNAK;               // Acknowledge handshake phase of device request
        GpifInit( );
        FIFORESET = 0x80;  // set NAKALL bit to NAK all transfers from host
        SYNCDELAY;
        FIFORESET = 0x02;  // reset EP2 FIFO
        SYNCDELAY;
        FIFORESET = 0x06;  // reset EP6 FIFO
        SYNCDELAY;
        FIFORESET = 0x00;  // clear NAKALL bit to resume normal operation
        SYNCDELAY;

        EP2FIFOCFG = 0x00; // allow core to see zero to one transition of auto out bit
        SYNCDELAY;
        EP2FIFOCFG = 0x10; // auto out mode, byte-wide
        SYNCDELAY;
        EP6FIFOCFG = 0x08; // auto in mode, byte-wide
        SYNCDELAY;
        prg_enable = TRUE;
        IOE = IOE & 0xEF;             //PROG_B signal low

        EZUSB_Delay1ms();             //PROG_B signal kept asserted for 1 ms
        IOE = IOE | 0x10;             //PROG_B signal high
        SYNCDELAY;
```

OBSOLETE

```
        IOE = IOE & 0xF7;              //Assert RDWR_B low
        SYNCDELAY;



         break;
    }
```

The GPIF waveform is triggered by writing into the GPIFTRIG register in TD_Poll(). The transaction count register is written with the length of the configuration bitstream.

```
void TD_Poll(void)
{
  // Handle OUT data...
if(prg_enable)
{
  if( GPIFTRIG & 0x80 )                // if GPIF interface IDLE
  {
    if ( ! ( EP24FIFOFLGS & 0x02 ) )  // if there's a packet in the peripheral domain for EP2
    {
     if ( INIT )                  // if the FPGA is ready
     {
                       SYNCDELAY;
                       GPIFTCB2 = 0x04;
                      SYNCDELAY;
                       GPIFTCB1 = 0x54;            //setup transaction count. The bitstream
                       SYNCDELAY;                  //size is constant for each family of FPGA.
                       GPIFTCB0 = 0x8A;            //this can be changed by host using a
                                                   //vendor command

        SYNCDELAY;
        GPIFTRIG = GPIFTRIGWR | GPIF_EP2;  //launch GPIF FIFO WRITE Transaction from EP2 FIFO
        SYNCDELAY;

           while( !( GPIFTRIG & 0x80 ) )      // poll GPIFTRIG.7 GPIF Done bit
        {
          ;
        }
        SYNCDELAY;
       prg_enable= FALSE;     //end of configuration
      }
    }
   }
  }
}
```

After the successful configuration, a vendor command 0xB3 is sent from the host to switch the mode of operation from GPIF to Slave FIFO. The vendor command handler for command 0xB3 is as follows:

```
case StartXfer:
{
  EP0CS |= bmHSNAK;
  prg_enable = FALSE;
  PINFLAGSAB = 0x08;                    // FLAGA - EP6FF
  SYNCDELAY;
  PINFLAGSCD = 0xE0;                    // FLAGD - EP2EF
  SYNCDELAY;
  PORTACFG |= 0x80;
  IFCONFIG = 0xE3;


  SYNCDELAY;
```

OBSOLETE

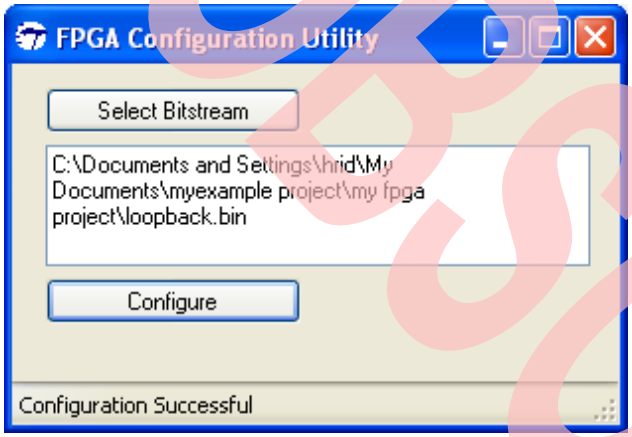
```
  FIFORESET = 0x80;             // activate NAK-ALL to avoid race conditions
  SYNCDELAY;                    // see TRM section 15.14
  FIFORESET = 0x02;             // reset, FIFO 2
  SYNCDELAY;                    //
  FIFORESET = 0x04;             // reset, FIFO 4
  SYNCDELAY;                    //
  FIFORESET = 0x06;             // reset, FIFO 6
  SYNCDELAY;                    //
  FIFORESET = 0x08;             // reset, FIFO 8
  SYNCDELAY;                    //
  FIFORESET = 0x00;             // deactivate NAK-ALL

  // handle the case where we were already in AUTO mode...
  // ...for example: back to back firmware downloads...
  SYNCDELAY;                    //
  EP2FIFOCFG = 0x00;            // AUTOOUT=0, WORDWIDE=0

