AN84868

Configuring an FPGA over USB Using Cypress EZ-USB® FX3™

Author: Rama Sai Krishna. V
Associated Project: Yes
Associated Part Family: CYUSB3014
Software Version: EZ-USB FX3 SDK 1.2.1
Related Application Notes: AN65974 Designing with the EZ-USB® FX3™ Slave FIFO Interface

If you have a question, or need help with this application note, contact the author at rskv@cypress.com.

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 using 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

Introduction

FX3 has a configurable, parallel, general programmable interface, called GPIF II, which can connect to external devices. Those include image sensors, external processors, ASICs, or FPGAs. As a result, users can integrate USB 3.0 capability into almost any system.

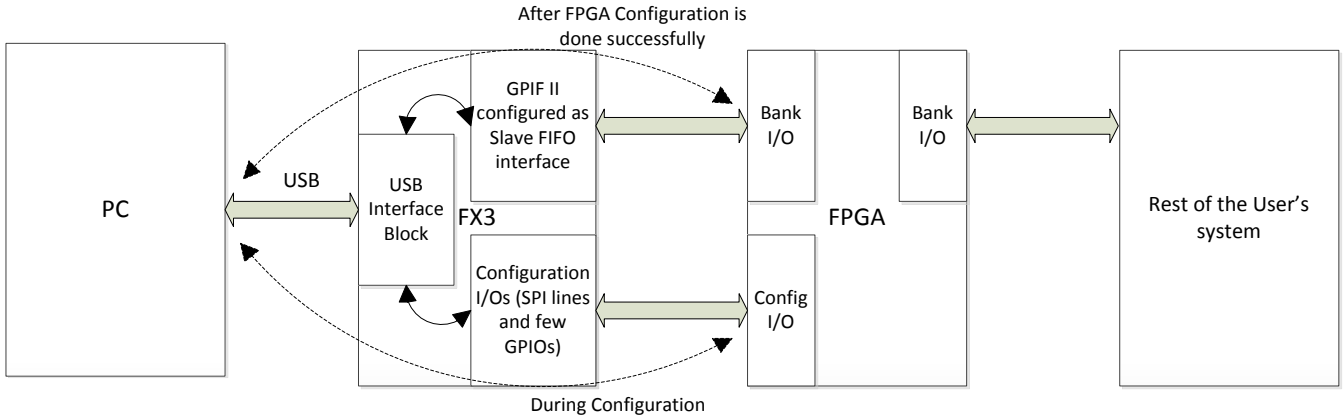
In addition, FX3 provides interfaces to connect to serial peripherals, such as UART, SPI, I2C, and I2S.

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, please 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.

Acting as a master, FX3 can configure the Xilinx FPGA in two modes: Slave Parallel (SelectMAP) and Slave Serial. This application note describes only the Slave Serial mode. It also describes how FX3 firmware switches to Slave FIFO interface after the FPGA configuration is completed. Figure 1 shows the block diagram in which FX3 configures the FPGA at the start and then switches to the Slave FIFO interface after the configuration is successful.

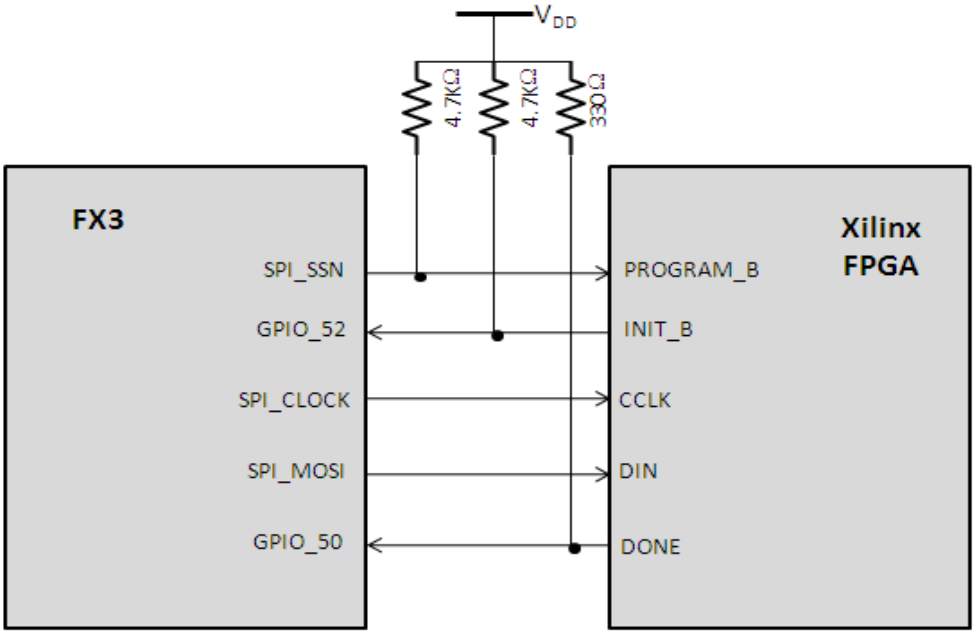
In the following sections, let's look at the details of the Xilinx Slave Serial configuration interface and its design implementation using FX3.

Figure 1. System-level application block diagram



Xilinx Slave Serial Configuration Interface

Figure 2. Hardware connection between FX3 and Xilinx Spartan-6 FPGA



Note: PROGRAM_B, INIT_B and DONE are open-drain signals. Connect pull-up resistors of suitable value on these lines. Resistor values mentioned in Figure 2 are taken from the Xillinx FPGA Configuration Guide. Please note that there is no need to connect these pull-up resistors if FX3 DVK is used but pull-up resistors should be placed in the final design.

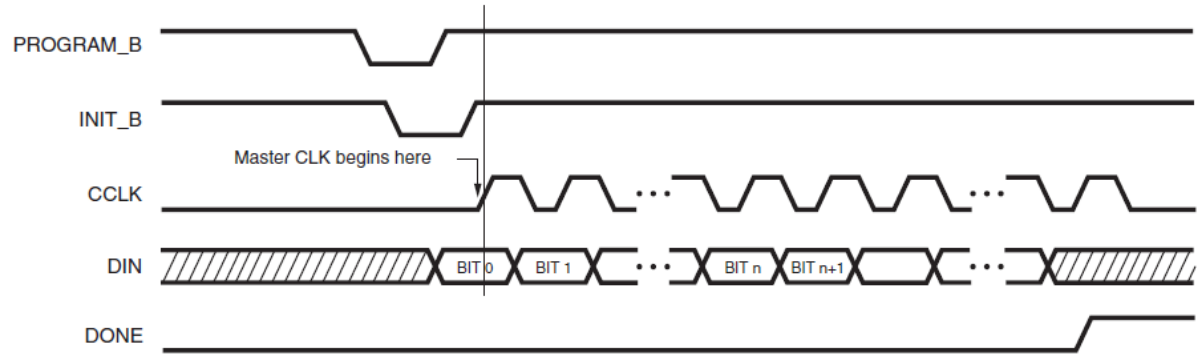
Table 1 shows the interface signals in the Slave Serial Interface.

