

Designing with EZ-USB® FX2LP™ Slave FIFO Interface

Author: Rama Sai Krishna V

Associated Project: Yes

Associated Part Family: [CY7C68013A/CY7C68014A/CY7C68015A](#)

Software Version: None

Related Application Notes: [AN65209](#), [AN63620](#)

More code examples? We heard you.

For a consolidated list of USB HighSpeed Code Examples, visit [here](#).

AN61345 provides a sample project to interface an FX2LP™ with FPGA using Slave FIFO interface. The interface, described in the sample implementation, adds High-Speed USB connectivity to applications such as data acquisition, industrial control and monitoring, and image processing. The project provided with this application note is implemented and tested with Xilinx® Spartan® 6 FPGA.

Contents

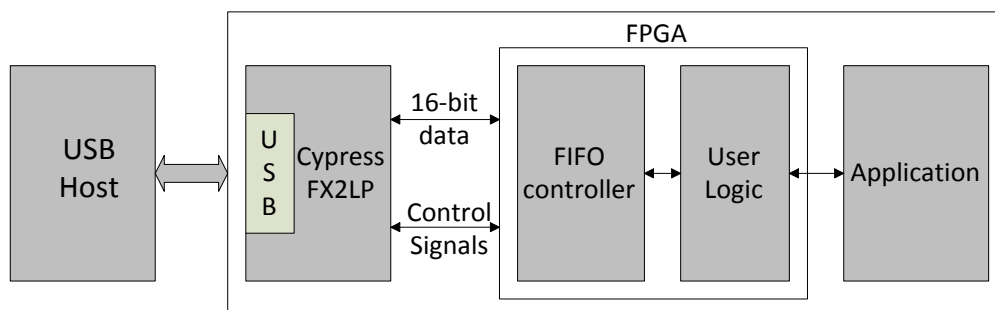
1	Introduction.....	1	6	Design Example	7
2	Hardware Connections	2	6.1	ZTEX Hardware Setup.....	7
3	Firmware Implementation	3	6.2	Firmware and Software Components.....	10
3.1	FX2LP Code Architecture	3	6.3	Operating Procedure.....	13
4	FPGA Code Architecture	4	6.4	Throughput Measurement.....	20
4.1	Data Loopback.....	4	7	Associated Project Files	21
4.2	Stream IN Transfers.....	4	8	How to Port This Design to Work With Altera® FPGA	21
4.3	Stream OUT Transfers.....	5	9	Summary	21
5	Simulation Waveforms.....	6			

1 Introduction

The Cypress EZ-USB FX2LP is a flexible USB 2.0 peripheral controller designed to handle maximum USB 2.0 bandwidth. To take full advantage of the USB 2.0 480 Megabits per second signaling rate, FX2LP contains specialized hardware to buffer the USB data and connect seamlessly to a variety of high-bandwidth external devices such as MCUs, ASICs, and FPGAs.

An FX2LP-FPGA interface is implemented to add High-Speed USB connectivity for FPGA based applications, such as data acquisition, industrial control and monitoring, and image processing. The FX2LP functions in synchronous Slave FIFO mode and the FPGA acts as the master. This application note also provides a sample FX2LP firmware for Slave FIFO implementation and a sample VHDL and Verilog project for FPGA implementation.

Figure 1. FX2LP-FPGA System



The FX2LP can be interfaced with FPGA over two different modes. They are general programmable interface (GPIF) mode and Slave FIFO mode.

- **GPIF Mode:** In this mode, FX2LP acts as a master to an external system and generates all the necessary control signals to read and write data from the external system. GPIF mode is usually preferred when the external system is not intelligent enough to act as a master to FX2LP (for example, USB camera application where image sensor is interfaced to FX2LP). In this case, most of the complexity of the interface implementation resides in the FX2LP firmware.
- **Slave FIFO Mode:** In this mode, the external system interfaced to FX2LP is intelligent enough to generate the necessary read and write control signals, and it can act as a master to FX2LP. Here in this application note, FX2LP is configured to operate in the Slave FIFO mode.

This application note describes the implementation of synchronous 16-bit Slave FIFO on FX2LP, and includes Verilog and VHDL sample projects that show how to interface an external FPGA to FX2LP's Slave FIFO interface.

Note: Sample projects are implemented and tested on Xilinx Spartan 6 FPGAs. But codes provided with this application note are standard Verilog/VHDL codes. Hence, you can use these files as a reference for implementation on any FPGA. You need to select the correct FPGA device while synthesizing and implementing the project.

It is assumed that you are familiar with the Slave FIFO interface, Verilog/VHDL coding, FPGA synthesis and implementation tools. Please refer to chapter 9 (Slave FIFOs) of the [EZ-USB Technical Reference Manual](#).

2 Hardware Connections

The following figure illustrates the hardware connections required for interfacing the FX2LP to the FPGA.

Figure 2. Hardware Connections Diagram

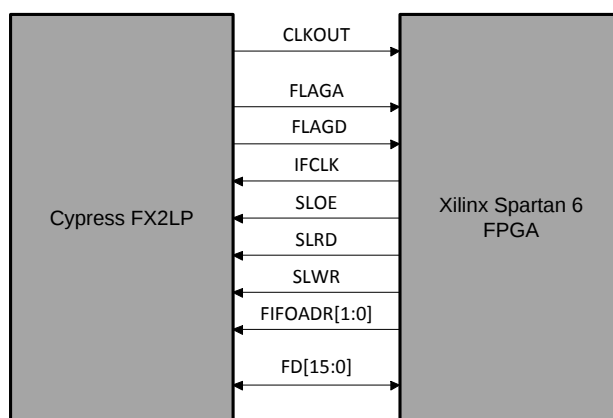


Table 1 describes the Slave FIFO interface signals as shown in Figure 2.

Table 1. Interface signals between FX2LP and FPGA

Pin Name	Description
SLRD	SLRD pin should be asserted by the master to read data from the FIFO.
SLWR	SLWR pin should be asserted by the master to write data to the FIFO.
SLOE	This is the enable signal for the FIFO's output driver.
FIFOADR[1:0]	These signals select the active endpoint.
FD[15:0]	16-bit data bus.
FLAGA/FLAGB/ FLAGC/FLAGD	These flags are used by the FIFO to indicate status (Full, Empty, and Programmable).
IFCLK	The clock for the synchronous Slave FIFO interface. In the design attached to this application note, this clock is configured to 48 MHz and it is generated by the FPGA interfaced to FX2LP.
CLKOUT	FX2LP has a CLKOUT pin which can supply 12-, 24-, or 48-MHz clock.

3 Firmware Implementation

The FX2LP firmware was developed using Keil uVision 2.0 IDE, the evaluation version of the IDE is present in the FX2LP DVK ([CY3684](#)) contents. This section describes the configuration required for implementing the Slave FIFO interface on the slave (FX2LP) and the master side (FPGA).

3.1 FX2LP Code Architecture

The firmware configures auto mode for both the IN and OUT endpoint FIFOs. This means that the packets are committed automatically from the external peripheral to the USB domain for IN transfers and vice versa for OUT transfers. The 8051 CPU is not involved in committing packets. Refer Slave FIFOs chapter in [EZ-USB Technical Reference Manual](#) to get more details on configuration of endpoint FIFOs in auto or manual mode. As bulk transfers are being used in this application, you need to configure the endpoints as Bulk. But based on the end application you can configure endpoint type as Interrupt, Control, or Isochronous in the USB descriptor file.

Because the slave works in AUTO mode, no code is required for data transfer to and from the master, except for the initialization of the following registers (shown in [Table 2](#)).

Table 2. Slave FIFO Configuration Registers

Register Name	Register Description
IFCONFIG	Configure the IFCONFIG register to place FX2LP in Slave FIFO mode
PINFLAGSAB/ PINFLAGSCD	Configure the FIFO Flags. FIFO Flags can be configured either to act in fixed mode or indexed mode. In indexed mode, FLAGA, FLAGB, and FLAGC are automatically configured as the Empty, Full, and Programmable flags respectively, for whichever endpoint the FIFOADR [1:0] lines are pointing at a particular instant. In fixed mode, each of the four flags can be configured to act as Empty, Full, or Programmable for any of the endpoints by writing into the programmable flag registers. Here in this design, FLAGA is configured as empty flag for EP2 OUT FIFO and FLAGD is configured as full flag for EP6 IN FIFO.
EP2CFG/ EP6CFG	Configure EP2 as OUT, 512 bytes, quad-buffered endpoint and EP6 as IN, 512 bytes, quad-buffered endpoint.
EP2FIFOCFG/ EP6FIFOCFG	Configure the FIFO as byte wide and operating in AUTO mode
EP6AUTOINLENH/ EP6AUTOINLENL	When the number of bytes in the EP6 FIFO becomes equal to the value specified by these registers, the packet gets auto-committed. The value specified should be less than or equal to the maximum packet size specified in the endpoint descriptor.

4 FPGA Code Architecture

The main function of the FPGA code is to monitor the Full and Empty flags of the Slave FIFO, and then read and write into the FIFOs accordingly.

