AN84868

Configuring a Xilinx FPGA Over USB Using Cypress EZ-USB FX3

Author: Rama Sai Krishna Vakkantula

Associated Project: Yes

Associated Part Family: CYUSB3014

For latest FX3 SDK: click here

More code examples? We heard you.

To access a variety of FX3 code examples, please visit our USB SuperSpeed Code Examples webpage.

AN84868 shows you how to configure a Xilinx® FPGA over a slave serial interface using EZ-USB® FX3™, which is the next-generation USB 3.0 peripheral controller. This interface lets you download configuration files into a Xilinx FPGA over USB 2.0 or 3.0. The firmware files with this application note are designed and tested for Xilinx FPGAs, but you can customize them for other FGPAs with a similar interface.

Contents

1 Introduction

FX3 has a configurable, parallel General Programmable Interface (GPIF II) that can connect to external devices like image sensors, external processors, ASICs, or FPGAs. As a result, users can integrate USB 3.0 capability into almost any system.

In addition, FX3 provides interfaces to connect to serial peripherals such as UART, SPI, I^2C, and I^2S.

FX3 allows you to add SuperSpeed capability to any FPGA-based design. In most applications, FPGA acts as a master and the GPIF II operates in a synchronous Slave FIFO interface. For more details on the Slave FIFO interface, see AN65974 – Designing with the EZ-USB® FX3™ Slave FIFO Interface.

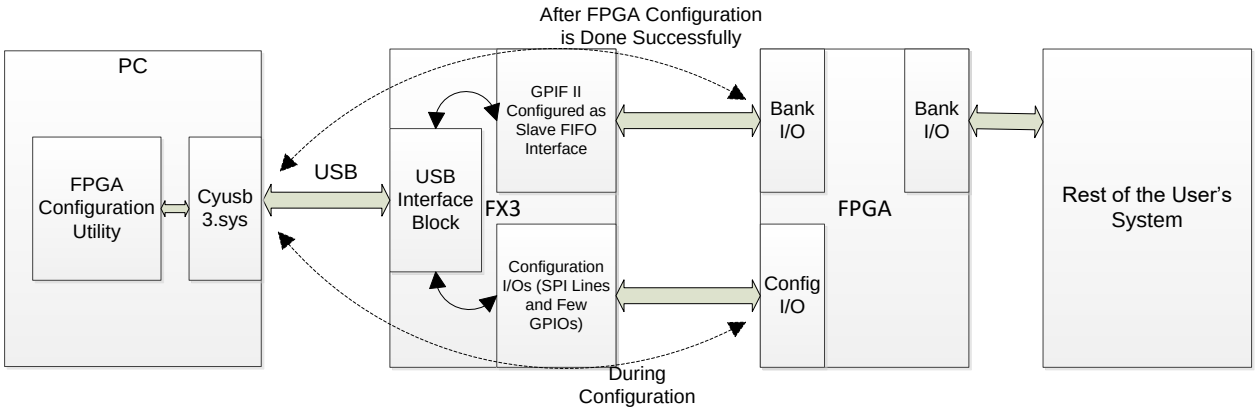
You can configure the FPGA using the controller (in this case, FX3) connected to it. Using FX3 eliminates the need for a dedicated configuration chip (for example, a PROM or a processor) for the FPGA. In addition, this method can act as a replacement for the popular JTAG configuration interface that requires JTAG connectors on the board. This method reduces the cost and board space. The FPGA configuration can also be loaded using FX3 from an external SPI flash or EEPROM, which is preloaded with the FPGA configuration file. See FX3 + FPGA + HelionVision ISP-Based Industrial Camera Reference Design – KBA222700 where FX3 is interfaced with a FPGA for imaging application.

Acting as a master, FX3 can configure the Xilinx FPGA in two modes: Slave Parallel (SelectMAP) and Slave Serial. See the Xilinx Spartan-6 FPGA Configuration User Guide to get information on the various options to configure an FPGA. This application note describes only the Slave Serial mode. It also describes how FX3 firmware switches to the Slave FIFO interface after the FPGA configuration is complete. Figure 1 shows a block diagram in which FX3 configures the FPGA at the start and then switches to the Slave FIFO interface after the configuration is successful.

This application note uses a host application to load the FPGA configuration file using vendor commands. The vendor commands will start the FPGA configuration process through SPI interface with the FPGA configuration data received from the host.

The following sections examine the details of the Xilinx Slave Serial configuration interface and its design implementation using FX3.

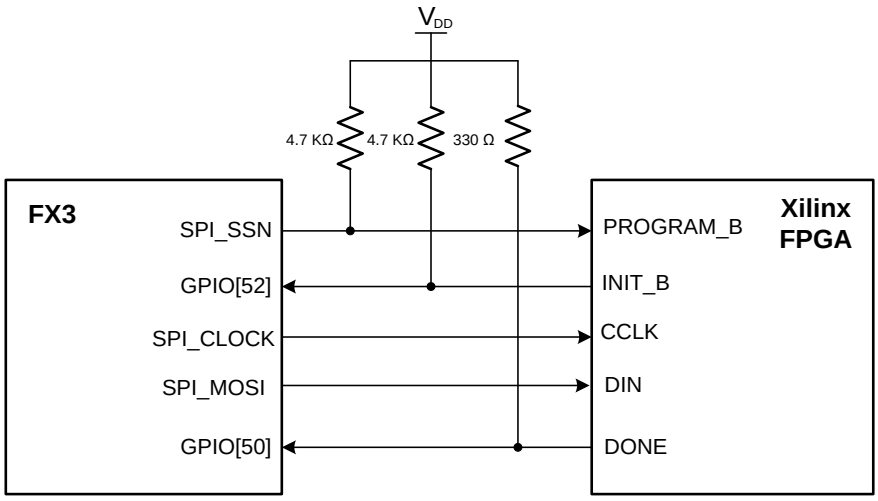
Figure 1. System-Level Application Block Diagram



2 Xilinx Slave Serial Configuration Interface

This section describes the details of Xilinx Slave Serial interface. Figure 2 shows the interface pins associated with the Xilinx Slave Serial interface and Table 2 contains the description of Slave Serial interface pins.

Figure 2. Hardware Connections Between FX3 and Xilinx Spartan-6 FPGA



PROGRAM_B, INIT_B, and DONE are open-drain signals. Connect pull-up resistors of suitable value on these lines. The resistor values mentioned in Figure 2 are taken from the Xilinx Spartan-6 FPGA Configuration User Guide. Note that there is no need to connect these pull-up resistors if the FX3 CYUSB3KIT-001 board is used, but pull-up resistors should be placed in the final design.

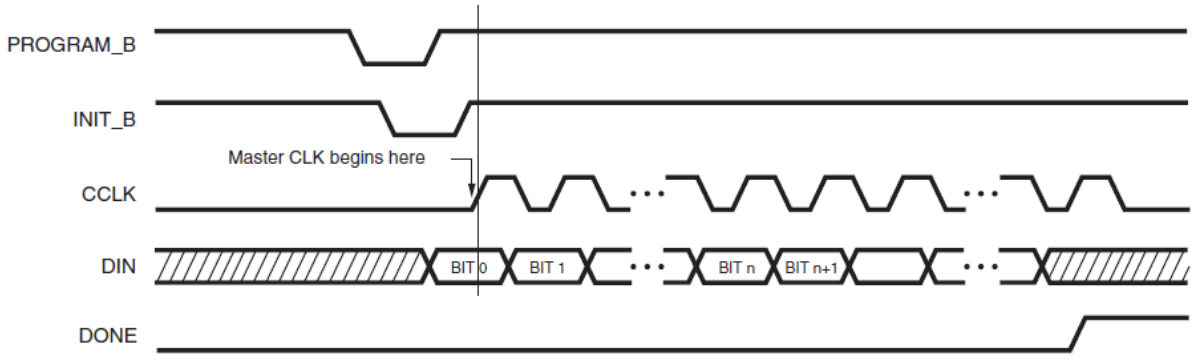
Table 1 shows the interface signals in the Slave Serial interface.