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 pulsed 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



Implementation

FX3 starts the configuration by pulsing PROGRAM_B and monitoring the INIT_B pin. When it 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 start-up (see clocking sequence diagram in Figure 3.

Hardware Details

Hardware boards

- Xilinx SP601 Evaluation kit
- EZ-USB FX3 DVK
- Samtec-to-FMC interconnection board
- Wires to interconnect configuration signals

The SPI hardware block in FX3 serializes the configuration data from the PC. SPI_SSN (slave select), SPI_CLOCK

and SPI_MOSI of FX3 are connected to the PROGRAM_B, CCLK and DIN of Xilinx FPGA, respectively. 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.

FX3 Firmware

The attached firmware has the following components:

1. Configuration of the Xilinx FPGA connected to FX3 over the Slave Serial interface.
2. The Slave FIFO interface configuration works exactly 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 2 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. Please use FX3 API Guide as a reference for more details on the FX3 SDK APIs.

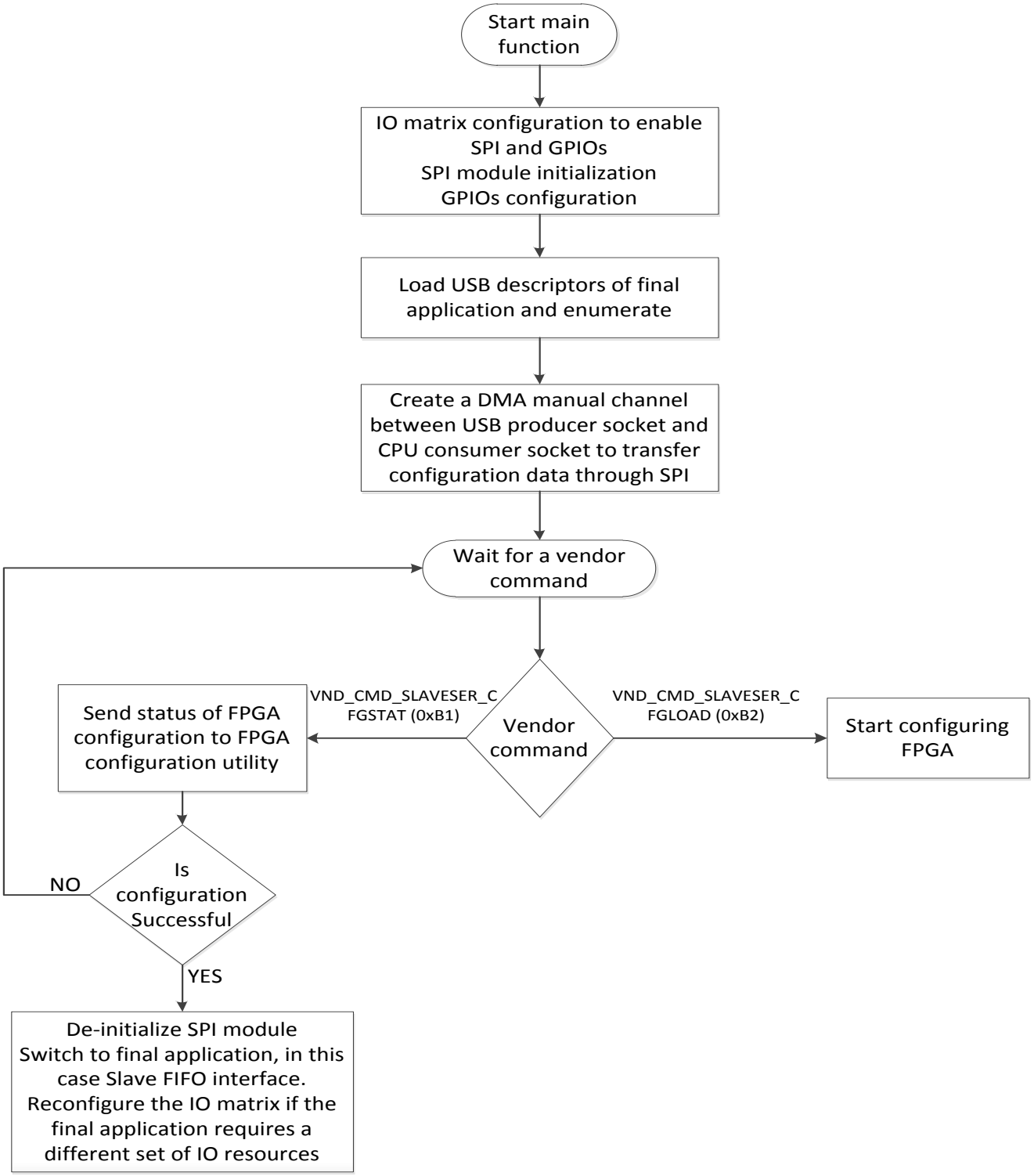
Table 2 describes the files present in the firmware source code, which is attached to this application note.

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 Configure FPGA in Slave Serial mode example. It contains the following functions: • main () • CyFxConfigFpgaApplnInit () • CyFxConfigFpgaApplnStart () • CyFxConfigFpgaApplnStop () • CyFxConfigFpga()
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 () • SlFifoAppThread_Entry () • CyFxSwitchtoslFifo () • CyFxSlFifoApplnInit () • CyFxSlFifoApplnStart () • CyFxSlFifoApplnStop () • CyFxSlFifoApplnUSBEventCB () • CyFxSlFifoApplnUSBSetupCB () • CyFxSlFifoApplnDebugInit()
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. This file 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.

The flow chart in Figure 4 describes the FX3 firmware.