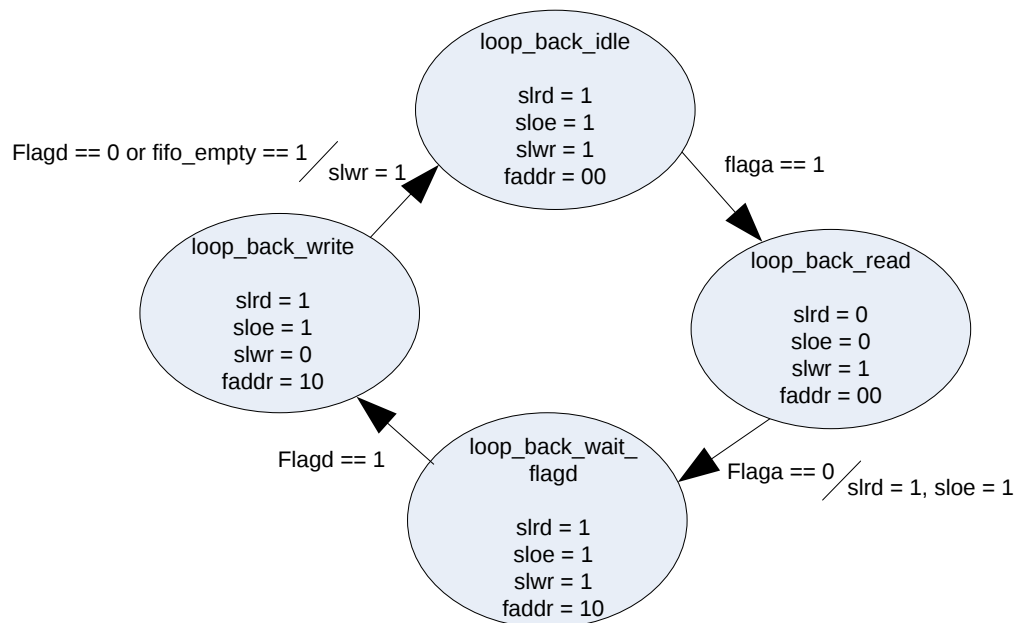
The interface clock (IFCLK) coming from the FPGA is shifted by 180 degrees to meet the setup time requirements of the Slave FIFO interface of FX2LP.

Both Verilog and VHDL projects are implemented to show how to configure an FPGA to act as a master to an FX2LP Slave FIFO. [Xilinx ISE Design Suite](#) is used for code development.

4.1 Data Loopback

Any data written into the EP2 FIFO of FX2LP is read by the FPGA, and written into the EP6 FIFO. The associated project folder, *Loopback* (attached with this application note) contains this project.

Figure 3. Loop Back State Diagram

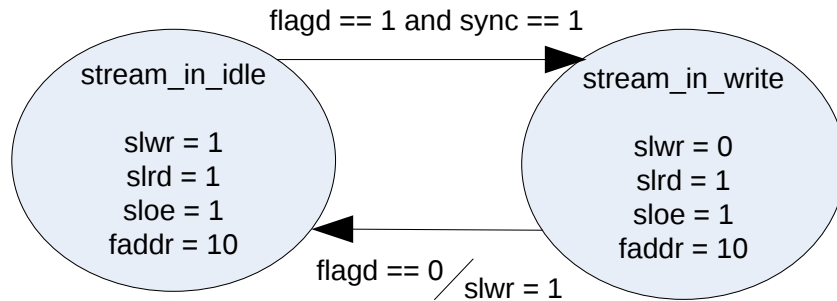


State **loop_back_idle** is the idle state in which the SLRD, SLOE, and SLWR signals are high (de-asserted). The state machine moves to **loop_back_read** state when EP2 FIFO Empty flag (FLAGA) becomes High. In **loop_back_read**, FPGA reads data by asserting both SLRD and SLOE. State machine moves to **loop_back_wait_flagd** state when EP2 FIFO Empty flag becomes Low. During this transition SLRD and SLOE signals are de-asserted. In **loop_back_wait_flagd** state, FIFO address lines are driven to address EP6. It stays in this state till EP6 Full flag (FLAGD) is Low. State machine moves to **loop_back_write** state when FLAGD becomes high. In this state, FPGA writes same data to EP6 FIFO by asserting SLWR signal.

4.2 Stream IN Transfers

The FPGA monitors the Full flag of EP6 (FLAGD) and Sync (PC0 of FX2LP) signal. FPGA continuously writes incrementing data into the FIFO when both FLAGD and Sync signals are high. While writing data into the EP6 FIFO, the FPGA pauses the writing as soon as the Full flag gets asserted, and resumes the writing when the flag gets de-asserted. The associated project folder, *Stream IN* contains this project.

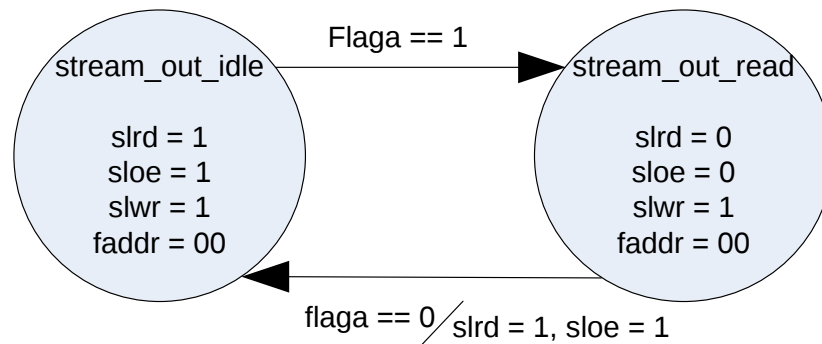
Figure 4. Data Stream IN State Diagram



There are two distinct states in this state machine: `stream_in_idle` and `stream_in_write`. `stream_in_idle` is the idle state, where `SLWR` stays high. As long as the EP6 Full flag (`FLAGD`) stays low (asserted), the state machine remains in `stream_in_idle`. After the Full flag and Sync (`PC0` of `FX2LP`) signal goes high, there is a transition from state `stream_in_idle` to `stream_in_write`. In state `stream_in_write`, FPGA continuously writes incremental data to EP6 FIFO. State machine goes back to `stream_in_idle` state when `FLAGD` becomes Low. `SLWR` signal is de-asserted during this transition.

4.3 Stream OUT Transfers

Figure 5. Stream OUT State Diagram



There are two distinct states in this state machine: `stream_out_idle` and `stream_out_read`. `stream_out_idle` is the idle state, where `SLRD` and `SLOE` stays high. As long as the EP2 Empty flag (`FLAGA`) stays low (asserted), the state machine remains in `stream_out_idle`. After the EP2 Empty flag goes high, there is a transition from state `stream_out_idle` to `stream_out_read`. In state `stream_out_read`, FPGA continuously reads data from EP2 FIFO. State machine goes back to `stream_in_idle` state when `FLAGA` becomes Low. `SLRD` and `SLOE` signals are de-asserted during this transition. The associated project folder, *Stream OUT* contains this project.

5 Simulation Waveforms

This section shows you the simulation waveforms of Slave FIFO interface signals under different modes (Stream IN and Stream OUT). These waveforms are captured using Xilinx ISim tool.

Figure 6. Data Stream IN – Waveform Front Beginning of Burst Write

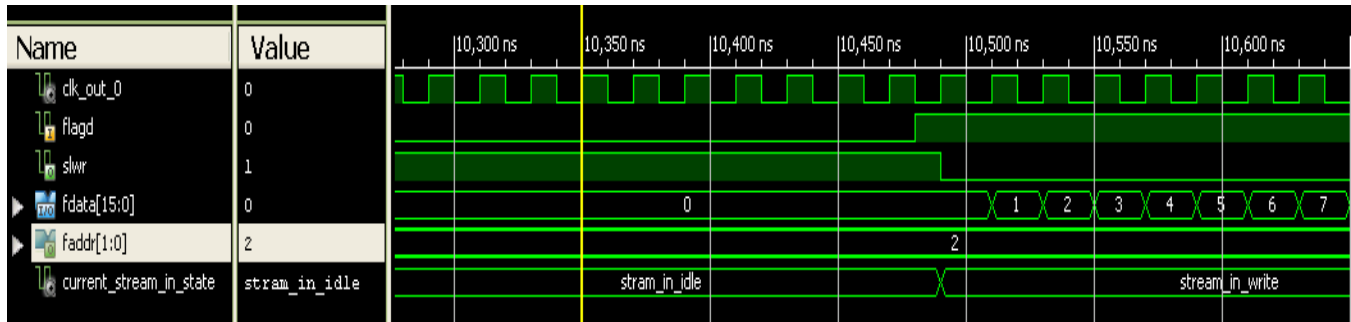
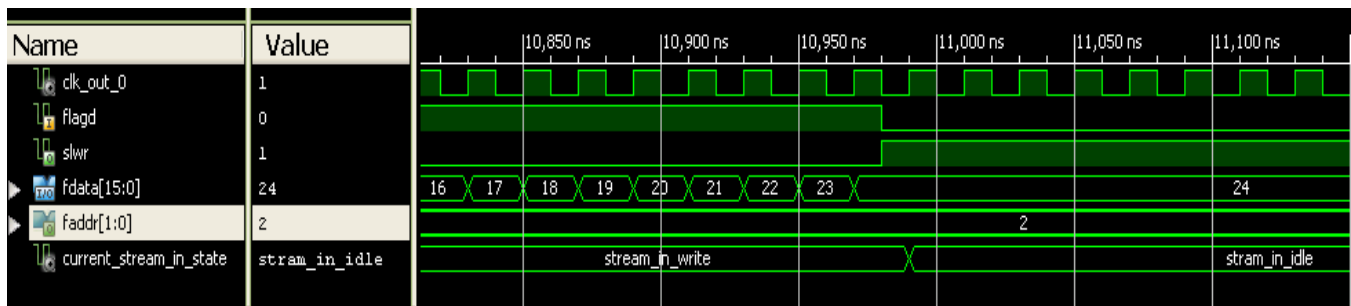


Figure 6 shows the SLWR assertion when FLAGD, or the Full flag is de-asserted.