  // core needs to see AUTOOUT=0 to AUTOOUT=1 switch to arm endp's

  SYNCDELAY;                    //
  EP2FIFOCFG = 0x10;            // AUTOOUT=1, WORDWIDE=0

  SYNCDELAY;                    //
  EP6FIFOCFG = 0x4C;            // AUTOIN=1, ZEROLENIN=1, WORDWIDE=0

  SYNCDELAY;
  break;
}
```

OBSOLETE

FPGA Configuration Utility

An example host application, FPGA Configuration Utility, created for configuring the FPGA, is included in the design. The application was developed in Visual C# 2008 Express Edition using the Cypress Application Development Library CyUSB.Net DLL which is included in SuiteUSB 3.4. The device has to be bound to CyUSB.sys, a general purpose driver developed by Cypress. This application helps as a reference for developing your own host applications for FPGA configuration. This provides the flexibility to select the bitstream file for configuration. The following figure displays the FPGA Configuration Utility.

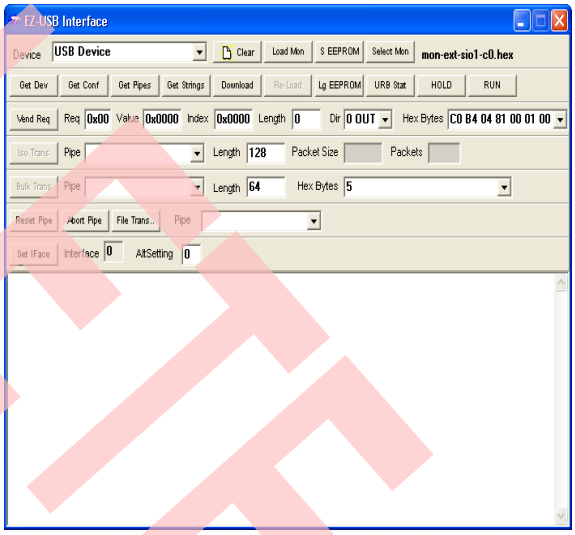
Figure 7. FPGA Configuration Utility



Click **Select Bitstream** to select the required configuration file and click **Configure** to perform the configuration process. 'Configuration Successful' message displays after the completion of the process.

Testing the Project

- You can either download firmware to FX2LP using a host application or write a script file to download the firmware automatically when the device is plugged in. AN50963 explains the implementation of scripting for automatic firmware download.
- For testing purposes, you can follow these steps to manually download the firmware to FX2LP using CyConsole, a development host application included in the SuiteUSB3.4 SDK.
  - Connect the board to the PC. It enumerates with the default internal descriptors.
  - Use the CyUSB.inf file inside the Drivers folder to bind with the device. For help on binding the driver, refer to *MatchingDriverToUSBDevice.htm* inside the Drivers folder. AN61465 contains detailed explanations on the generation of an .inf file.
  - Start the CyConsole host application. In the **Options** menu, select **EZ-USB Interface**. The following window is displayed



  - Click the **Download** button and navigate to *master.hex* file in the associate project folder. Wait for the firmware to re-enumerate and reconnect as a Cypress EZ-USB device.
  - A window is displayed prompting you to bind the driver. Use the appropriate *CyUSB.inf* file inside the Drivers folder to bind.
  - Now, you can open the FPGA Configuration Utility from the associated project folder.
- After the firmware download and enumeration of the device with VID/PID 0x04B4/1002, open the FPGA configuration utility. Click **Select Bitstream** to select the bitstream file loopback.bin that is included in the project source folder. This is the byte-swapped

OBSOLETE

bitstream file generated by the iMPACT GUI included in Xilinx ISE. *Loopback.bin* corresponds to the Data Loopback example explained in AN61345. The example implements a loopback on endpoints EP2OUT and EP6IN. Data sent from host to EP2 is read by the FPGA, increment and written to the EP6 endpoint FIFO.

- Click **Configure** to perform the configuration process. 'Configuration Successful' message displays after the completion of the process.
- Verify the loopback operation is being performed successfully using the Cypress CyConsole application included in SuiteUSB 3.4.2.

Summary

This application note implements a method for configuring an FPGA over USB using the GPIF controller of FX2LP. For an FPGA design that needs high-speed USB functionality, usage of FX2LP for configuring the FPGA with high-speed USB connectivity helps in reducing component cost and board space.

About the Author

Name: Prajith Cheerakkoda

Title: Applications Engineer.

OBSOLETE

Document History

Document Title: Configuring a Xilinx Spartan-3E FPGA Over USB Using EZ-USB FX2LP™

Document Number: 001-63620

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3008686	HRID	08/17/2010	New application note
*A	3164512	HRID	03/04/2011	Modified title, abstract, and introduction, added more description to section on testing the product, included description of Firmware FrameWorks.
*B	4118977	PRJI	10/9/2013	Completing sunset review
*C	5142682	GAYA	02/18/2016	Obsoleting the document.

OBSOLETE

EZ-USB FX2LP is a trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2010–2016. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.