Figure 4. FX3 firmware flow chart



I/O matrix configuration

In the `main()` function, configure the I/O matrix (shown below) 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 DONE and INIT_B pins of Xilinx FPGA (see Figure 2 for the hardware interface diagram). You can find this piece of code in the function `main()` present in file `cyfxconfigfpga.c`.

```
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);
```

SPI module initialization

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

```
/* 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);
```

GPIOs configuration

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

```
/* 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);
```

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 shown below. You can find this piece of code in the function `CyFxConfigFpgaApplnStart()` present in the file `cyfxconfigfpga.c`

```
/* 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);
```

Communication between FPGA Configuration Utility and FX3 firmware

Two vendor commands are used to control the FX3's firmware functionality from the application that runs on the PC "FPGA Configuration Utility". FX3's 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 piece of code is used to do the job that has been described above. You can find this code in the function `CyFxSlFifoApplnUSBSetupCB()` present in the file `cyfxslfifosync.c`.

```
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;
                  }

            }
          }
```

Actions based on events

FX3's firmware continuously looks for the events mentioned above and takes the actions corresponding to those events. `SlFifoAppThread_Entry()` in `cyfxslfifosync.c` has the code shown below.

```
/* 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();

       }
  }
```

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 the whole configuration data to 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. Please see Figure 1 for better clarity on the timing diagram. You can find this function in `cyfxconfigfpga.c`.

```
/* 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;

}
```

Reconfiguring the I/O matrix

The FPGA Configuration Utility sends the vendor command 0xB1 (`VND_CMD_SLAVESER_CFGSTAT`) automatically after the entire configuration data has been sent to FX3. FX3 firmware will switch to the Slave FIFO interface only if the FPGA configuration is successful. The piece of code shown below 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 re-configured because the Slave FIFO firmware (taken from AN65974) uses the 32-bit interface on GPIF II. Make sure that `CyU3PSpiDeInit()` is called before doing the reconfiguration of the I/O matrix. The I/O matrix configuration needed to work as 32-bit Slave FIFO interface is shown below. Find this piece of code in the function `CyFxSwitchtoslFifo ()` present in the file `cyfxslfifosync.c`.