Figure 7. Data Stream IN End of Burst Write



After the Full flag gets asserted, SLWR gets de-asserted as shown in Figure 7.

Figure 8. Data Stream OUT Beginning of Burst Read

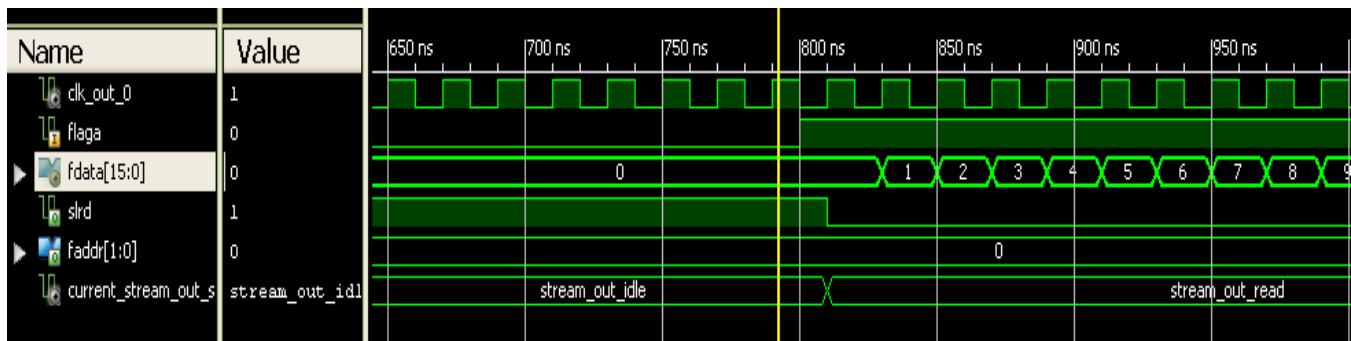
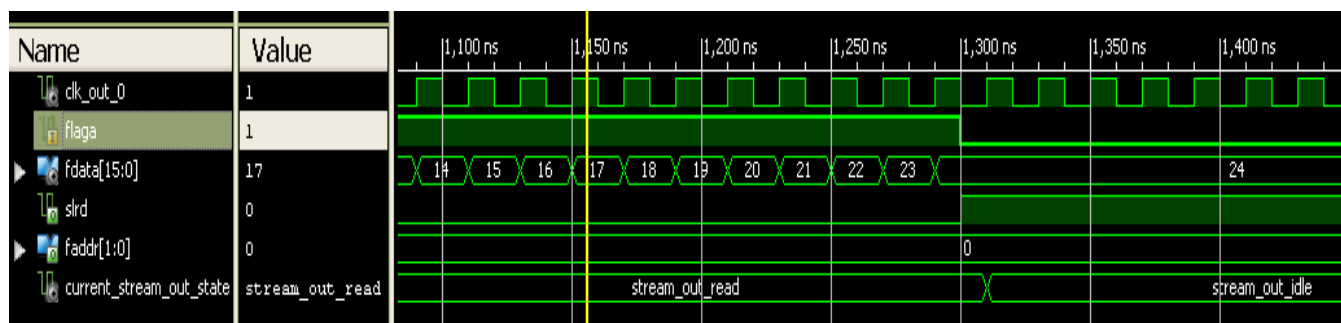


Figure 8 shows the SLRD assertion when FLAGA (EP2 EF) is de-asserted.

Figure 9. Data Stream OUT End of Burst Read



As soon as the FLAGA is asserted, the SLRD signal is deasserted as shown in [Figure 9](#).

6 Design Example

This section provides a complete design example in which, a Xilinx Spartan 6 FPGA is connected to FX2LP over the synchronous Slave FIFO interface.

A Spartan-6 compatible example is provided along with this application note. The example implements stream IN, stream OUT and loopback transfers. The behavior of the FPGA bit files are explained in the section [FPGA Code Architecture](#).

The hardware, firmware, and software components that were used to implement this design are discussed in the following sections:

6.1 ZTEX Hardware Setup

The [ZTEX FX2LP - FPGA Module1.11](#) (shown in [Figure 11](#)) is used along with [Experimental Board 1.3](#) (shown in [Figure 12](#)). The experimental board needs a power supply and a JTAG cable (to configure the FPGA). CON1 (on [Experimental Board 1.3](#)) is a standard DC power jack with 2.1-mm center pin (+) diameter and 5.5-mm barrel (-) diameter for a supply voltage of 4.5 V to 16 V. CON9 (on [Experimental Board 1.3](#)) is a 14-pin, 2.0-mm pitch JTAG connector standardized by Xilinx. The Polarization Key(hole) named '1' is present on both the module and the experimental board. [ZTEX FX2LP - FPGA Module1.11](#) has to be mounted on [Experimental Board 1.3](#) such that the polarization, on both, lie on the same corner. To identify the polarization holes on both the boards, refer to the layout diagrams available at the above links.

Mount the FPGA Module 1.11 on [Experimental Board 1.3](#) as shown in [Figure 13](#). Power up the board with 5V or 12V power supply and connect to a host PC using a mini USB cable.

[Platform USB Cable II](#) (JTAG adapter) is used to configure the Xilinx Spartan-6 FPGA present on the [ZTEX FX2LP - FPGA Module1.11](#). 'Chipscope Pro' (described in the following step) is the software compatible with this JTAG adapter.

The hardware connection diagram is shown in [Figure 10](#):

Figure 10. Hardware Connections (Ztex board)