Table 1. Xilinx Slave Serial Configuration Pin Description

Pin Name	Pin Direction (to FPGA)	Pin Description
PROGRAM_B	Input	**Program FPGA.** Active LOW. When asserted LOW for 500 ns or longer (300 ns in the Spartan-3 FPGAs), it forces the FPGA to restart its configuration process by clearing configuration memory.
INIT_B	Open drain bidirectional I/O	**FPGA Initialization Indicator.** Drives LOW after power-on reset (POR) or when PROGRAM_B pulses LOW while the FPGA is clearing its configuration memory. If a CRC error is detected during configuration, FPGA again drives INIT_B LOW.
CCLK	Input	**Configuration Clock.**
DIN	Input	**Data Input.** Serial data. FPGA captures data on rising CCLK edge.
DONE	Open drain bidirectional I/O	**FPGA Configuration Done.** LOW during configuration. Goes HIGH when FPGA successfully completes configuration.

Figure 3 shows the clocking sequence diagram of the Xilinx Slave Serial configuration.

Figure 3. Xilinx Slave Serial Configuration Clocking Sequence



3 Implementation

FX3 starts the configuration by pulsing PROGRAM_B and monitoring the INIT_B pin. When the INIT_B pin goes HIGH, the FPGA is ready to receive data. The FX3 then starts supplying data and clock signals until either the DONE pin goes HIGH, indicating a successful configuration, or until the INIT_B pin goes LOW, indicating a configuration error. The configuration process requires more clock cycles than indicated from the configuration file size. These additional clocks are required during the FPGA's startup (see Figure 3).

3.1 Hardware Details

3.1.1 Hardware Boards

- Xilinx SP601 Evaluation Kit
- SuperSpeed Explorer Kit (CYUSB3KIT-003) or EZ-USB FX3 DVK (CYUSB3KIT-001)
- Samtec-to-FMC interconnection board (for CYUSB3KIT-001) or CYUSB3ACC005 Interconnection board (for CYUSB3KIT-003).
- Wires to interconnect configuration signals

The SPI hardware block in FX3 serializes the configuration data from the PC. The SPI_SSN (slave select), SPI_CLOCK, and SPI_MOSI of FX3 are connected to the PROGRAM_B, CCLK, and DIN of the Xilinx FPGA, respectively. The INIT_B and DONE pins of the FPGA are connected to GPIOs 52 and 50, respectively. Figure 2 shows the connections between FX3 and the Xilinx FPGA.

3.2 FX3 Firmware

The attached firmware has the following parts:

- Configuration of the Xilinx FPGA connected to FX3 over the Slave Serial interface.
- The Slave FIFO interface configuration works the same as described in AN65974 if the Xilinx FPGA configuration is successful.

The SPI hardware block in FX3 serializes the data that FX3 receives from the PC application "FPGA Configuration Utility," as Figure 1 shows. The FPGA Configuration Utility is designed to identify the USB devices with Cypress VID (0x04B4) and PID (0x00F1).

Review the FX3 application structure chapter of the FX3 Programmer's Manual to learn the structure of application firmware. Use the FX3 Firmware API Guide as a reference for more details on the FX3 SDK APIs.

Table 2 describes the files present in the firmware source code, which is attached to this application note.

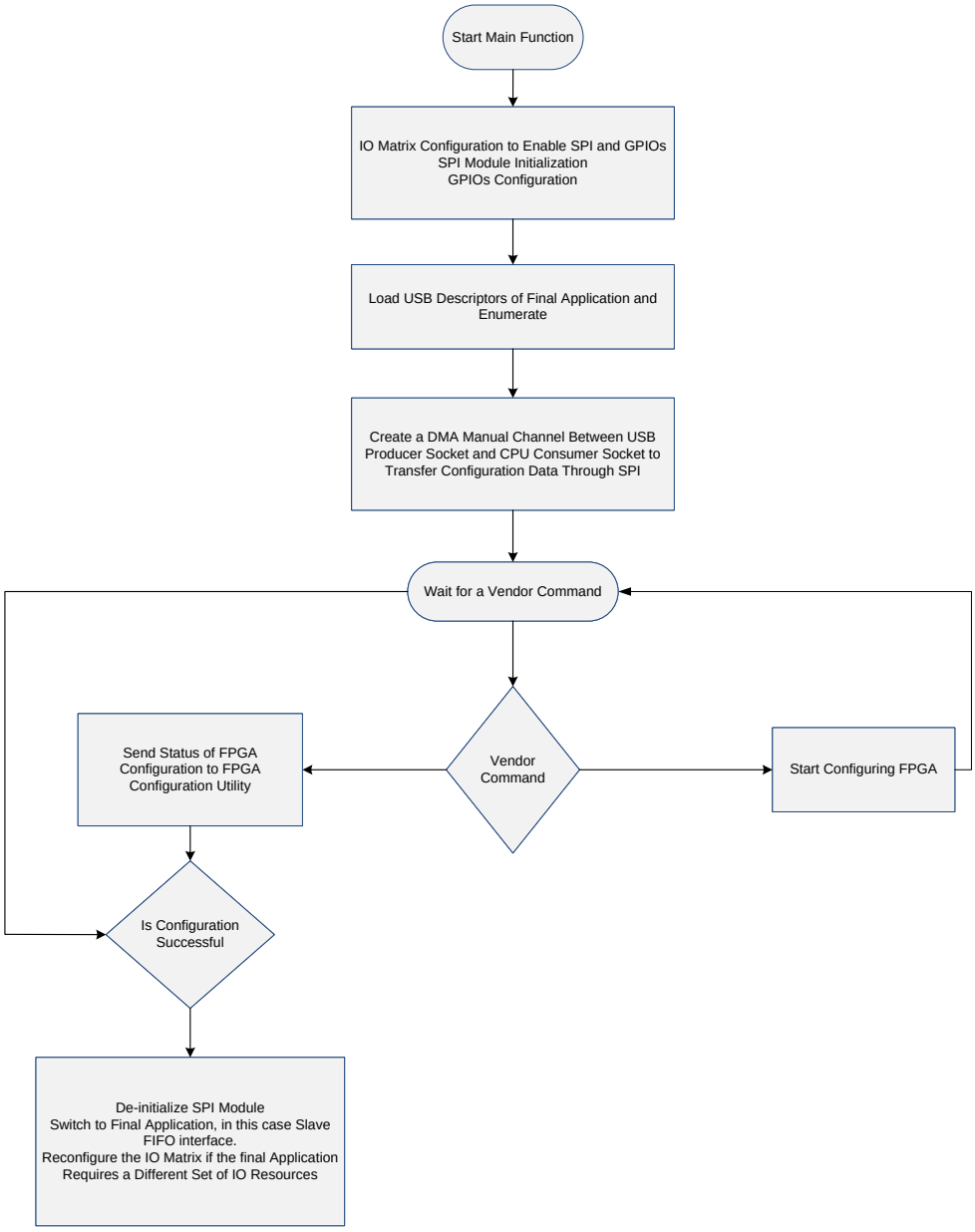
Table 2. Description of FX3 Firmware Source Files

File Name	Description
cyfx_gcc_startup.S	Cypress FX3 firmware startup code
cyfxconfigfpga.c	This file illustrates the configuration of FPGA in a Slave Serial mode example. It contains the following functions: • **`Main`**: Initializes the FX3 device, sets up caches, configures the FX3 I/Os, and starts the RTOS kernel. • **`CyFxConfigFpgaApplnInit`**: Initializes the FX3 GPIO and SPI modules. Configure GPIO[50] and GPIO[52] as input signals. Initializes the FX3 USB block for enumeration. • **`CyFxConfigFpgaApplnStart`**: Endpoint configuration for USB transfers and DMA channel configuration for data transfers from USB block to SPI block of FX3. • **`CyFxConfigFpgaApplnStop`**: De-initializes the FX3 GPIO and SPI modules to allow reconfiguration of the I/O matrix. • **`CyFxConfigFpga`**: Writes configuration data to the Xilinx FPGA over the Slave Serial interface.
cyfxconfigfpga.h	This file contains the constants and definitions used by the Configure FPGA application example.
cyfxslfifosync.c	This file illustrates the Slave FIFO Synchronous mode example. It contains the following functions: • **`CyFxApplicationDefine`**: Creates an application thread to perform data transfers over the Slave FIFO interface. • **`SlFifoAppThread_Entry`**: Application thread function that calls initialization functions for internal blocks of FX3. Waits for the events to configure FPGA and switches to the Slave FIFO interface once the FPGA configuration is successful. • **`CyFxSwitchtoslFifo`**: Reconfigures the FX3 I/O matrix per the Slave FIFO interface requirement. • **`CyFxSlFifoApplnInit`**: Initializes the processor interface block, loads the GPIF configuration for the Slave FIFO interface, and starts the GPIF state machine. • **`CyFxSlFifoApplnStart`**: Endpoint configuration for USB transfers and DMA channel configuration for data transfers between the USB block and the GPIF II block of FX3. • **`CyFxSlFifoApplnStop`**: This function stops the Slave FIFO application. This is called whenever a RESET or DISCONNECT event is received from the USB host. The endpoints are disabled, and the DMA channel is destroyed by this function. • **`CyFxSlFifoApplnUSBEventCB`**: Handles USB events such as suspend, cable disconnect, reset, and resume. • **`CyFxSlFifoApplnUSBSetupCB`**: Callback to handle the USB setup requests. • **`CyFxSlFifoApplnDebugInit`**: Initializes the FX3 UART block for printing debug messages. The debug prints are routed to the UART and can be seen using a UART console running at 115200 baud.
cyfxslfifosync.h	This file contains the constants and definitions used by the Slave FIFO application.
cyfxslfifousbdscr.c	This file contains the USB descriptors needed for the Slave FIFO example.
cyfxtx.c	This file defines the porting required for the ThreadX RTOS. It is provided in source form and must be compiled with the application source code.
cyfxgpif2config.h	This file contains the GPIF II descriptors for the 16-bit and 32-bit Slave FIFO interface.