`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;
    /*io_cfg.lppMode   = CY_U3P_IO_MATRIX_LPP_DEFAULT;*/
#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);
```

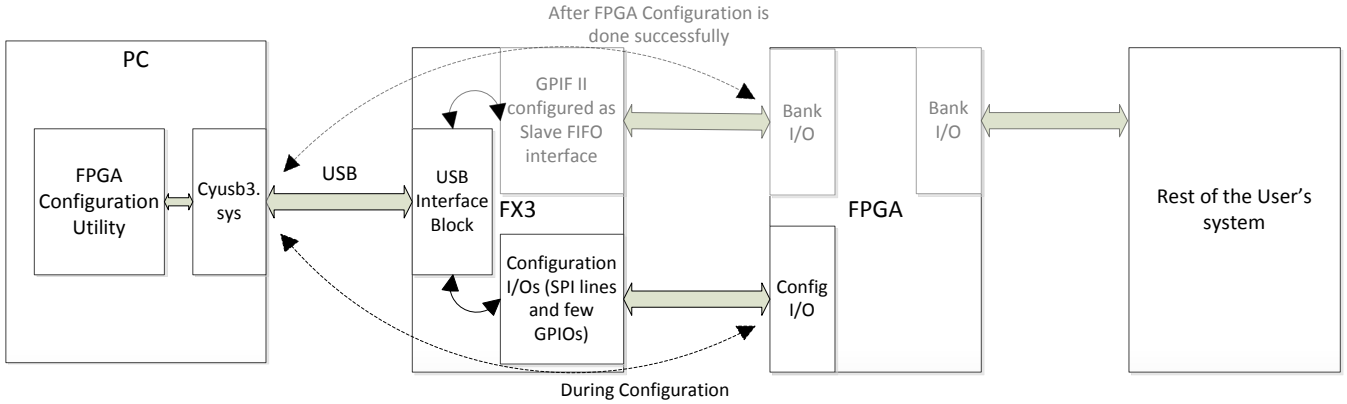
Integrate the configuration firmware into your design

This section shows you how to integrate the configuration firmware into your design. Please refer to the project in the attachments to this application note for reference while you read the steps described below:

1. Import `cyfxconfigfpga.c` and `cyfxconfigfpga.h` files to your project.
2. Comment `main()` function in your design because `main()` is implemented in `cyfxconfigfpga.c`
3. Call `CyFxConfigFpgaApplnInit()` in thread entry function in the place of your application initialization function. In this example, `CyFxConfigFpgaApplnInit()` is called in function `SlFifoAppThread_Entry()` in the place of `CyFxSlFifoApplnInit()`
4. Call `CyFxConfigFpgaApplnStart()` in USB event call back function in the place of your application start function. In this example, `CyFxConfigFpgaApplnStart()` is called in function `CyFxSlFifoApplnUSBEventCB` in the place of `CyFxSlFifoApplnStart()`.
5. Comment out the piece of code that handles 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()`.

Software Details

Figure 5. System-level block diagram showing software details on PC side



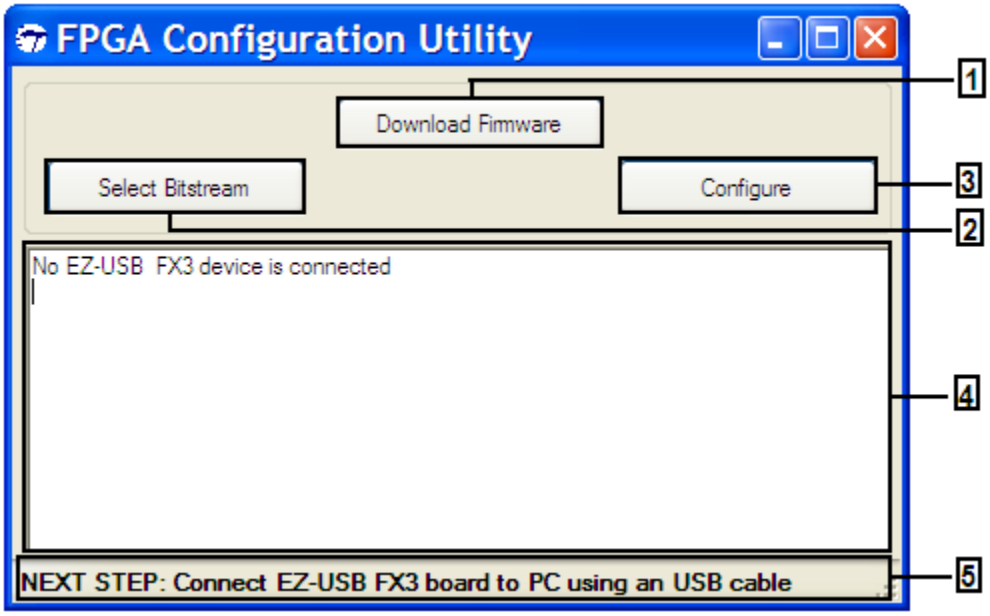
Host Application