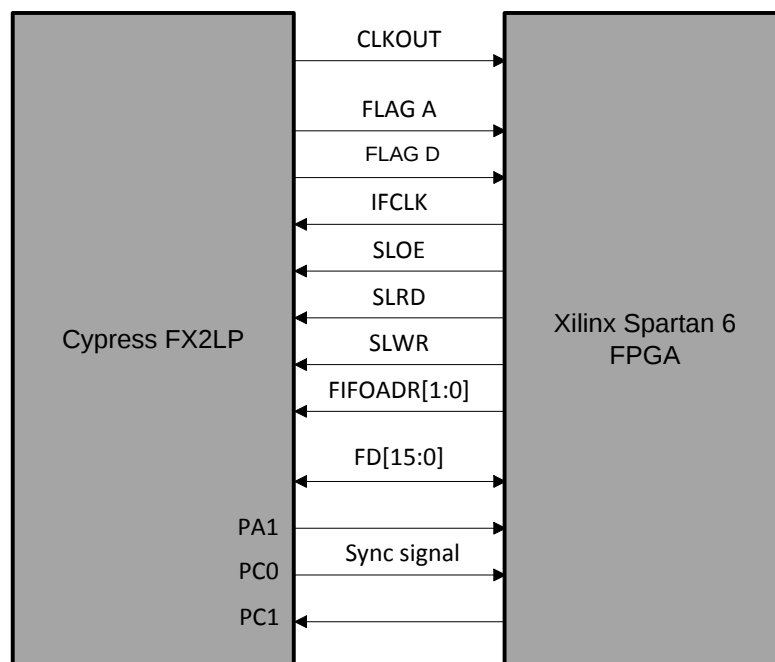


Figure 11. Ztex FX2LP – FPGA module 1.11



Figure 12. Ztex Experimental Board 1.3

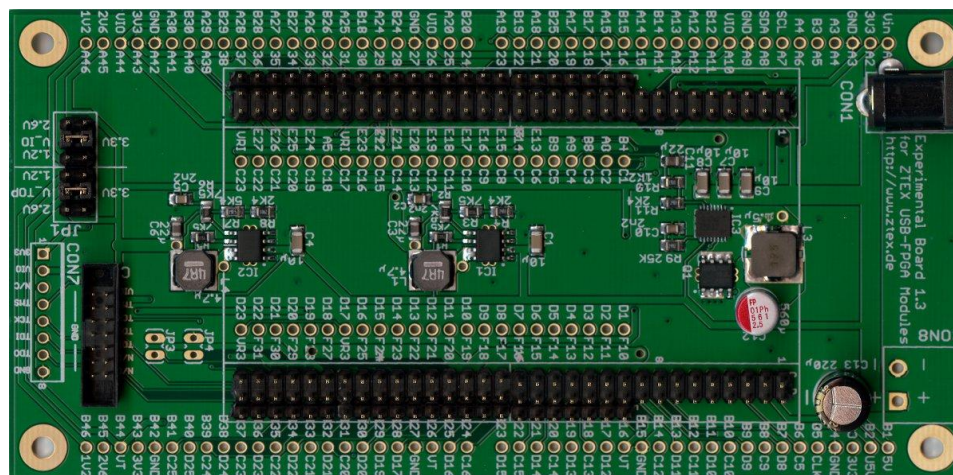
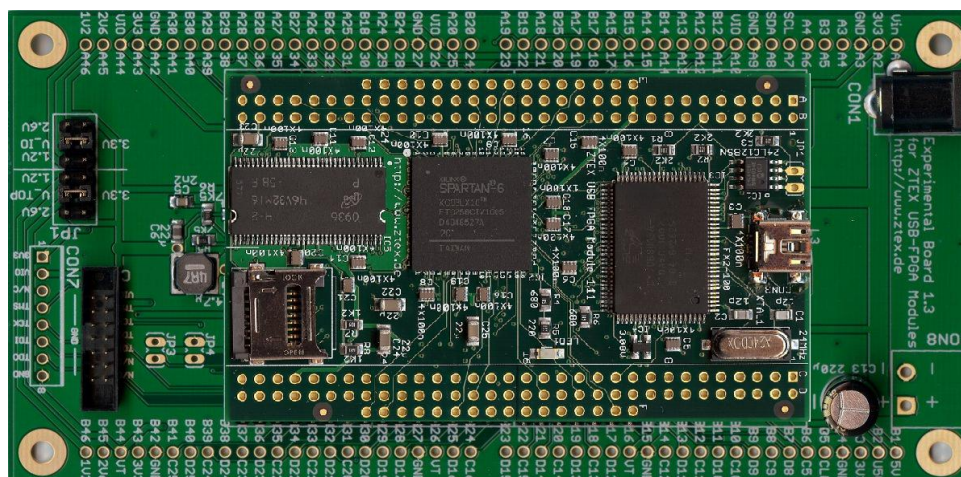


Figure 13. Ztex FX2LP-FPGA Module on Top of Experimental Board

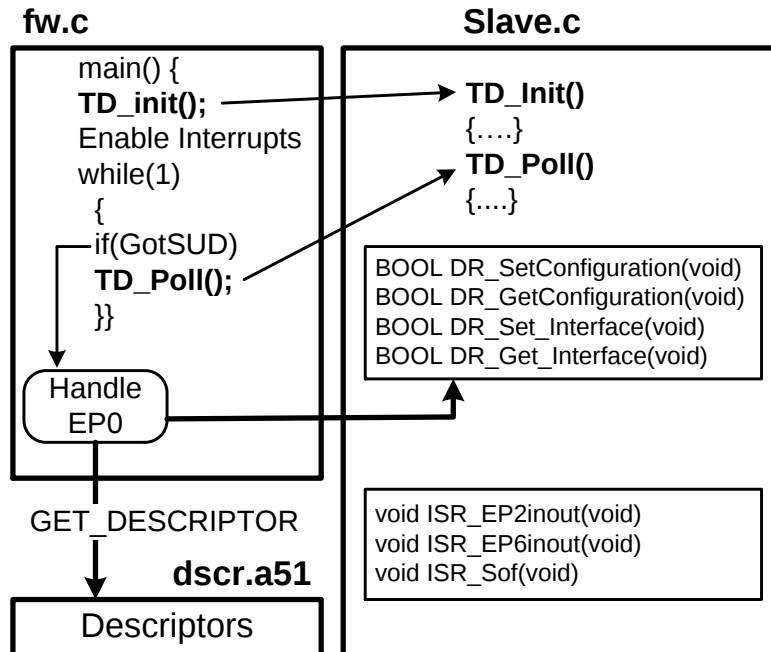


6.2 Firmware and Software Components

6.2.1 FX2LP Firmware

Figure 14 shows how the FX2LP firmware modules fit together.

Figure 14. FX2LP Firmware for Slave FIFO Interface



The *Fw.c* file contains the **main** function. It performs much of the USB maintenance such as enumeration, and calls specifically-named external functions in the application code (*Slave.c*) whenever customization is required. The *Fw.c* file mostly does not require your modification. After performing various housekeeping steps, it calls an external function called `TD_init`, which you provide in *Slave.c*. (The prefix `TD` stands for “Task Dispatcher”.) Then it enters an endless loop that checks for arrival of SETUP packets over CONTROL endpoint 0. The loop also checks for the USB suspend event, but this is not used by the Slave FIFO application. Every time through the loop, it calls the external `TD_Poll` function which you provide in the *Slave.c* file. In this application, the `TD_Poll` function takes care of the synchronization of data transfers between FPGA and the FX2LP. This function does nothing once the data transfers are started as endpoint FIFOs are configured in auto mode.

Every USB peripheral receives two types of requests over its CONTROL endpoint: enumeration and operational.

Enumeration

When a USB device is attached, the host PC sends multiple `GET_DESCRIPTOR` requests to discover the device type and its requirements as part of a process called enumeration. The *fw.c* code intercepts these requests and handles them using the values stored in the *dscr.a51* file.

An advantage of using USB Frameworks is that the code has been tested and verified to pass USB “Chapter 9” requirements. Chapter 9 refers to the chapter in the USB Specification that deals with device requests (over EP0) and their proper responses.

Operational

Wherever user code is needed, *fw.c* calls a specifically-named external function with the `DR` prefix (Device Request) that you provide in the *Slave.c* file. For a simple application like Slave FIFO, there is only one configuration and one interface, so the two `DR_Set-Get` function pairs in Figure 14 simply store the Set values sent by the host and echo them back when the host issues the Get requests. For more complex configurations, you can use these `DR` calls (“hooks”) to do things such as changing camera resolutions or routing requests to two different interfaces.

The remainder of this section describes the three portions of this file that require user code to implement the Slave FIFO application.

TD_Init

This function does the following:

- Sets the 8051 clock to 48 MHz.
- Configures the Slave FIFO interface to use internal 48-MHz clock.

```
IFCONFIG = 0xE3; //Internal clock, 48 MHz, Slave FIFO interface
SYNCDELAY;
```

- Configures EP2 as a BULK-OUT endpoint and EP6 as a BULK-IN endpoint. Both are quad-buffered and use 512-byte FIFOs. EP4 and EP8 are deactivated as they are not used in this design.

```
EP2CFG = 0xA0; //out 512 bytes, 4x, bulk
SYNCDELAY;
EP6CFG = 0xE0; //in 512 bytes, 4x, bulk
SYNCDELAY;
EP4CFG = 0x02; //clear valid bit
SYNCDELAY;
EP8CFG = 0x02; //clear valid bit
SYNCDELAY;
```

- Resets FIFOs.
- Configures endpoint 2 FIFO in auto OUT mode with 16-bit interface and configures endpoint 6 in auto IN mode with 16-bit interface.

```
EP2FIFOCFG = 0x00; // AUTOOUT=0, WORDWIDE=1
// core needs to see AUTOOUT=0 to AUTOOUT=1 switch to arm endpoints
SYNCDELAY;
EP2FIFOCFG = 0x11; // AUTOOUT=1, WORDWIDE=1
SYNCDELAY;
EP6FIFOCFG = 0x0D; // AUTOIN=1, ZEROLENIN=1, WORDWIDE=1
SYNCDELAY;
```

- Configure FIFO flag outputs. FLAGA is configured as empty flag for EP2 OUT FIFO and FLAGD is configured as full flag for EP6 IN FIFO.

```
PINFLASAB = 0x08; // FLAGA - EP2EF
SYNCDELAY;
PINFLAGSCD = 0xE0; // FLAGD - EP6FF
SYNCDELAY;
```

- Sets the PA1 pin (connected to PROG_B pin of FPGA) to high. This is required to [enable the JTAG configuration](#) of the FPGA. If a new FX2LP firmware is being used with the ZTEX hardware setup, make sure to set the PA1 pin of FX2LP high.

```
OEA|=0x02; //Declare PA.1 as output
SYNCDELAY;
IOA|=0x02; //output 1 on PA.1
SYNCDELAY;
```

- Configures PC0 pin as output and PC1 as input. PC0 is used for synchronizing the data transfers between the FPGA and FX2LP. PC1 is used to know the readiness of FPGA to supply the Slave FIFO interface clock (IFCLK).

```
OEC|=0x01; //PC.0 as output (SYNC signal)
SYNCDELAY;
IOC|=0x00; //output 0 on PC.0...SYNC signal is LOW
SYNCDELAY;
```

```
OECD=0xFD; //PC.1 as input (Clock changing signal)
SYNCDelay;
```

TD_Poll

TD_Poll is called in an infinite loop residing in the *fw.c* file (Figure 14). In this Slave FIFO design, **TD_Poll** is used to just change the interface clock (IFCLK) source.