Note: See the “FX3 Terminology” section in the Getting Started with EZ-USB FX3 application note to learn the terms specific to FX3.

The flow chart in Figure 4 describes the FX3 firmware.

Figure 4. FX3 Firmware Flow Chart



3.3 I/O Matrix Configuration

In the `main()` function, configure the I/O matrix (shown in the following code) according to the application requirement. The GPIF II interface is configured to 16-bit to enable the SPI interface. GPIOs 50 and 52 are enabled to connect with the DONE and INIT_B pins of the Xilinx FPGA (see Figure 2 for the hardware interface diagram). You can find this code snippet in the function `main()` present in the *cyfxconfigfpga.c* file.

```
io_cfg.useUart   = CyTrue;
io_cfg.useI2C    = CyFalse;
io_cfg.useI2S    = CyFalse;
```

```
io_cfg.useSpi    = CyTrue;
io_cfg.isDQ32Bit = CyFalse;
io_cfg.lppMode   = CY_U3P_IO_MATRIX_LPP_DEFAULT;
/* GPIOs 50 and 52 are enabled. */
io_cfg.gpioSimpleEn[0]  = 0x00000000;
io_cfg.gpioSimpleEn[1]  = 0x00140000;
io_cfg.gpioComplexEn[0] = 0;
io_cfg.gpioComplexEn[1] = 0;
status = CyU3PDeviceConfigureIOMatrix (&io_cfg);
```

3.3.1 SPI Module Initialization

The SPI module is initialized and configured by the following code. It is configured to run at a 25-MHz clock frequency. The FX3 SPI hardware block can support up to a 33-MHz clock frequency. You can find this code snippet in the function `CyFxConfigFpgaApplnInit()`present in the *cyfxconfigfpga.c* file.

```
/* Start the SPI module and configure the master. */
apiRetStatus = CyU3PSpiInit();

/* Start the SPI master block. Run the SPI clock at 25MHz and configure
   the word length to 8 bits. Also configure the slave select using FW. */
        CyU3PMemSet ((uint8_t *)&spiConfig, 0, sizeof(spiConfig));
        spiConfig.isLsbFirst = CyFalse;
        spiConfig.cpol       = CyTrue;
        spiConfig.ssnPol     = CyFalse;
        spiConfig.cpha       = CyTrue;
        spiConfig.leadTime   = CY_U3P_SPI_SSN_LAG_LEAD_HALF_CLK;
        spiConfig.lagTime    = CY_U3P_SPI_SSN_LAG_LEAD_HALF_CLK;
        spiConfig.ssnCtrl    = CY_U3P_SPI_SSN_CTRL_FW;
        spiConfig.clock      = 25000000; /* Maximum value of SPI clock is 33 MHz*/
        spiConfig.wordLen    = 8;

apiRetStatus = CyU3PSpiSetConfig (&spiConfig, NULL);
```

3.3.2 GPIO Configuration

The GPIO module is initialized and configured with the help of the following code. GPIO 52 and GPIO 50 are configured as inputs so that GPIO 52 can be used to monitor the INIT_B pin and GPIO 50 can be used to monitor the DONE signal coming from the Xilinx FPGA. You can find this snippet of code in the function `CyFxConfigFpgaApplnInit()`present in the *cyfxconfigfpga.c* file.

```
/* Init the GPIO module */
        gpioClock
.fastClkDiv = 2;
         gpioClock.slowClkDiv = 0;
       gpioClock.simpleDiv = CY_U3P_GPIO_SIMPLE_DIV_BY_2;
       gpioClock.clkSrc = CY_U3P_SYS_CLK;
       gpioClock.halfDiv = 0;
apiRetStatus = CyU3PGpioInit(&gpioClock, NULL);

/* Configure GPIO 52 as input */
      gpioConfig.outValue = CyTrue;
         gpioConfig.inputEn = CyTrue;
         gpioConfig.driveLowEn = CyFalse;
         gpioConfig.driveHighEn = CyFalse;
         gpioConfig.intrMode = CY_U3P_GPIO_INTR_BOTH_EDGE;
apiRetStatus = CyU3PGpioSetSimpleConfig(FPGA_INIT_B, &gpioConfig);

/* Configure GPIO 50 as input */
apiRetStatus = CyU3PGpioSetSimpleConfig(FPGA_DONE, &gpioConfig);
```

3.3.3 DMA Channel Creation to Set Up the Data Transfer

The DMA Manual channel is created between the producer USB socket and the consumer CPU socket so that the configuration data that has been received on the Bulk out endpoint (0x01) of FX3 can be directed manually to the SPI module. The code that helps to create a DMA Manual channel is as follows. You can find this code snippet in the function `CyFxConfigFpgaApplnStart()`present in the *cyfxconfigfpga.c* file.

```
/* Create a DMA MANUAL channel for U2CPU transfer. The DMA size is set based on
the USB speed. */
    dmaCfg.size  = size;
    dmaCfg.count = CY_FX_SLFIFO_DMA_BUF_COUNT;
    dmaCfg.prodSckId = CY_FX_PRODUCER_USB_SOCKET;
    dmaCfg.consSckId = CY_U3P_CPU_SOCKET_CONS;
    dmaCfg.dmaMode = CY_U3P_DMA_MODE_BYTE;
/* Enabling the callback for produce event. */
    dmaCfg.notification = 0;
    dmaCfg.cb = NULL;
    dmaCfg.prodHeader = 0;
    dmaCfg.prodFooter = 0;
    dmaCfg.consHeader = 0;
    dmaCfg.prodAvailCount = 0;

apiRetStatus = CyU3PDmaChannelCreate (&glChHandleUtoCPU,CY_U3P_DMA_TYPE_MANUAL_IN,
                                      &dmaCfg);
```

3.3.4 Communication Between the FPGA Configuration Utility and FX3 Firmware

Two vendor commands are used to control the FX3 firmware functionality from the application that runs on the PC FPGA Configuration Utility. The FX3 firmware sets the events based on the vendor commands that it receives. It sets the event `CY_FX_CONFIGFPGAAPP_START_EVENT` for starting the FPGA configuration after it receives the vendor command `0xB2` (`VND_CMD_SLAVESER_CFGLOAD`) along with the length of the configuration bit file. The firmware also sets the event `CY_FX_CONFIGFPGAAPP_SW_TO_SLFIFO_EVENT` for switching to the Slave FIFO interface after it receives the vendor command `0xB1` (`VND_CMD_SLAVESER_CFGSTAT`) and only if the FPGA configuration is successful. The following code snippet is used to do this job. You can find this code in the function `CyFxSlFifoApplnUSBSetupCB ()` present in the *cyfxslfifosync.c* file.

```
if (bRequest == VND_CMD_SLAVESER_CFGLOAD)
   {
     if ((bReqType & 0x80) == 0)
           {
           CyU3PUsbGetEP0Data (wLength, glEp0Buffer, NULL);
           filelen = uint32_t)(glEp0Buffer[3]<<24)|(glEp0Buffer[2]<<16)|
                              (glEp0Buffer[1]<<8)|glEp0Buffer[0];
           glConfigDone = CyTrue;
/* Set CONFIGFPGAAPP_START_EVENT to start configuring FPGA */
           CyU3PEventSet(&glFxConfigFpgaAppEvent,
                         CY_FX_CONFIGFPGAAPP_START_EVENT, CYU3P_EVENT_OR);
           isHandled = CyTrue;
         }

   }
if (bRequest == VND_CMD_SLAVESER_CFGSTAT)
   {
      if ((bReqType & 0x80) == 0x80)
        {
           glEp0Buffer [0]= glConfigDone;
           CyU3PUsbSendEP0Data (wLength, glEp0Buffer);
/* Switch to slaveFIFO interface when FPGA is configured successfully*/
           if (glConfigDone)
            CyU3PEventSet(&glFxConfigFpgaAppEvent,
```

```
                        CY_FX_CONFIGFPGAAPP_SW_TO_SLFIFO_EVENT,
                        CYU3P_EVENT_OR);
            isHandled = CyTrue;
            }
    }
```