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 has to be bound to CyUSB3.sys, a general-purpose driver developed by Cypress. This host application provided with this application note serves as a reference example for developing an FPGA Configuration Utility. This provides an option to download the firmware image into FX3's 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. The FPGA Configuration Utility elements annotation is shown in Figure 6.

Figure 6. FPGA Configuration Utility elements annotation



1. Downloads the firmware image into FX3's RAM

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

3. Downloads the selected configuration file over FX3

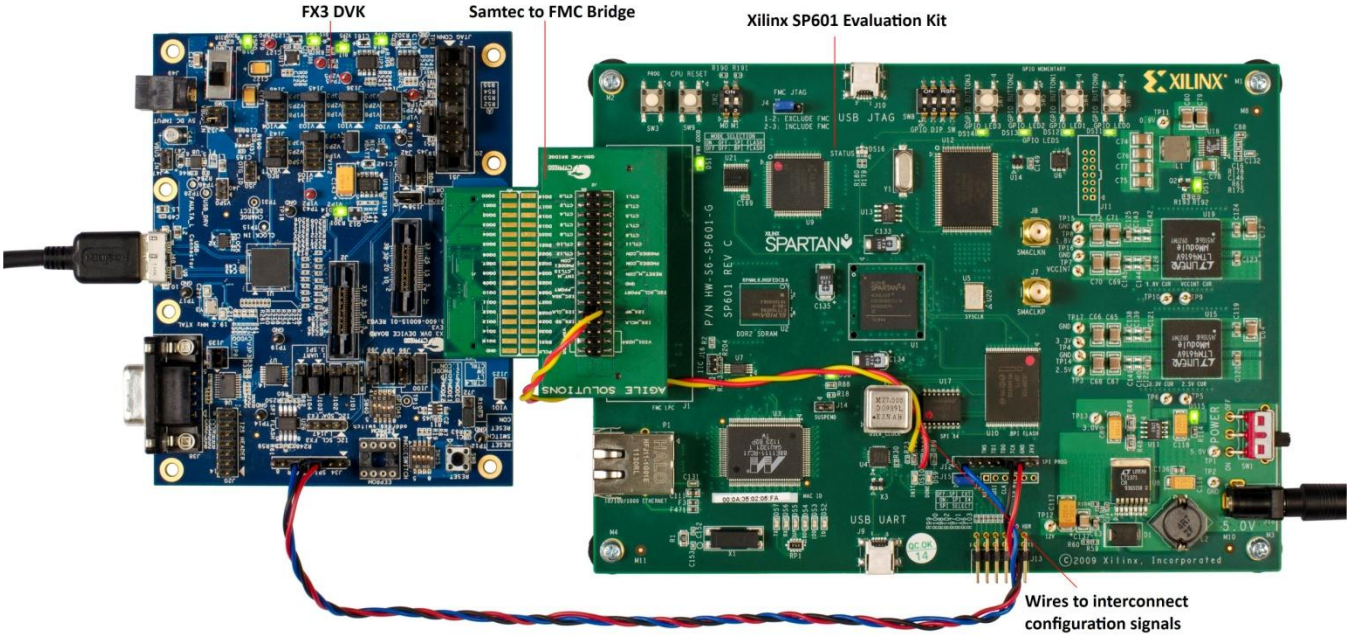
4. Displays the status of each step during the configuration of the Xilinx FPGA

5. Displays the next step

Operating Instructions

This section shows you how to configure the Xilinx FPGA connected to the FX3 DVK 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 DVK, as Figure 7 shows. These connections are the same as the ones shown in the hardware interconnection diagram (Figure 2). In addition, connect the FX3 DVK 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.

Figure 7. Hardware connections between Xilinx SP601 evaluation kit and FX3 DVK

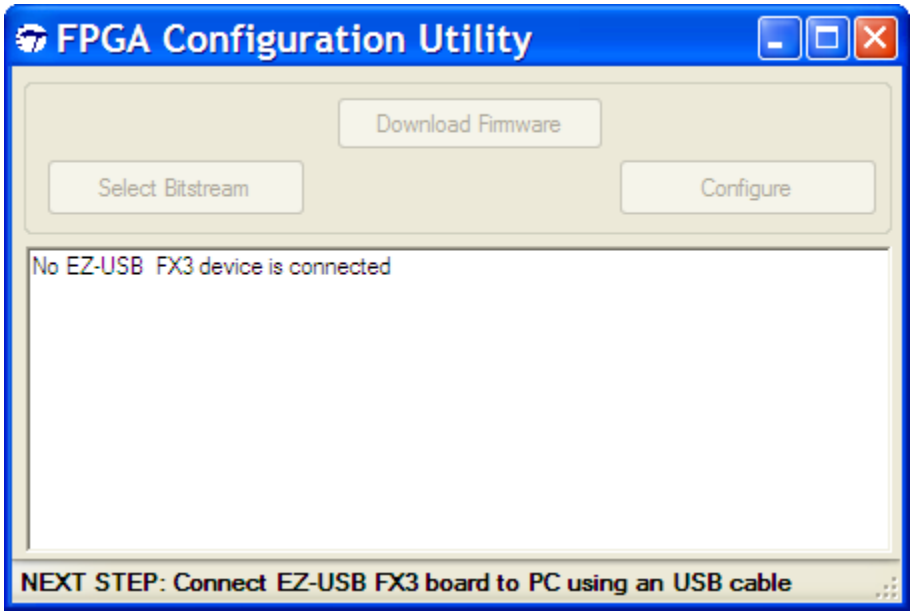


1. Run Template.exe present in the folder FPGA_Config_Utility\bin\Debug and see the utility that appears on the screen. You can observe the following status message appearing on the utility.