The example projects attached to this application note are designed to take the interface clock (IFCLK) from the FPGA. If you are using the ZTEX hardware board, then the FX2LP needs to be programmed first to set PROG_B pin (which is connected to PA1) of FPGA to HIGH to configure it. But the FX2LP firmware cannot configure the IFCONFIG register to work with external clock before configuring the FPGA to supply the interface clock (Note that the external IFCLK source must be present before the firmware sets IFCONFIG.7 = 0 (IFCLK is provided by external device)). So to meet this condition, initially IFCONFIG register is configured to use internal IFCLK and then IFCONFIG.7 is modified to take IFCLK from the FPGA once FPGA is up and running with the bit-stream. PC1 of FX2LP is used to do this change. PC0 of FX2LP is used as a Sync signal between FX2LP and FPGA. The following code is used to change the clock source from internal to external.

```
if(!(IOC & 0x02))
{
    done_frm_fpga = 1;
}
if((done_frm_fpga) && (IOC & 0x02))
{
    IFCONFIG = 0x03; //external clock input, Slave FIFO interface
    SYNCDelay;

    IOC|=0x01; //output 1 on PC.0...SYNC signal is HIGH
    SYNCDelay;
    done_frm_fpga = 0;
}
```

FX2LP firmware is there as part of an attachment to this application note. Build it and download into FX2LP using the Control Center utility before you download bitstream into the FPGA. These steps are explained in the section [Operating Procedure](#).

6.2.2 Control Center utility

Use the Cypress [Control Center](#) utility to download firmware and perform BULK transfers to FX2LP. The Control Center utility is available by installing the [SuiteUSB](#) development tools available from Cypress.

6.2.3 ChipScope Pro

'[ChipScope Pro](#)' software (provided along with [Xilinx ISE Design Suite 14.1](#)) is used to configure the Xilinx Spartan-6) FPGA. The evaluation version of [Xilinx ISE Design Suite 14.1](#) could be used without license for 30 days. Or any other suitable way of configuring Xilinx Spartan-6 FPGA could be adopted. The steps to configure Xilinx Spartan-6 FPGA using ChipScope Pro are discussed in the following section.

If any other version of 'ChipScope Pro' software is being used, make sure it supports Xilinx Spartan-6 FPGA.

6.3 Operating Procedure

6.3.1 Assigning CYUSB driver

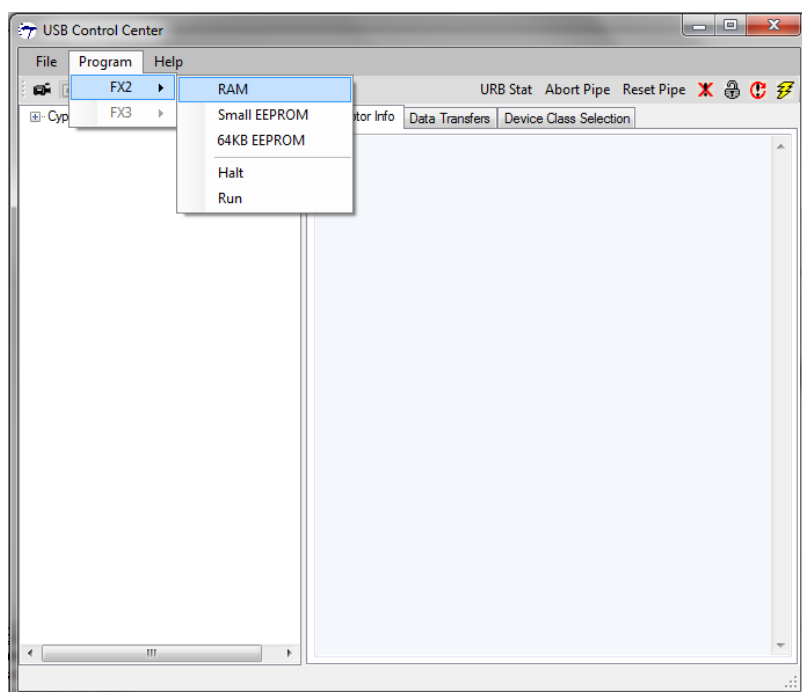
After the ZTEX FPGA module is connected to a host through USB, check the VID and PID of the device. This will be 0x04b4 and 0x8613 if no firmware is flashed in the EEPROM that is on the board. This can be 0x2214 and 0x0100 if the firmware built with the ZTEX SDK is flashed. Make sure to add these VID and PID values in the *CYUSB.inf* file. Assign 'CYUSB.sys' driver to this device. This is required for the device to show up in the Control Center utility.

Please refer to the document *CyUSB.pdf* for more details about assigning *CYUSB.sys* for device with custom VID and PID. When you install the SuiteUSB development tools available from Cypress, *CyUSB.pdf* is available in the path: C:\Cypress\Cypress Suite USB 3.4.7\Driver (may vary with the installation path).

6.3.2 Downloading FX2LP firmware

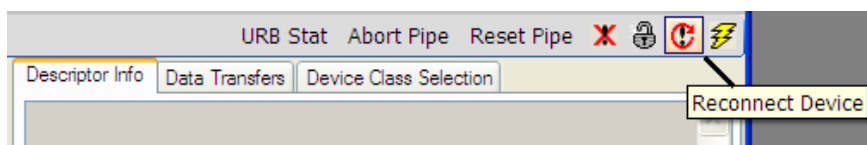
Open the Control Center utility and download the FX2LP firmware (Click **Program>FX2>RAM** and navigate to *slave.hex* file in the associate project folder).

Figure 15. Downloading Firmware Image into FX2LP RAM

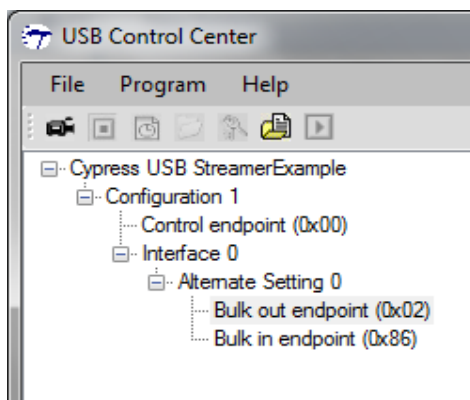


You need to press the button shown in the following figure to reconnect the device after you download *slave.hex*, if the default VID and PID of your device is 0x2214 and 0x0100.

Figure 16. Reconnect Button on Control Center



FX2LP enumerates as shown in [Figure 17](#). Endpoint 2 is configured as Bulk OUT endpoint and Endpoint 6 is configured as Bulk IN endpoint.

Figure 17. FX2LP Device View on Control Center After Downloading *slave.hex*


6.3.3 Downloading FPGA bit-stream

Connect the JTAG cable to the JTAG connector provided on the ZTEX Experimental board 1.3. After the previous step, the PROG_B pin of FPGA (connected to the PA1 pin of FX2LP) will be high, which is required to enable the JTAG configuration of FPGA. Now, configure the FPGA with the Spartan-6 compatible bit-stream (Stream IN or Stream OUT or Loopback bit-streams provided with this application note) as shown in the following figures.