3.3.5 Actions Based on Events

The FX3 firmware continuously looks for the events mentioned previously and takes actions corresponding to those events. `SlFifoAppThread_Entry()` in the *cyfxslfifosync.c* file contains the following code.

```
/* Wait for events to configure FPGA */
txApiRetStatus = CyU3PEventGet (&glFxConfigFpgaAppEvent,
                                (CY_FX_CONFIGFPGAAPP_START_EVENT |
                                 CY_FX_CONFIGFPGAAPP_SW_TO_SLFIFO_EVENT),
CYU3P_EVENT_OR_CLEAR, &eventFlag,
                                 CYU3P_WAIT_FOREVER);
if (txApiRetStatus == CY_U3P_SUCCESS)
   {
         if (eventFlag & CY_FX_CONFIGFPGAAPP_START_EVENT)
          {
           /* Start configuring FPGA */
          CyFxConfigFpga(filelen);
          }
         else if ((eventFlag & CY_FX_CONFIGFPGAAPP_SW_TO_SLFIFO_EVENT))
          {
           /* Switch to SlaveFIFO interface */
       CyFxConfigFpgaApplnStop();
       CyFxSwitchtoslFifo();
       CyFxSlFifoApplnInit();
       CyFxSlFifoApplnStart();
          }
  }
```

3.4 Slave Serial Interface Implementation

`CyFxConfigFpga` is the function that implements the Xilinx Slave Serial interface. To start the configuration process, FX3 drives PROGRAM_B LOW. Then FX3 waits for INIT_B to go LOW, and it starts to clock the data when INIT_B becomes HIGH again. After sending all configuration data to the FPGA, FX3 decides whether the configuration is successful based on the DONE signal. The DONE signal will be set HIGH if the configuration is successful. See Figure 1 for clarity on the timing diagram. You can find this function in the *cyfxconfigfpga.c* file.

```
/* This is the function that writes configuration data to the Xilinx FPGA */
CyU3PReturnStatus_t CyFxConfigFpga(uint32_t uiLen)
{
      uint32_t uiIdx;
      CyU3PReturnStatus_t apiRetStatus;
      CyU3PDmaBuffer_t inBuf_p;
      CyBool_t xFpga_Done, xFpga_Init_B;

/* Pull PROG_B line to reset FPGA */
      apiRetStatus = CyU3PSpiSetSsnLine (CyFalse);
      CyU3PGpioSimpleGetValue (FPGA_INIT_B, &xFpga_Init_B);
      CyU3PGpioSimpleGetValue (FPGA_INIT_B, &xFpga_Init_B);
                if (xFpga_Init_B)
                {
                     glConfigDone = CyFalse;
                     return apiRetStatus;
                }
      CyU3PThreadSleep(10);
/* Release PROG_B line */
```

```
      apiRetStatus |= CyU3PSpiSetSsnLine (CyTrue);
      CyU3PThreadSleep(10);   // Allow FPGA to startup

/* Check if FPGA is now ready by testing the FPGA_Init_B signal */
    apiRetStatus |= CyU3PGpioSimpleGetValue (FPGA_INIT_B, &xFpga_Init_B);
    if( (xFpga_Init_B != CyTrue) || (apiRetStatus != CY_U3P_SUCCESS) ){

          return apiRetStatus;
    }
/* Start shifting out configuration data */
   for(uiIdx = 0; (uiIdx < uiLen) && glIsApplnActive; uiIdx += uiPacketSize )
      {
      if(CyU3PDmaChannelGetBuffer (&glChHandleUtoCPU, &inBuf_p, 2000) !=
      CY_U3P_SUCCESS){
         glConfigDone = CyFalse;
         apiRetStatus = CY_U3P_ERROR_TIMEOUT;
      break;
      }
      apiRetStatus = CyU3PSpiTransmitWords(inBuf_p.buffer , uiPacketSize);
      if (apiRetStatus != CY_U3P_SUCCESS)
      {
         glConfigDone = CyFalse;
         break;
      }
      if(CyU3PDmaChannelDiscardBuffer (&glChHandleUtoCPU) != CY_U3P_SUCCESS)
      {
       glConfigDone = CyFalse;
       apiRetStatus = CY_U3P_ERROR_TIMEOUT;
       break;
      }
    }
    CyU3PThreadSleep(1);

    apiRetStatus |= CyU3PGpioSimpleGetValue (FPGA_DONE, &xFpga_Done);
    if( (xFpga_Done != CyTrue) )
    {
       glConfigDone = CyFalse;
    apiRetStatus = CY_U3P_ERROR_FAILURE;
    }
    return apiRetStatus;
}
```

3.5 Reconfiguring the I/O Matrix

The FPGA Configuration Utility sends the vendor command 0xB1 (`VND_CMD_SLAVESER_CFGSTAT`) automatically after all the configuration data has been sent to FX3. FX3 firmware will switch to the Slave FIFO interface only if the FPGA configuration is successful. The following code snippet is used to reconfigure the I/O matrix. This is not necessary to do if the same I/O resources are used in the final application. However, in this case, the I/O matrix needs to be reconfigured because the Slave FIFO firmware (taken from AN65974) uses the 32-bit interface on GPIF II. Make sure that all the affected peripheral modules are deinitialized before reconfiguring the I/O matrix. In this application, the GPIO and SPI modules are deinitialized before reconfiguring the I/O matrix. The I/O matrix configuration needed to work as 32-bit Slave FIFO interface follows. Find this code snippet in the function `CyFxSwitchtoslFifo ()` present in the *cyfxslfifosync.c* file.

```
    io_cfg.useUart   = CyTrue;
    io_cfg.useI2C    = CyFalse;
    io_cfg.useI2S    = CyFalse;
    io_cfg.useSpi    = CyFalse;
#if (CY_FX_SLFIFO_GPIF_16_32BIT_CONF_SELECT == 0)
    io_cfg.isDQ32Bit = CyFalse;
    io_cfg.lppMode   = CY_U3P_IO_MATRIX_LPP_UART_ONLY;
```

```
#else
    io_cfg.isDQ32Bit = CyTrue;
    io_cfg.lppMode   = CY_U3P_IO_MATRIX_LPP_DEFAULT;
#endif
    /* No GPIOs are enabled. */
    io_cfg.gpioSimpleEn[0]  = 0x00000000;
    io_cfg.gpioSimpleEn[1]  = 0;
    io_cfg.gpioComplexEn[0] = 0;
    io_cfg.gpioComplexEn[1] = 0;
    status = CyU3PDeviceConfigureIOMatrix (&io_cfg);
```

3.5.1 Endpoint Configuration and Restoring the Sequence Number

The same producer endpoint (EP1 OUT BULK) is used for FPGA configuration and for transferring data from USB to the FPGA connected to FX3 over the Slave FIFO interface after the FPGA configuration is successful. However, the EP1 is reconfigured to enable burst transfers to support high-bandwidth data transfers after the Slave FIFO interface is enabled. So, the CyU3PSetEpConfig API is called twice for configuring the same endpoint. This API clears the sequence number associated with the endpoint. Data transfers fail when the USB 3.0 Host and FX3 device find a mismatch in the sequence number. Therefore, you need to restore the sequence number so that the USB 3.0 Host can perform successful data transfers even after reconfiguring the EP1. This is valid only for USB 3.0 data transfers.

The `CyU3PUsbGetEpSeqNum` API gets the current sequence number for an endpoint, and `CyU3PUsbSetEpSeqNum` sets the active sequence number for an endpoint.