No EZ-USB FX3 device is connected

As Figure 8 shows, the next step is to connect an EZ-USB FX3 board to the PC using a USB cable.

Figure 8. FPGA Configuration Utility when no EZ-USB FX3 device is connected

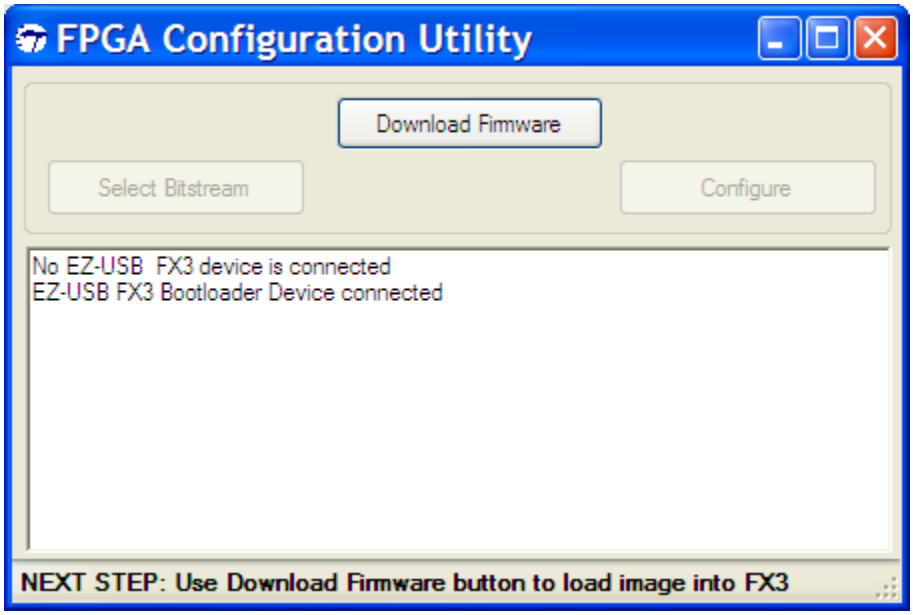


2. Connect the EZ-USB FX3 DVK to a PC using a USB cable. Then you can identify the following status message

EZ-USB FX3 Bootloader device connected

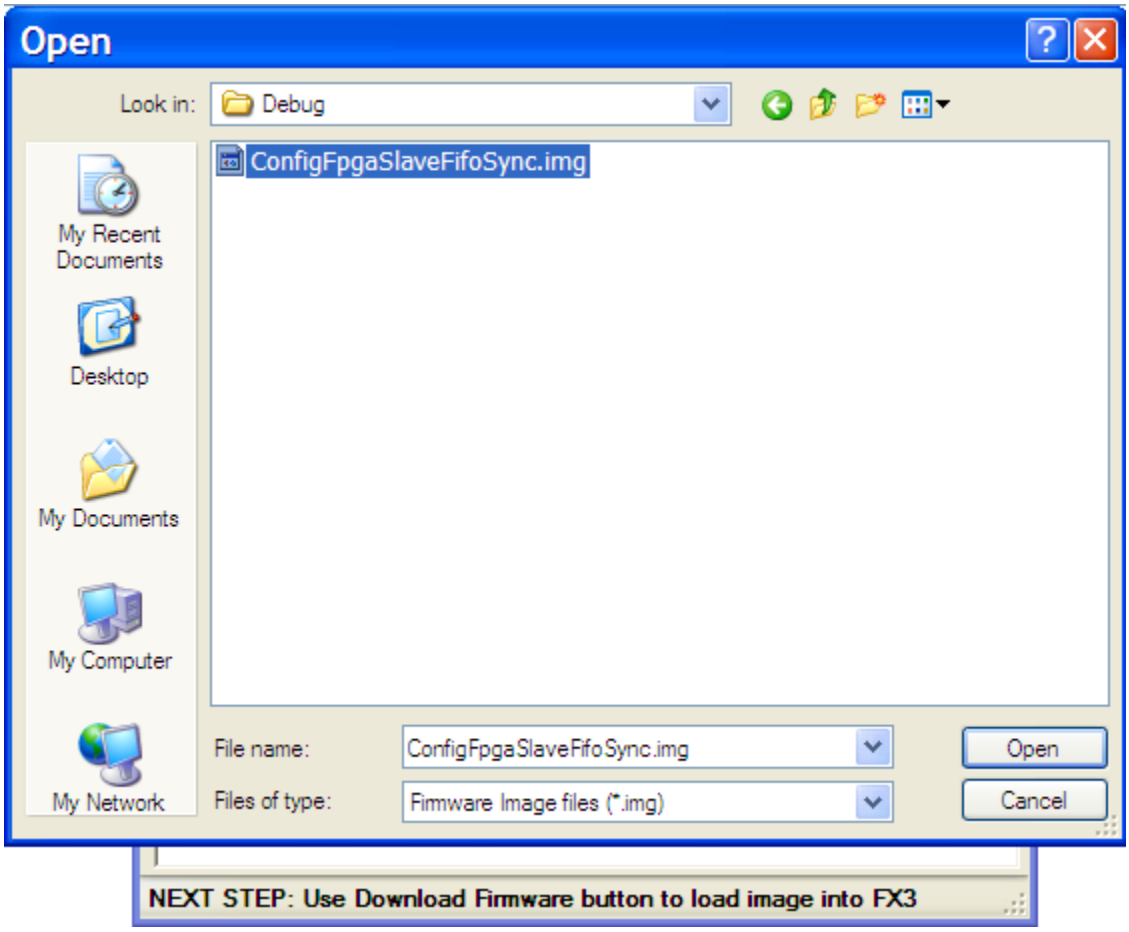
appearing in the text box (as Figure 9 shows), and the next step is to download the firmware image into FX3's RAM with the help of "Download Firmware" button.

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



3. Click the "Download Firmware" button and browse to the location of the "ConfigFpgaSlaveFifoSync.img" file, as Figure 10 shows. Then, click Open.

Figure 10. Selecting the FX3's firmware image



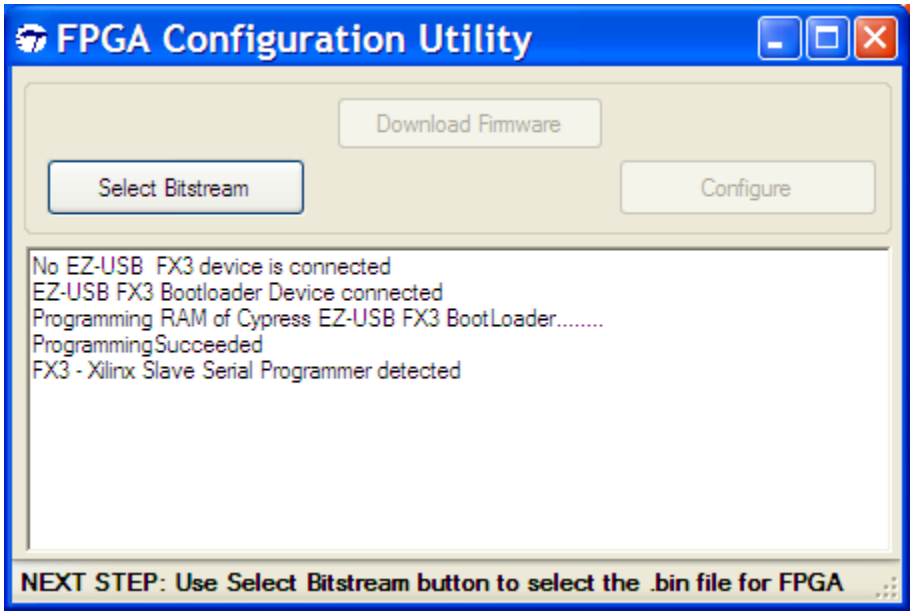
The following status messages appear as part of this step, as Figure 11 shows.

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's RAM



As it is appearing on the utility, the next step is to click the "Select Bitstream" button to select the .bin file for the FPGA.

4. Browse to the location where you have "fpga_master.bin" file, as Figure 12 shows. Click Open.

Figure 12. Selecting the configuration bit file (.bin) for the Xilinx FPGA

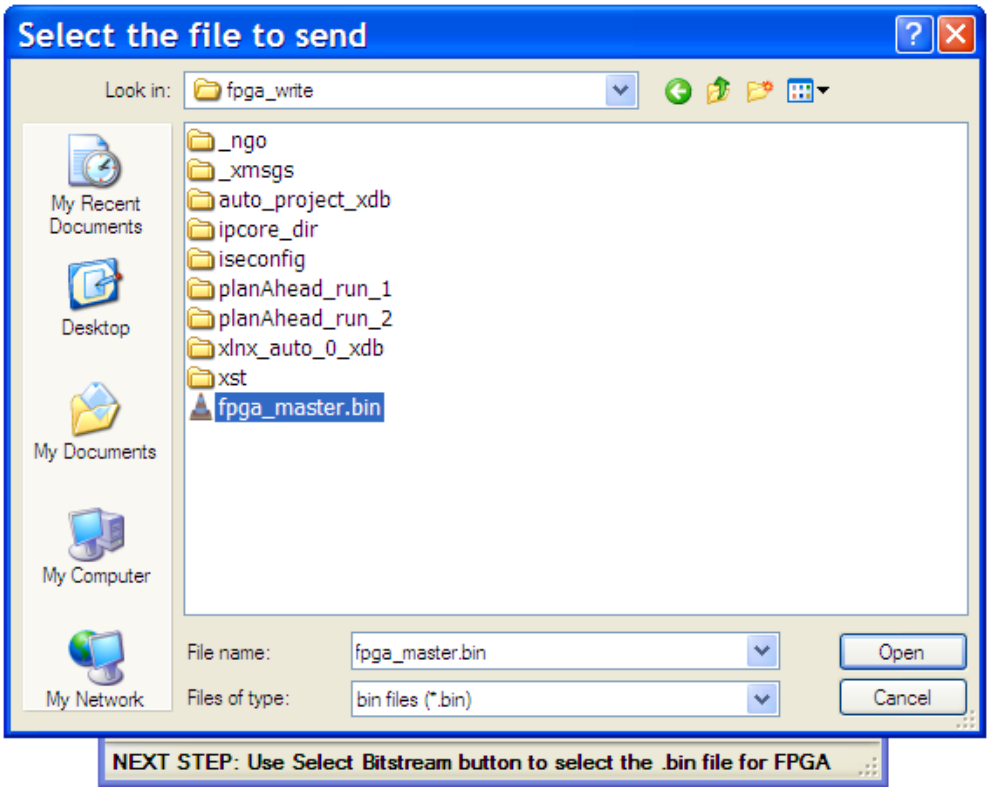