Figure 18. ChipScope Pro

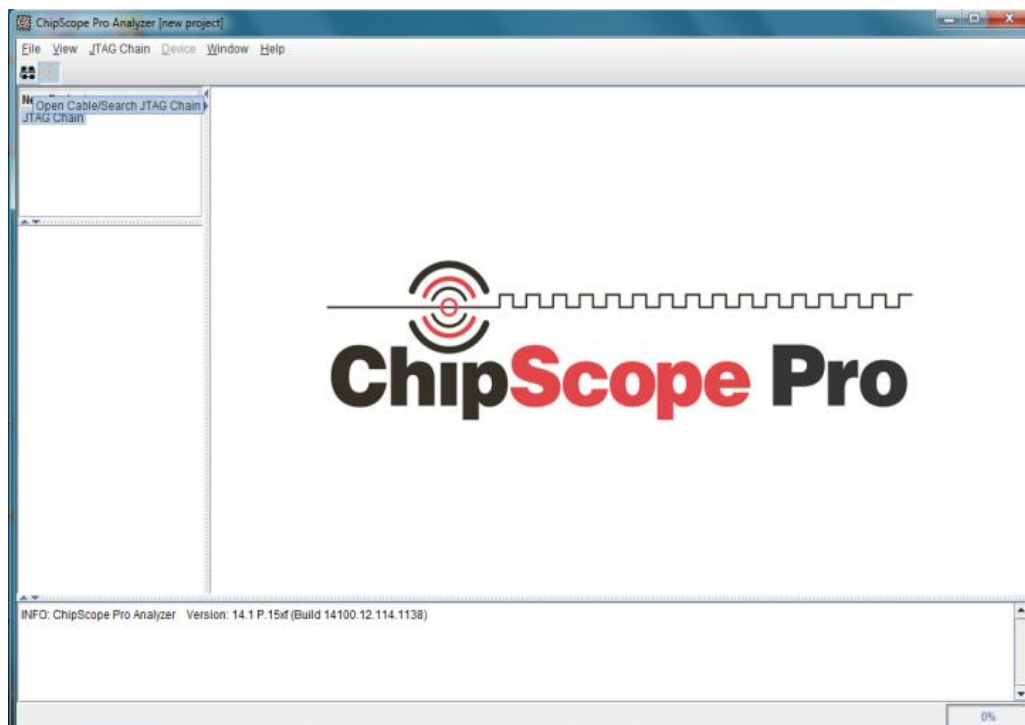


Figure 19. Configuring FPGA Using ChipScope Pro

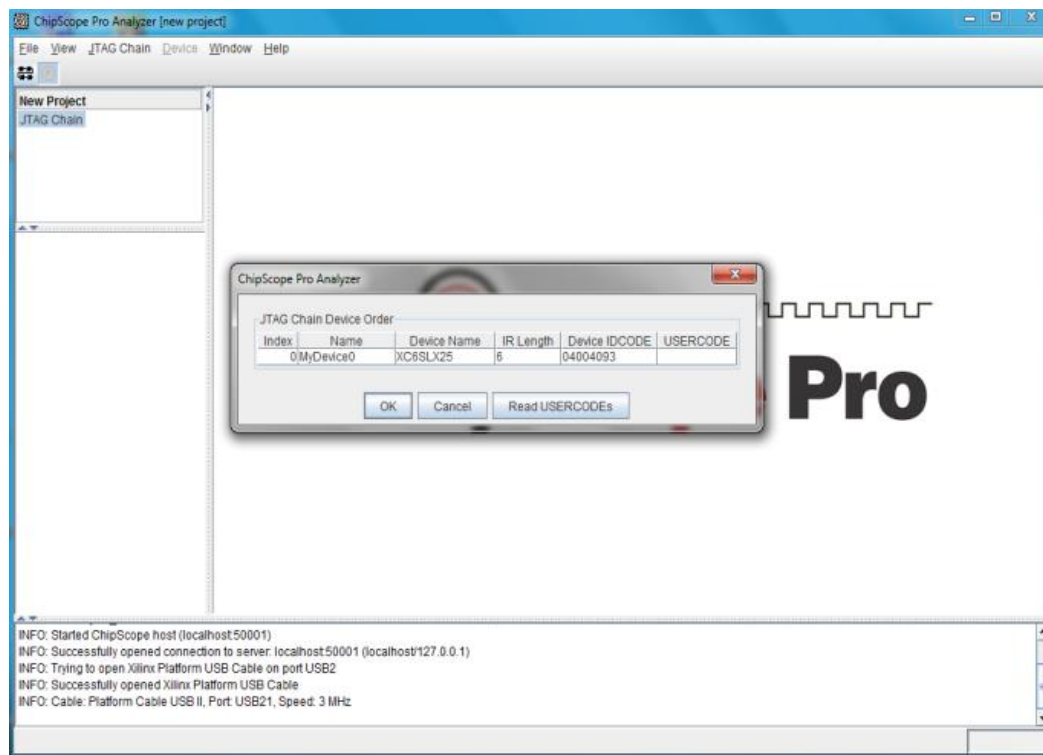


Figure 20. Configuring FPGA Using ChipScope Pro

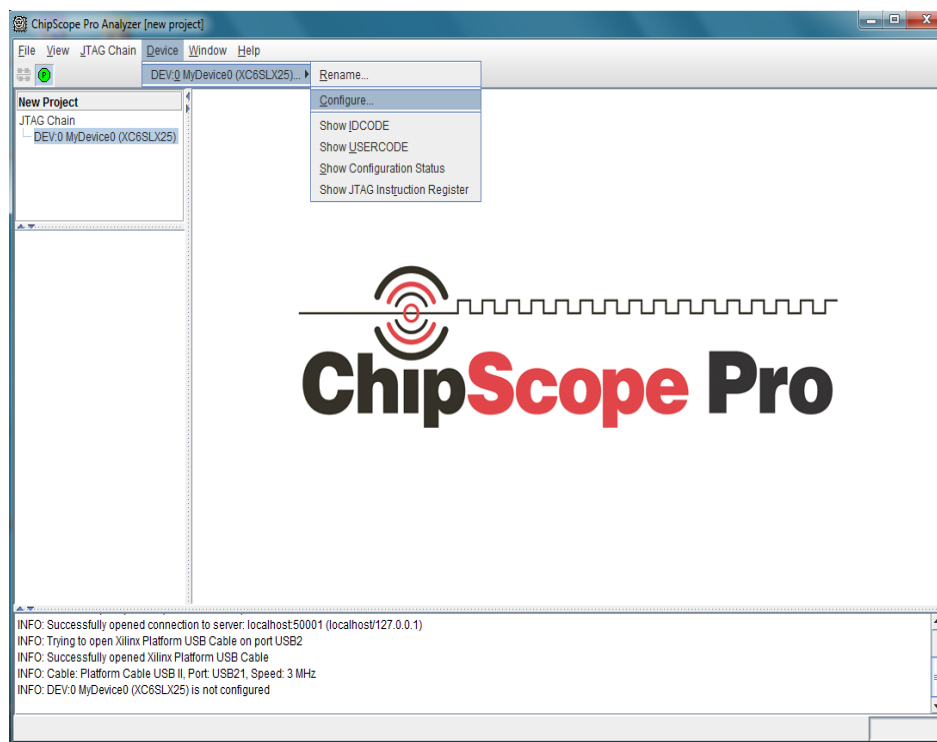
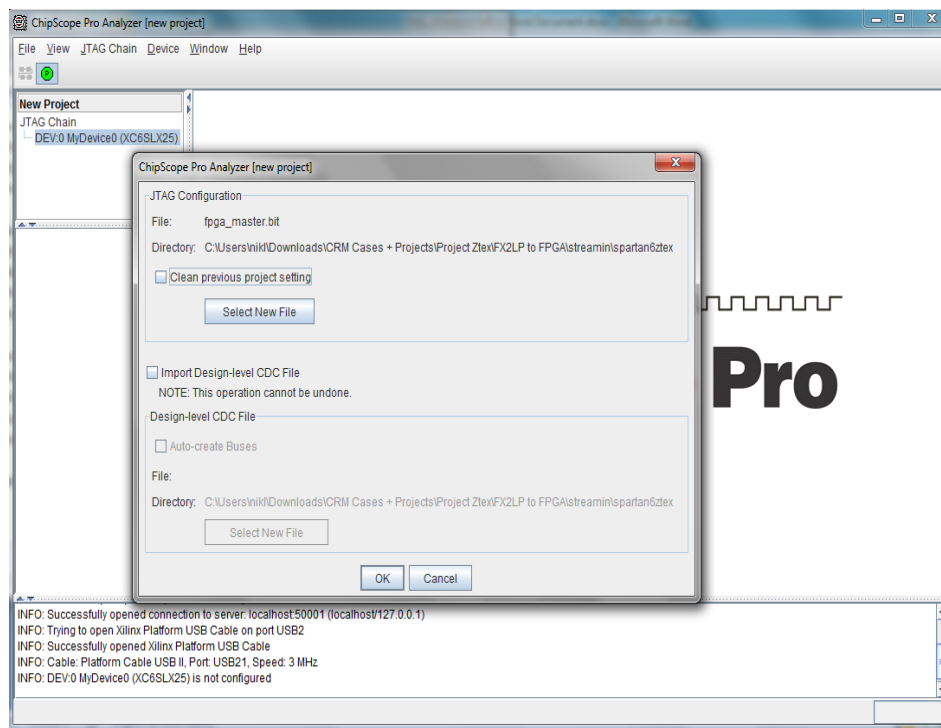


Figure 21. Configuring FPGA Using ChipScope Pro



6.3.4 Verifying the result for Loopback bit-stream

If FPGA is configured with Loopback bit-stream (in step 3), it implements a loopback on endpoints EP2OUT and EP6IN. Data sent from host to EP2 is read by the FPGA, and written to the EP6 endpoint FIFO.

For verifying loopback operation, in the Control Center window choose **Bulk Out Endpoint (0x02)**, then click the **Transfer File-OUT** button and browse *512_count.hex* (provided with the attachment) to perform the data transfer. Now read the EP6IN buffer by clicking **Transfer Data-IN** to verify the data written into it. The steps are shown in the following figures.