3.6 Integrating the Configuration Firmware into Your Design

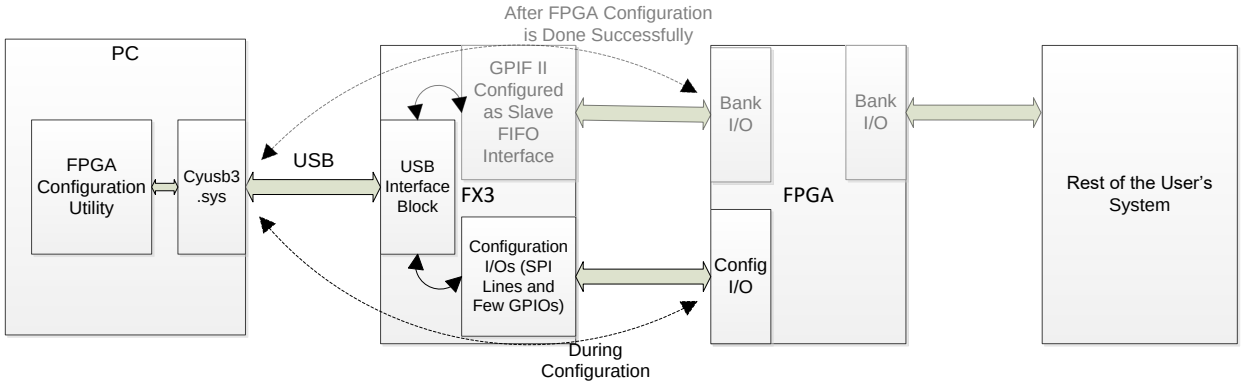
This section explains how to integrate the configuration firmware into your design. See the project in the attachments to this application note while you read the following steps.

1. Import the *cyfxconfigfpga.c* and *cyfxconfigfpga.h* files into your project.
2. Comment the `main()` function in your design because `main()` is implemented in *cyfxconfigfpga.c*.
3. Call `CyFxConfigFpgaApplnInit()` in the thread entry function in place of your application initialization function. In this example, `CyFxConfigFpgaApplnInit()` is called in function `SlFifoAppThread_Entry()` in place of `CyFxSlFifoApplnInit()`.
4. Call `CyFxConfigFpgaApplnStart()` in the USB event callback function in place of your application start function. In this example, `CyFxConfigFpgaApplnStart()`is called in function `CyFxSlFifoApplnUSBEventCB` in place of `CyFxSlFifoApplnStart()`.
5. Comment out the code snippet that handles the USB enumeration part in `CyFxSlFifoApplnInit()` since the `CyFxConfigFpgaApplnInit()` already handles it.
6. Add the support for vendor commands and events as they are implemented in this example.
7. The I/O matrix needs to be reconfigured if your application requires a different set of resources. In this example, the I/O matrix reconfiguration code can be found in the function `CyFxSwitchtoslFifo ()` in *cyfxslfifosync.c*.
8. Change your application thread entry function similar to `SlFifoAppThread_Entry()`.

3.7 Software Details

This section describes the host application and the USB 3.0 driver needed for running the project files attached to this application note. Figure 5 shows the system-level block diagram including the Host application and drivers needed on PC to configure the FPGA interfaced to FX3.

Figure 5. System-Level Block Diagram Showing Software Details on PC Side



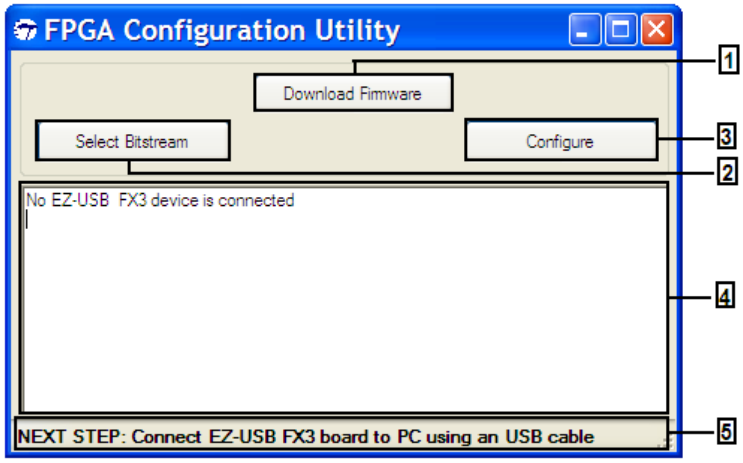
3.7.1 Host Application

The FPGA Configuration Utility is developed specifically for this application and is available as an attachment.

USB Driver: *cyusb3.inf* and *cyusb3.sys* are part of the EZ-USB FX3 SDK.

An example host application, the FPGA Configuration Utility, created for configuring the FPGA is included in the design. The application is developed in Visual C# 2008 Express Edition using the Cypress Application Development Library *CyUSB.dll*, which is included in the Cypress SuperSpeed USBSuite. The device must be bound to *CyUSB3.sys*, a general-purpose driver developed by Cypress. The Host application provided with this application note serves as a reference for developing an FPGA Configuration Utility. It provides an option to download the firmware image into FX3 RAM and the flexibility to select the bitstream (*.bin*) file for Xilinx FPGA configuration. In addition, this application gives the status of each step and shows the next step to run the demo successfully. Figure 6 shows an FPGA Configuration Utility elements annotation.

Figure 6. FPGA Configuration Utility Elements Annotation



1: Downloads the firmware image into FX3 RAM

2: Selects the configuration file for the Xilinx FPGA (*.bin* file)

3: Downloads the selected configuration file over FX3

4: Displays the status of each step during the configuration of the Xilinx FPGA

5: Displays the next step

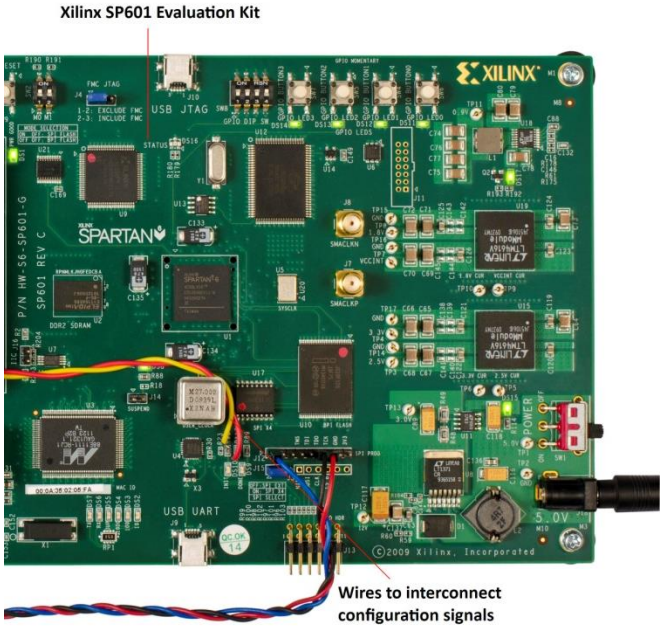
4 Operating Instructions

This section shows you how to configure the Xilinx FPGA connected to the FX3 SuperSpeed Explorer Kit with the help of software and firmware projects that are attached to this application note. Make the hardware connections between the Xilinx Spartan-6 SP601 Evaluation Kit and the FX3 SuperSpeed Explorer Kit (or CYUSB3KIT-001), as Table 3 shows. These connections are the same as the ones shown in the hardware interconnection diagram (Figure 2). In addition, connect the FX3 SuperSpeed Explorer Kit (or CYUSB3KIT-001) to the Xilinx Spartan-6 SP601 Evaluation Kit with the help of the Samtec-to-FMC connector. Note that the hardware setup used for this application note is the same as the one used in AN65974, but you need five wires to connect the signals required to configure the FPGA.