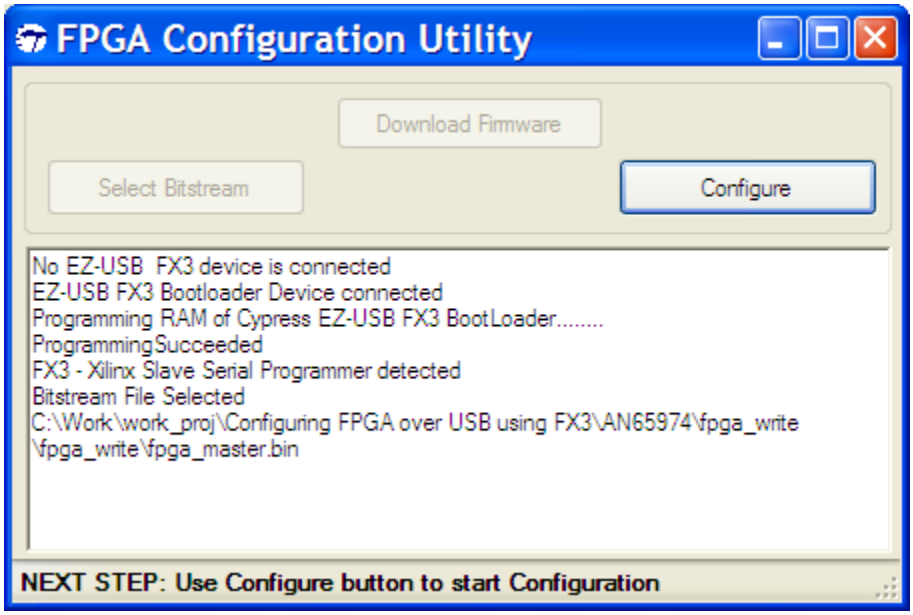


Figure 13. FPGA Configuration Utility after selecting the .bin file



5. Click the "Configure" button to start to configure the Xilinx FPGA. If the FPGA gets configured successfully, then FX3's firmware switches to the Slave FIFO interface. The following status messages appear as part of this step, as Figure 14 shows.

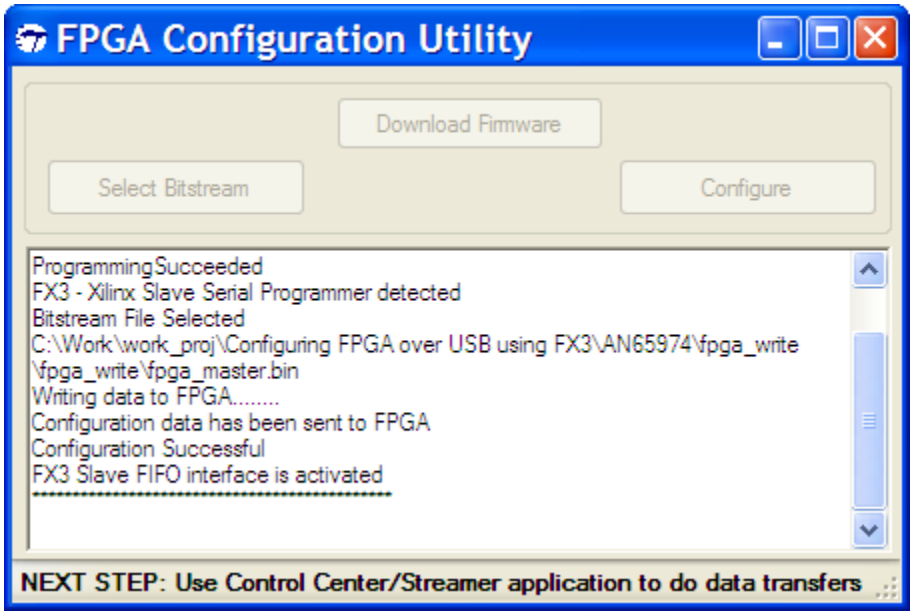
Writing data to FPGA

Configuration data has been sent to FPGA

Configurations Successful

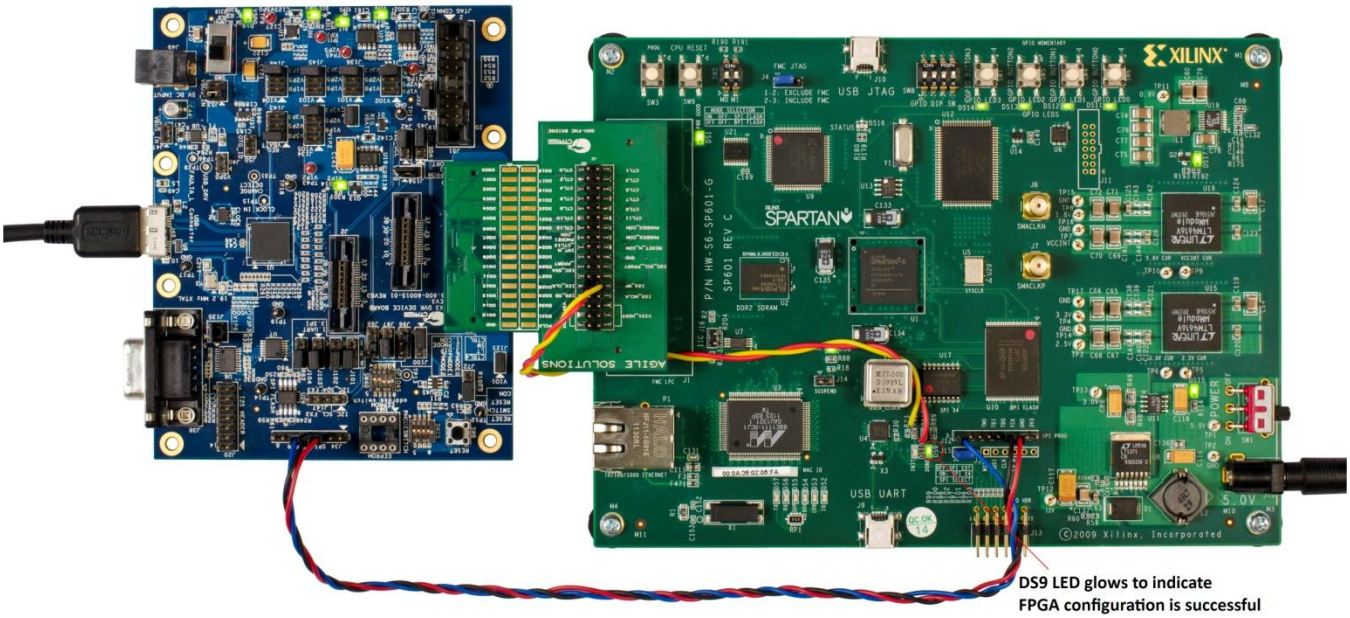
FX3 Slave FIFO interface is activated

Figure 14. FPGA Configuration Utility after FPGA configuration is done successfully



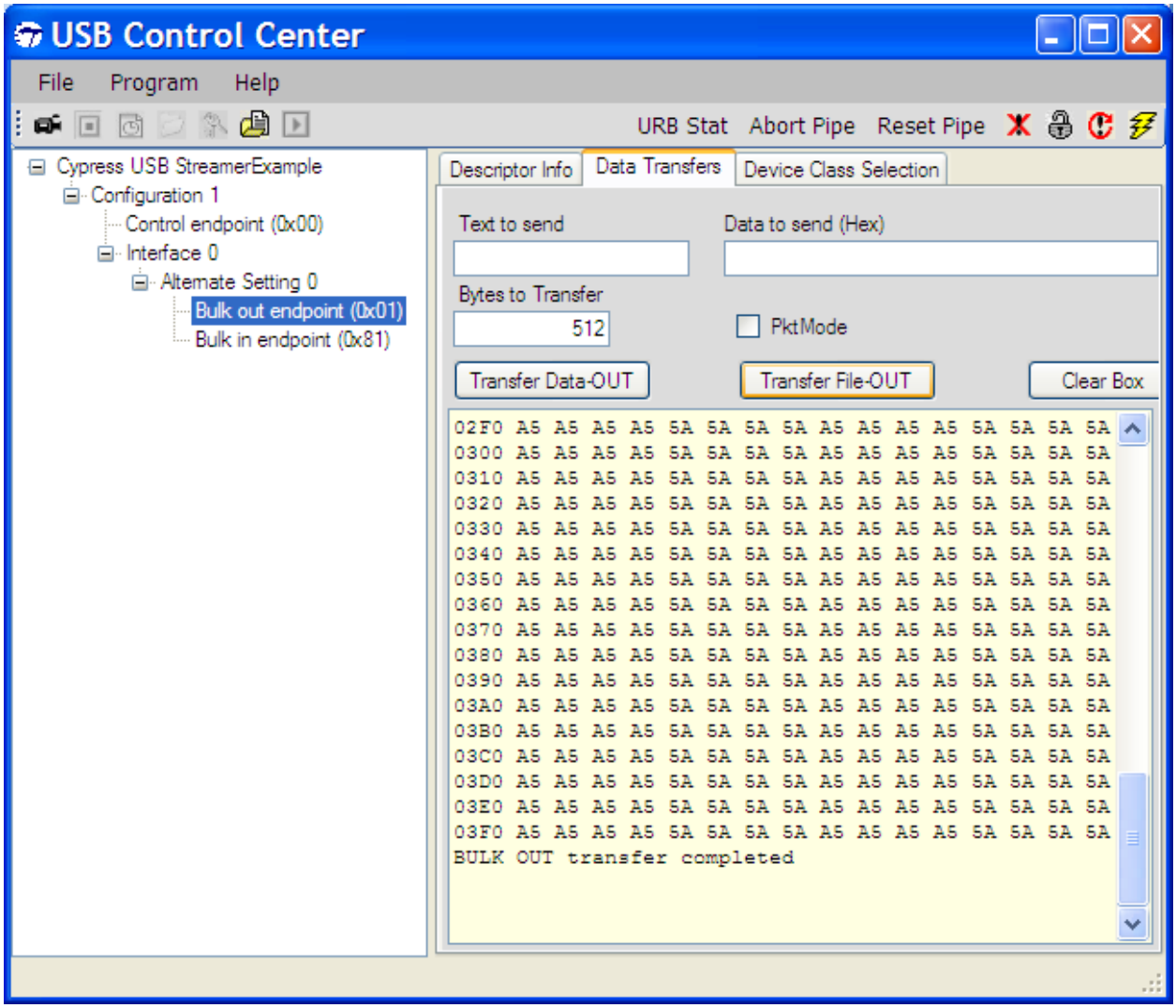
You can observe the DS9 LED glowing on the Xilinx FPGA board after the configuration is successful. It does not glow if something goes wrong during configuration. Figure 15 shows the DS9 LED glowing.

Figure 15. Hardware setup after FPGA is configured successfully



6. Use the Control Center application to verify the loop-back operation between FX3 and the Xilinx FPGA. Go to Bulk out endpoint (0x01) and click the “Transfer File-OUT” button to transfer a file called “TEST.txt”, which is located in the same folder. Then you can see the series of “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

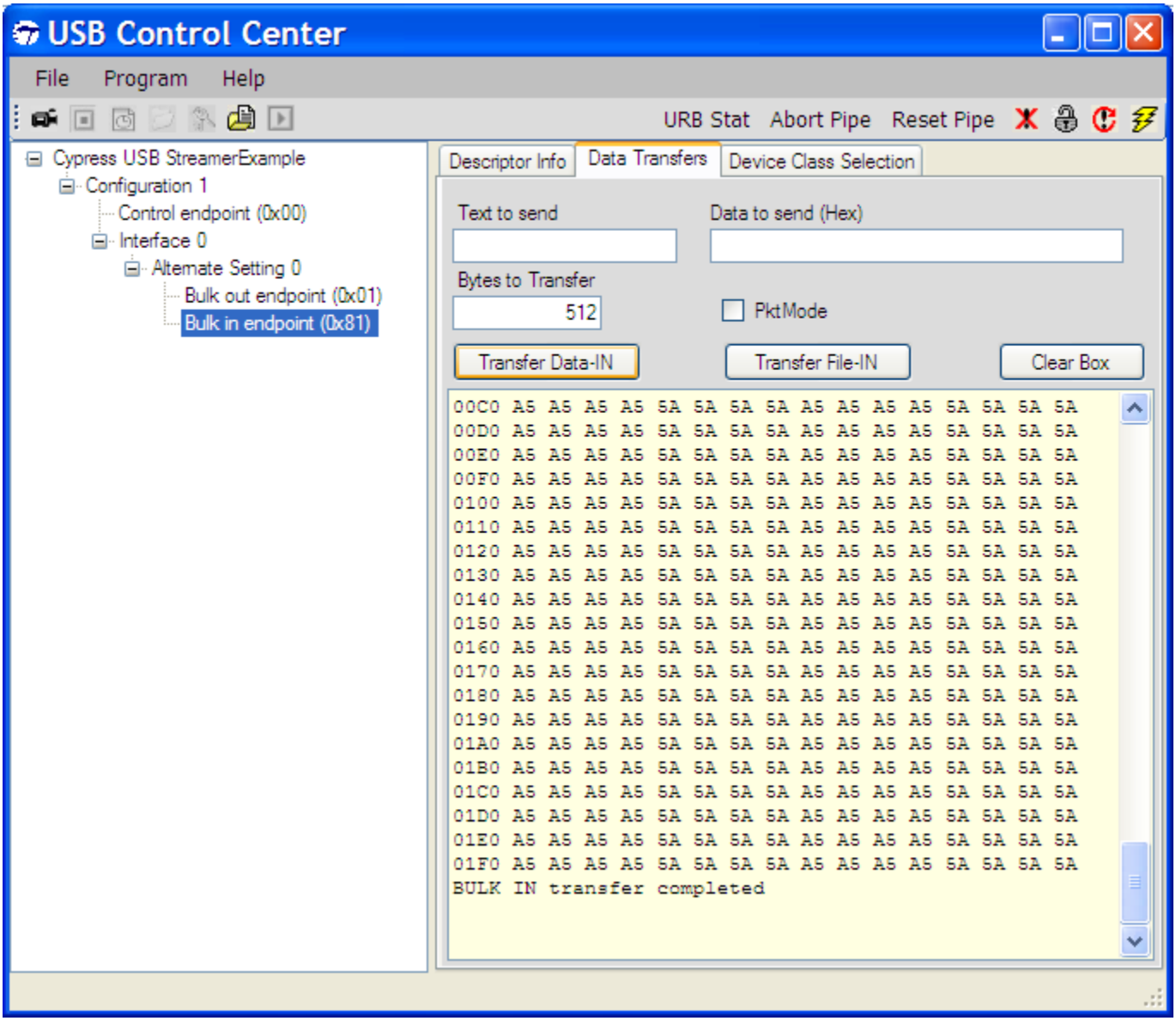


7. Now, Select “Bulk in endpoint” and click “Transfer Data IN”. Then you can observe that the received data is same as the data that has been transmitted to Bulk out endpoint, as Figure 17 shows.

The Data path is as follows:

Control Center → Bulk out endpoint of FX3 → FPGA read the data from Bulk out endpoint → FPGA written the same data to Bulk in endpoint of FX3 → Control Center

Figure 17. USB Control Center after doing Data-IN to get data from Bulk in endpoint



Summary

This application note has 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.

Associated Project Files

Table 3 describes the files attached to this application note.

Table 3. Description of the files in the attachment to this application note

Folder name	Description
FPGA Configuration Utility	Source code of the PC side application
FX3 firmware	Source code of the FX3's 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's firmware Template.exe – executable file of FPGA Configuration Utility

References

- CYUSB3014 datasheet
- Spartan-6 Generation Configuration User Guide – Xilinx UG380
- Spartan-6 FPGA SP601 Evaluation kit
- Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode

About the Author

Name:	Rama Sai Krishna
Title:	Application Engineer Senior
Contact:	rskv@cypress.com

Document History

Document Title: Configuring an FPGA over USB using Cypress EZ-USB® FX3™

Document Number: 001-84868

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3883306	RSKV	1/25/2013	New application note.

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