Figure 22. Transferring a File to OUT Endpoint 2

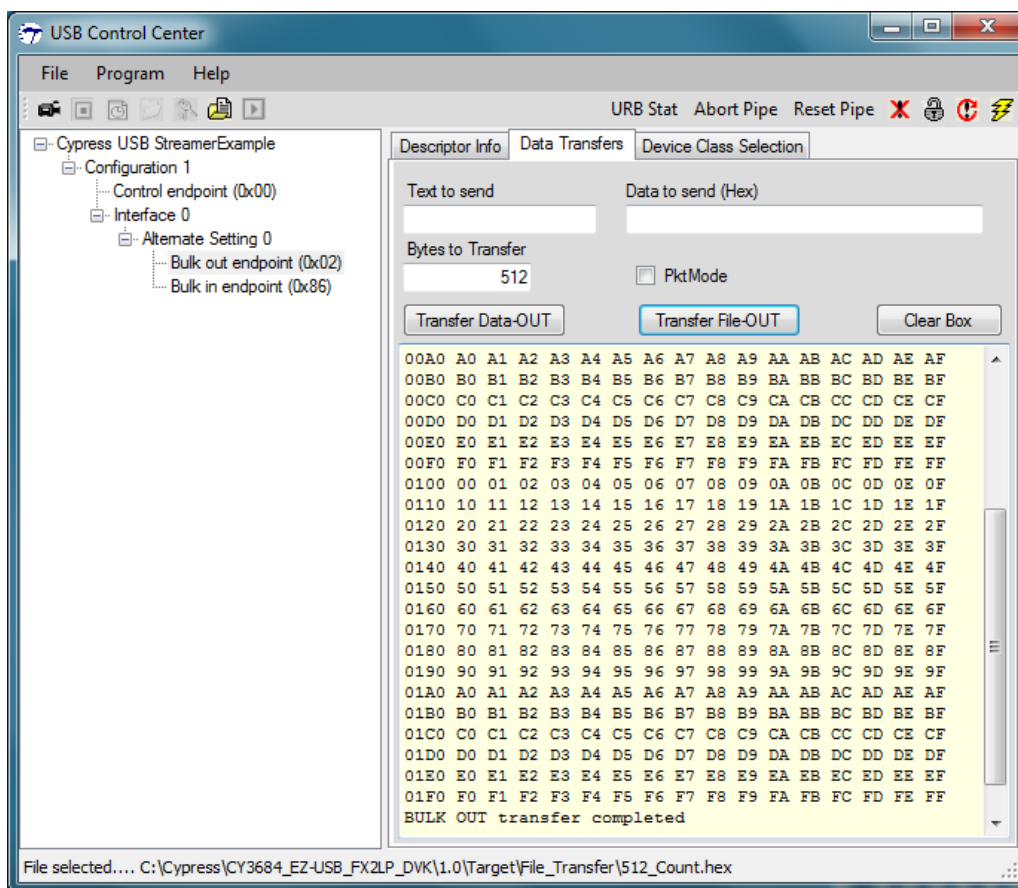
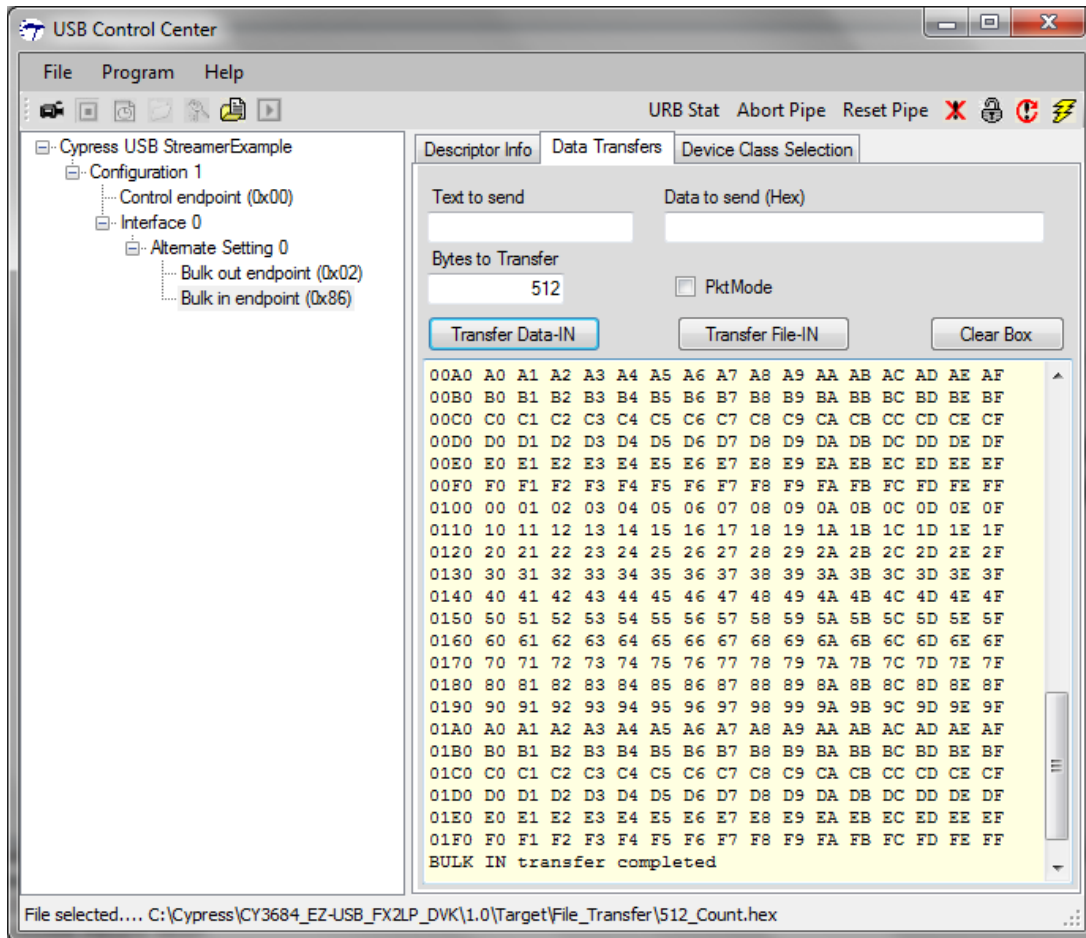


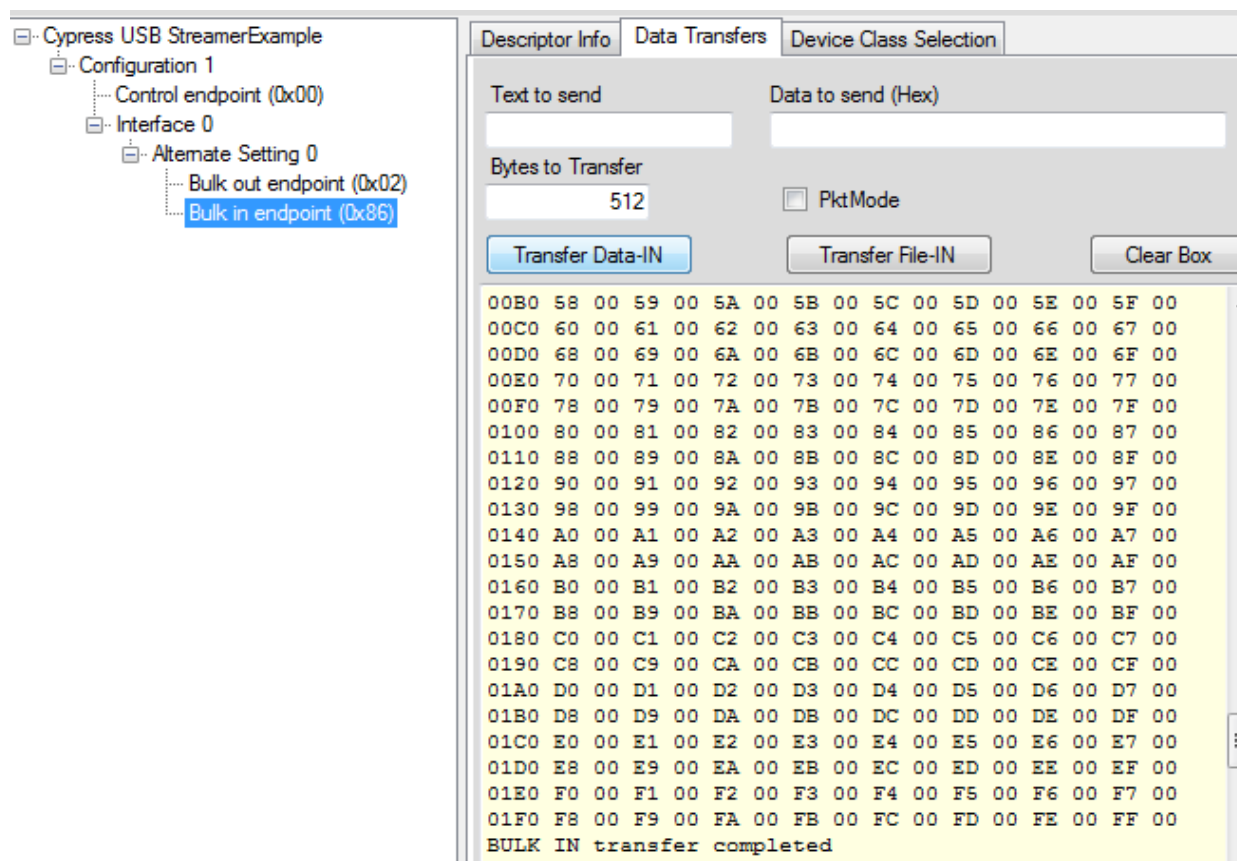
Figure 23. Reading the Same Data From IN Endpoint 6