Table 3. Hardware Connections Between Xilinx SP601 Evaluation Kit and FX3 Explorer Kit (or CYUSB3KIT-001)

Signal Name	Pin Placement on SP601 Evaluation Kit	Pin Placement on FX3 SuperSpeed Explorer Kit	Pin Placement on CYUSB3KIT-001
PROGRAM_B	Pin 1 of J12	Pin 23 of J7	Pin 2 of J102
INIT_B	One end of resistor R90 (as shown in Figure 7)	Pin 31 of J7	Pin 6 of J20
CCLK	Pin 7 of J12	Pin 27 of J7	Pin 2 of J101
DIN	Pin 6 of J12	Pin 19 of J7	Pin 2 of J104
DONE	One end of R113 or LED DS9 (as shown in Figure 7)	Pin 37 of J7	Pin 4 of J20

Figure 7 Hardware Connections on Xilinx SP601 Evaluation Kit



1. Run *Template.exe* present in the *FPGA_Config_Utility\bin\Debug* folder and see the utility that appears on the screen. The following status message appears: **No EZ-USB FX3 device is connected.**
2. Connect the EZ-USB FX3 Explorer Kit or a CYUSB3KIT-001 to a PC using a USB cable, as Figure 8 shows. Then observe the status message **EZ-USB FX3 Bootloader device connected** that appears in the text box, as Figure 9 shows.
3. Click **Download Firmware** to download the firmware image into FX3 RAM and browse to the location of the *ConfigFpgaSlaveFifoSync.img* file, as Figure 10 shows. Then, click **Open**.

Figure 8. FPGA Configuration Utility When No EZ-USB FX3 Device Is Connected

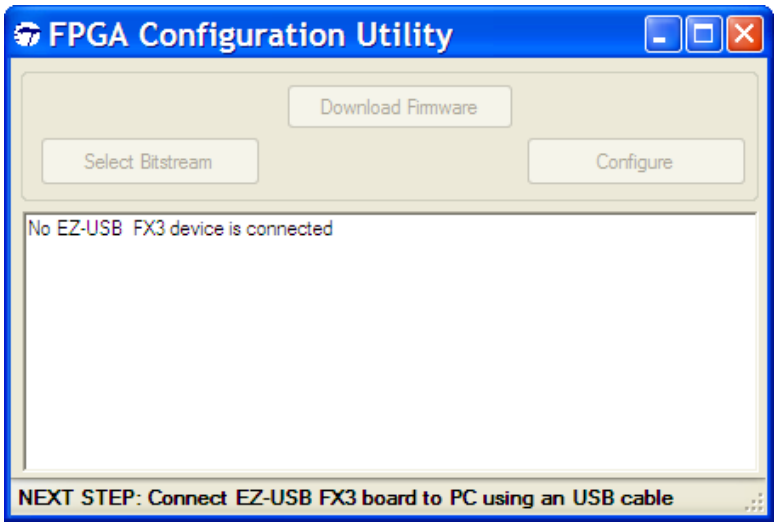


Figure 9. FPGA Configuration Utility after connecting the EZ-USB FX3 Kit to PC

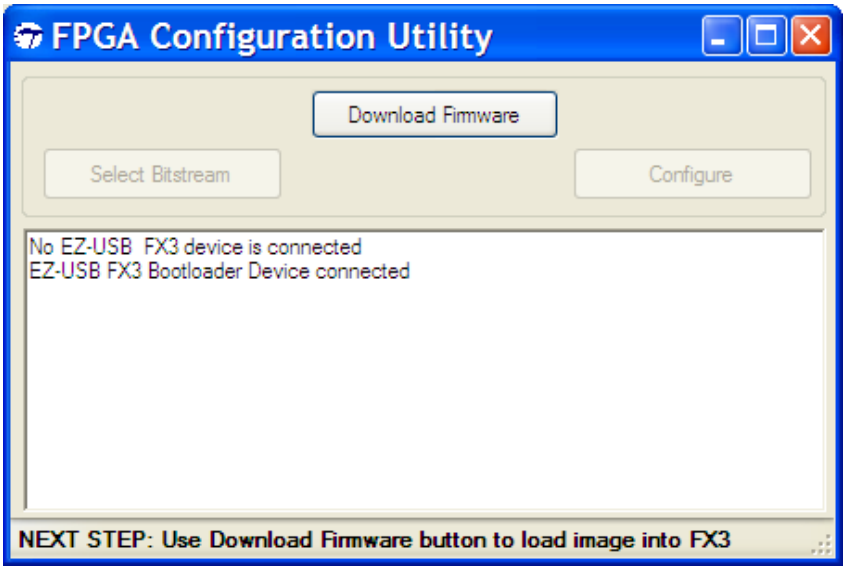
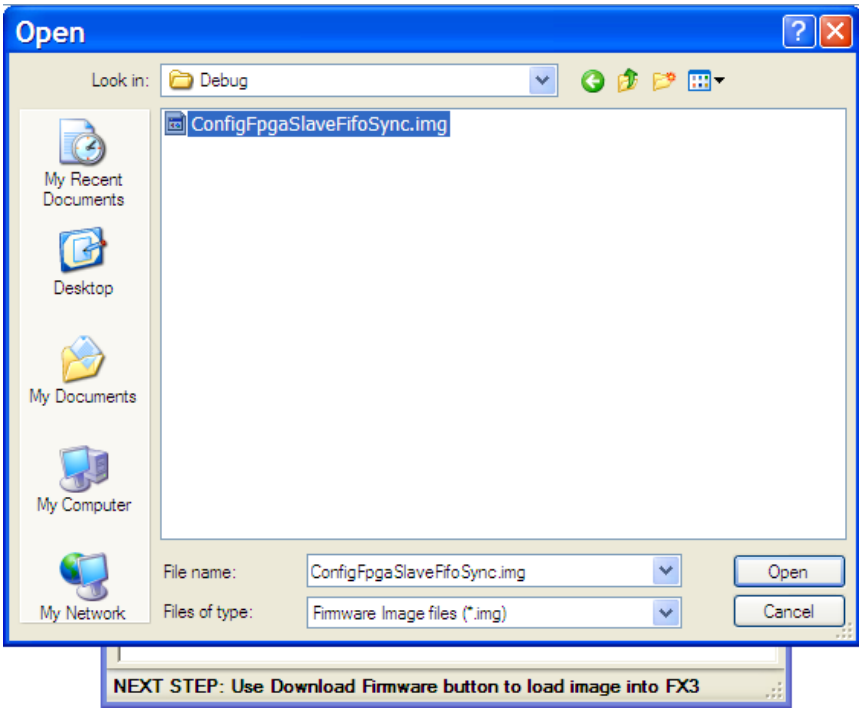


Figure 10. Selecting the FX3 Firmware Image



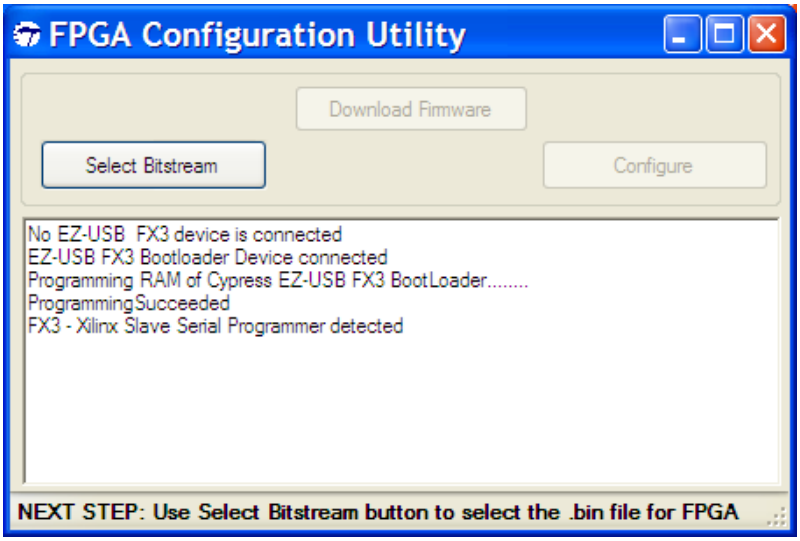
The following status messages appear, as Figure 11 shows:

Programming RAM of Cypress EZ-USB FX3 BootLoader…….

Programming Succeeded

FX3 – Xilinx Slave Serial Programmer detected

Figure 11. FPGA Configuration Utility After Image File is Downloaded into FX3 RAM

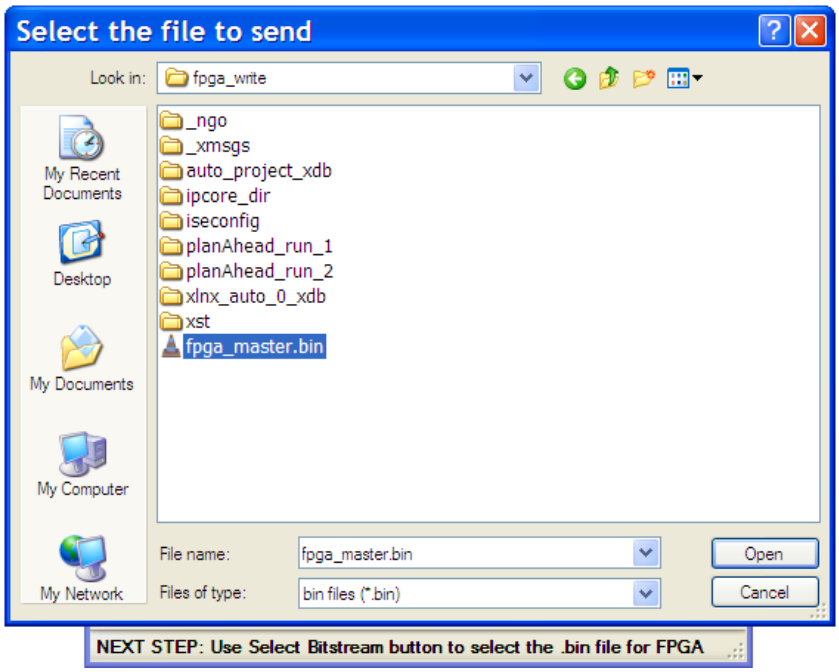


4. As the message is appearing on the utility, click **Select Bitstream** to select the *.bin* file for the FPGA.

If the *.bin* file is not available and you have only the *.bit* file, then convert the *.bit* to *.bin* by using the PromGen command line. Alternatively, you can use the iMPACT PROM File Formatter to create a *.bin* for a Xilinx PROM. Visit www.xilinx.com/support.html to get support on generating the *.bin* file.

5. Browse to the location of the *fpga_master.bin* file, as Figure 12 shows. Click **Open**.

Figure 12. Selecting the Configuration Bit File (.bin) for the Xilinx FPGA



6. Click **Configure** to configure the Xilinx FPGA, as shown in Figure 13. If the FPGA is configured successfully, then the FX3 firmware switches to the Slave FIFO interface. The following status messages appear, as Figure 14 shows.

   Writing data to FPGA

   Configuration data has been sent to FPGA

   Configurations Successful

   FX3 Slave FIFO interface is activated

Figure 13. FPGA Configuration Utility After Selecting the .bin File

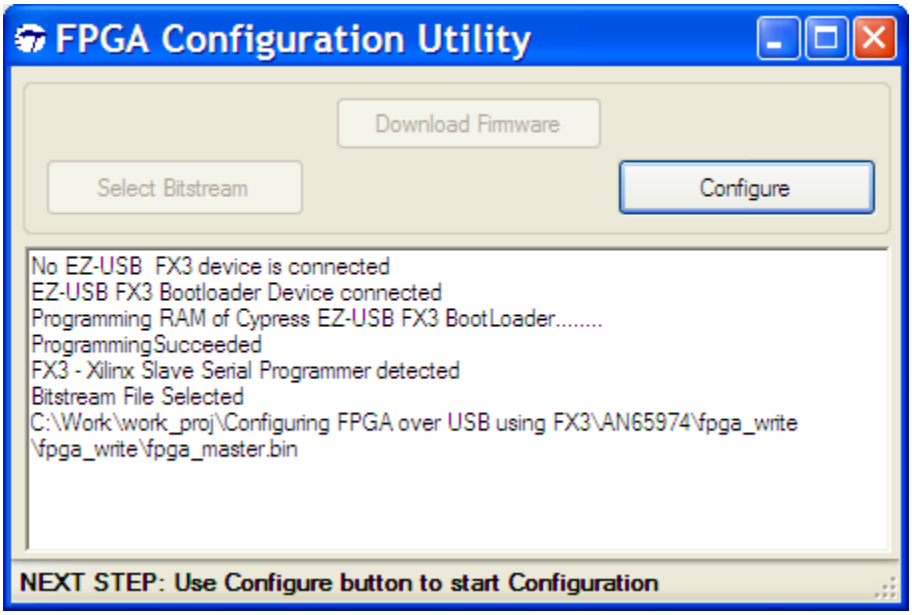
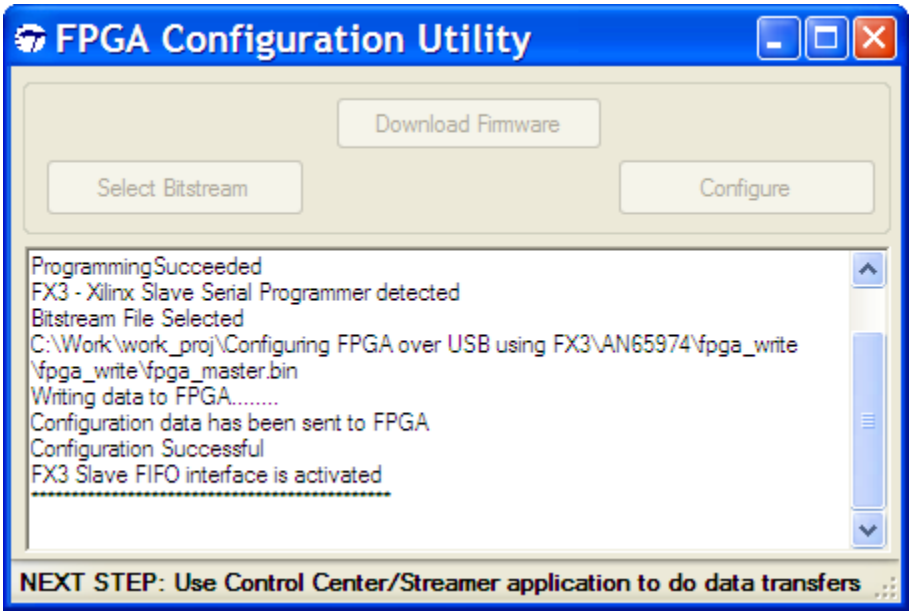
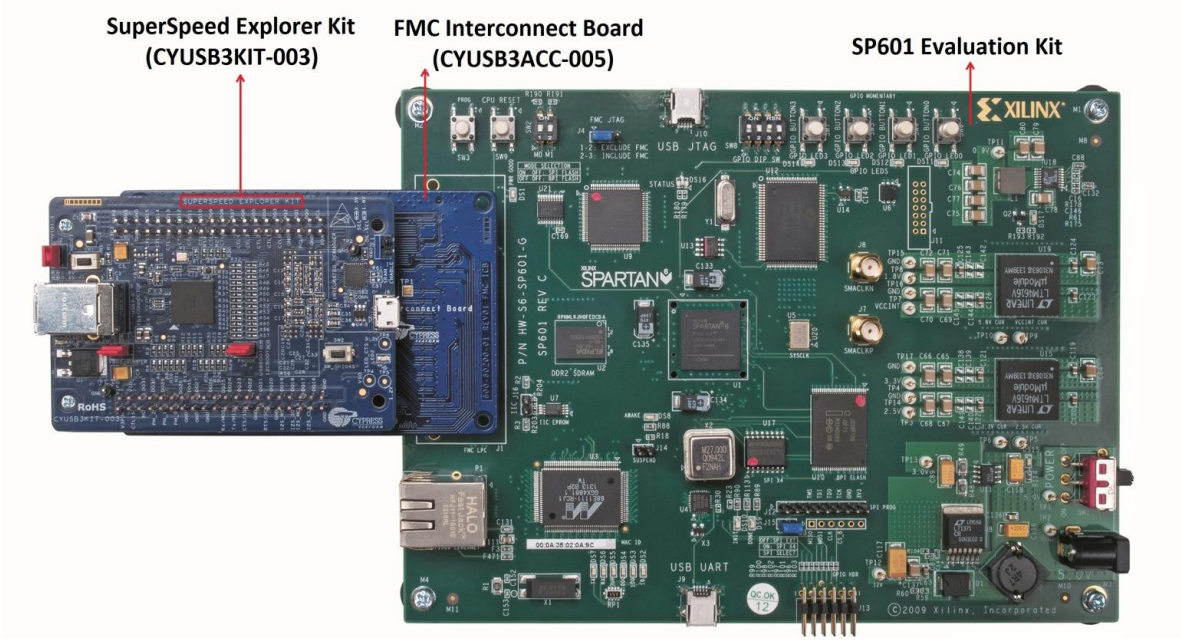


Figure 14. FPGA Configuration Utility After FPGA Configuration Is Done Successfully



You can observe the DS9 LED glowing on the Xilinx FPGA board after the configuration is successful. It does not glow if something goes wrong during configuration. Figure 15 shows the DS9 LED glowing.