6.3.5 Verifying the result for Stream IN bit-stream

If the FPGA is configured with Stream IN bit-stream (in step 4), it puts the FPGA in master mode. The FPGA then sends incremental data to endpoint 6 IN of FX2LP. For verifying the Stream IN operation, choose **Bulk in endpoint (0x86)** in the Control Center window and then click the **Transfer Data-IN** button. You can view the data as shown in the following figure:

Figure 24. Reading Data From IN Endpoint 6



6.3.6 Verifying the result for Stream OUT bit-stream

If the FPGA is configured with Stream OUT bit-stream (in step 4), it puts the FPGA in master mode. The FPGA then reads data from the OUT endpoint 2 of FX2LP. You can perform any number of OUT transfers to the endpoint 2 using USB Control Center or you can run a streamer application to verify the OUT transfers on endpoint 2. FPGA reads data from the OUT endpoint 2 based on the flags coming from the FX2LP. FPGA simply ignores the received data and waits for more data from the OUT endpoint 2 of FX2LP.

6.4 Throughput Measurement

The FPGA is configured with the code for data streaming and the interface throughput is measured with Cypress Streamer application, which is included in [SuiteUSB 3.4](#). A throughput of around 39 MB/s is measured on an Intel 7 series/c216 chipset family running Windows 7, 64-bit, with the device bound to CyUSB.sys driver (version 3.4.7). More information about the throughput evaluation of a streaming device is available in the application note [AN4053 - Streaming Data Through Isochronous/Bulk Endpoints on EZ-USB FX2™ and EZ-USB FX2LP™](#).

Figure 25. Throughput Measured for Stream IN Transfers

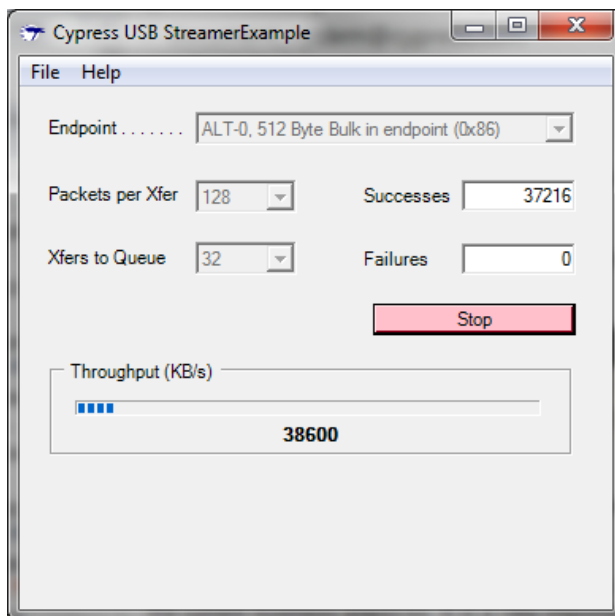
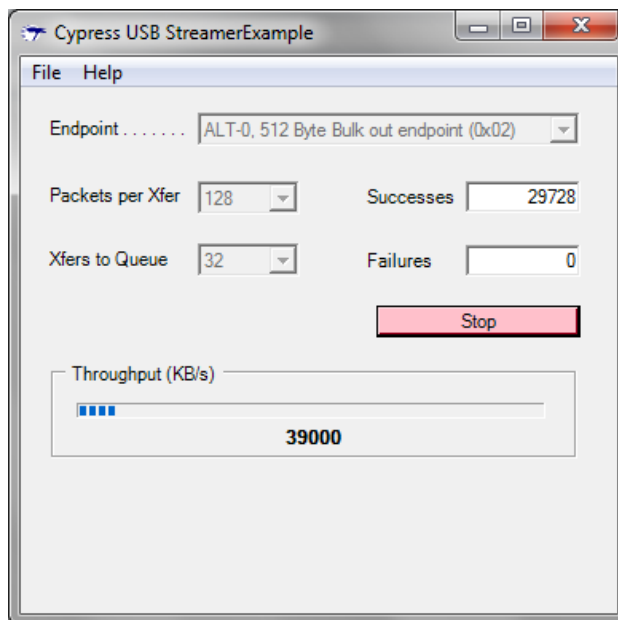


Figure 26. Throughput Measured for Stream OUT Transfers



Note: These throughput numbers are measured by selecting **256 “Packets per Xfer”** and **64 “Xfers to Queue”**, in the streamer application.

7 Associated Project Files

Table 3 describes the files attached with this application note.

Table 3. Description of Application Note Files

File/Folder Name		Description
FX2LP firmware		FX2LP firmware project source files
FPGA Source Code_Verilog	Loopback	FPGA Verilog source code to perform data loopback.
	Stream IN	FPGA Verilog source code to perform stream IN data transfers.
	Stream OUT	FPGA Verilog source code to perform stream OUT data transfers.
FPGA Source Code_VHDL	Loopback	FPGA VHDL source code to perform data loopback.
	Stream IN	FPGA VHDL source code to perform stream IN data transfers.
	Stream OUT	FPGA VHDL source code to perform stream OUT data transfers.
512_count.hex		512 bytes of data to perform file transfers.
Firmware_SDCC		FX2LP firmware project source files to be built using SDCC compiler.
Firmware_SDCC/Release		Files generated after building FX2LP firmware project using SDCC compiler.
Readme_SDCC.pdf		Document explaining the procedure to build the project using SDCC compiler.
Sdccman.pdf		The SDCC manual explaining about SDCC compiler.

8 How to Port This Design to Work With Altera® FPGA

You need to take care of the following steps while porting this design to work with Altera FPGA:

- Primitives to be generated using Altera tools:
 - PLL for clock generation
 - DDR to provide the interface clock to FX2LP
- Pin mapping of FX2LP Slave FIFO interface signals to FPGA.

9 Summary

This application note describes how to set up an FPGA-FX2LP interface with FX2LP configured in the Slave FIFO mode. The associated projects contain the firmware for initializing FX2LP in Slave FIFO mode and Verilog, HDL code for configuring an FPGA to act as master to FX2LP.

About the Author

Name: Rama Sai Krishna V
Title: Applications Engineer Staff.

Document History

Document Title: AN61345 - Designing with EZ-USB® FX2LP™ Slave FIFO Interface

Document Number: 001-61345

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2896051	HRID	04/30/10	New Application note.
*A	3028509	HRID	09/13/10	Details on the PC specification on which the throughput measurement screenshot was captured. Link added to the application note detailing throughput measurement AN4053 Streaming Data Through Isochronous/Bulk Endpoints on EZ-USB FX2™ and EZ-USB FX2LP . Addition of FPGA details which were used for the project implementation in abstract.
*B	3049785	HRID	10/06/10	Post to external web.
*C	3187032	HRID	03/03/2011	Included details of software tools used and the associated projects.
*D	3539468	PRJI	03/01/2012	Updated the document with Virtex5 FPGA. Updated template. Modified abstract and major rewrite.
*E	3697826	PRJI	08/01/2012	Updated the document with verilog code for 22partan-3E FPGA.
*F	3732422	PRJI	09/03/2012	Updated the associated project files.
*G	3871103	PRJI	01/15/2013	Added a section on "Design Example" with a Spartan 6 compatible FPGA example Added a note regarding timing constraints of the examples other than the Spartan 6 example
*H	3938845	RSKV	03/27/2013	Details of Virtex 3 and 5 are removed. Design is modified so that FX2LP accepts clock from FPGA. Design is modified to support 16-bit interface. Design is changed to support burst transfers. Ztex hardware board pictures are added and major rewrite.
*I	4209055	RSKV	12/03/2013	Updated attachments in attached Associated Project: FX2LP firmware source files that can be built with SDCC compiler are added in the attachment. Readme_SDCC.pdf is also provided with the attachment. This has the instructions to build the FX2LP firmware project with SDCC compiler. Updated Associated Project Files: Updated Table 3 (To include the changes done to the attachment). Updated in new template.
*J	4314060	RSKV	03/19/2014	Description of FX2LP firmware is added
*K	5194427	NIKL	03/29/2016	Added link to code examples Updated template
*L	5705702	BENV	04/21/2017	Updated logo and copyright

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

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

PSoC® Solutions

[PSoC 1](#) | [PSoC 3](#) | [PSoC 4](#) | [PSoC 5LP](#) | [PSoC 6](#)

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

© Cypress Semiconductor Corporation, 2010-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spanion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spanion, the Spanion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.