Figure 15. Hardware Setup After FPGA Is Configured Successfully

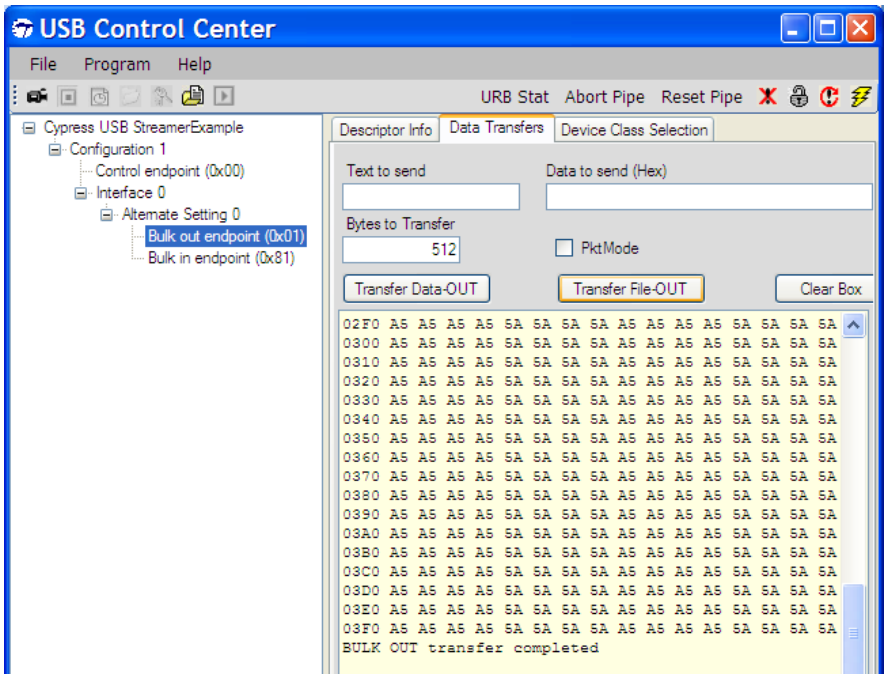


7. Use the Control Center application to verify the loopback operation between FX3 and the Xilinx FPGA. Make sure that the SW8 switch on SP601 Evaluation kit is kept in the following modes:

SW8[1]	SW8[2]	SW8[3]	SW8[4]
OFF	OFF	OFF	ON

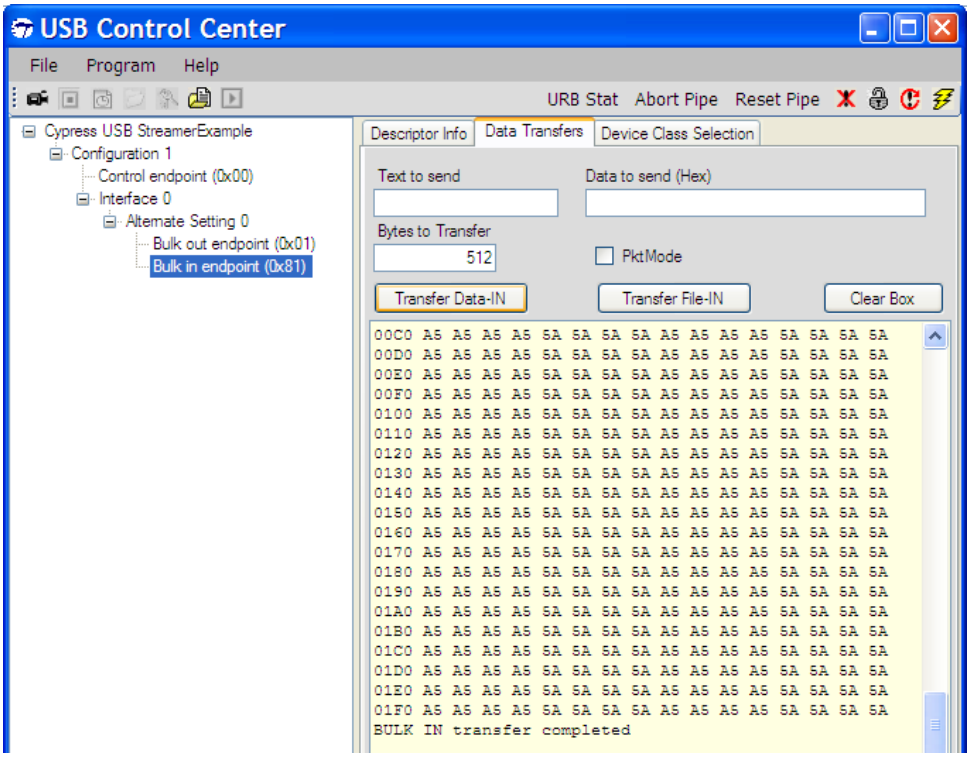
Go to **Bulk out endpoint (0x01)** and click the **Transfer File-OUT** button to transfer a file called *TEST.txt,* which is in the same folder. Then, you can see the series **A5 A5 A5 A5 5A 5A 5A 5A** get transmitted successfully to the **Bulk out endpoint**, as Figure 16 shows.

Figure 16. USB Control Center After Transferring TEST.txt File Over Bulk Out Endpoint



8. Select **Bulk in endpoint** and click **Transfer Data-IN**. Observe that the received data is the same as the data that has been transmitted to the **Bulk out endpoint**, as Figure 17 shows. The data path is as follows: **Control Center** > **Bulk out endpoint of FX3** > **FPGA reads the data from Bulk out endpoint** > **FPGA writes the same data to Bulk in endpoint of FX3** > **Control Center**.

Figure 17. USB Control Center After Data-IN to Get Data from Bulk in Endpoint



5 Summary

This application note demonstrated a solution for efficiently configuring a Xilinx FPGA over USB using Cypress FX3. You can integrate this solution into a system in which an FPGA acts as an interface with FX3 for USB 3.0 functionality, eliminating the need for a dedicated programming circuit to configure the FPGA.

6 Associated Project Files

Table 4 describes the files attached to this application note.

Table 4. Description of Files in the Attachment

Folder name	Description
FPGA Configuration Utility	Source code of the PC-side application
FX3 firmware	Source code of the FX3 firmware
fpga_write	Source code of Xilinx FPGA acting as a master device. This is same as the FPGA code available with AN65974.
bin	It contains the following files: *TEST.txt*—Data file that can be used to test the loopback operation between FX3 and FPGA. *ConfigFpgaSlaveFifoSync.img*—Image file of FX3 firmware. *Template.exe*—Executable file of FPGA Configuration Utility.

7 References

- CYUSB3014 datasheet
- Getting Started with FX3
- FX3 Slave FIFO Interface
- Spartan-6 Generation Configuration User Guide – Xilinx UG380
- Xilinx Spartan-6 FPGA SP601 Evaluation Kit
- Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode
- FX3 + FPGA + HelionVision ISP-Based Industrial Camera Reference Design – KBA222700

About the Author

Name: Rama Sai Krishna

Title: Application Engineer Staff

Document History

Document Title: AN84868 - Configuring a Xilinx FPGA Over USB Using Cypress EZ-USB FX3

Document Number: 001-84868

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3883306	RSKV	01/25/2013	New application note.
*A	4399551	RSKV	06/05/2014	Provided link for FPGA configuration guide in the introduction section. Updated Table 2. Updated “Reconfiguring the I/O Matrix” section. Added “Endpoint Configuration and Restoring the Sequence Number” section. Added Table 3 to list the hardware connections between the Xilinx SP601 Evaluation Kit and the CYUSB3KIT-001.
*B	4660848	AMDK	02/13/2015	Updated firmware to work in release build Added link to SuperSpeed USB code examples Sunset Review
*C	4909562	MDDD	12/21/2015	Fixed broken links Updated Figure based on new FX3 Explorer Kit Added connection information in Table 3 for FX3 Explorer Kit Updated template
*D	5701881	BENV	04/19/2017	Updated logo and copyright
*E	5851102	MDDD	04/26/2018	Changed the title Added the older FX3 kit MPN Updated the pin mapping for CYUSB3KIT-001 Updated Introduction with reference design link Added instructions to test data transfer with FPGA board Added KBA link in References

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

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709