ECHELON®

FTXL Hardware Guide

078-0364-01A

Echelon, LONWORKS, LONMARK, LonTalk, Neuron, 3120, 3150, and the Echelon logo are trademarks of Echelon Corporation registered in the United States and other countries. FTXL, 3190, and ShortStack are trademarks of Echelon Corporation.

Other brand and product names are trademarks or registered trademarks of their respective holders.

Neuron Chips and other OEM Products were not designed for use in equipment or systems, which involve danger to human health or safety, or a risk of property damage and Echelon assumes no responsibility or liability for use of the Neuron Chips in such applications.

Parts manufactured by vendors other than Echelon and referenced in this document have been described for illustrative purposes only, and may not have been tested by Echelon. It is the responsibility of the customer to determine the suitability of these parts for each application.

ECHELON MAKES AND YOU RECEIVE NO WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY OR IN ANY COMMUNICATION WITH YOU, AND ECHELON SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Echelon Corporation.

Printed in the United States of America.
Copyright © 2008 Echelon Corporation.

Echelon Corporation
www.echelon.com

Welcome

Echelon's FTXL™ products enable any product that contains an Altera® Nios® II processor to quickly and inexpensively become a networked smart device. An FTXL device includes a complete ANSI/CEA 709.1-B (EN14908.1) implementation that runs on the Nios II embedded processor. Thus, the FTXL 3190™ Smart Transceiver Chip provides a simple way to add LONWORKS® networking to new or existing smart devices. The FTXL Transceiver is easy to use because it has a simple host application programming interface (API), a pre-built link-layer driver, a simple hardware interface, and comprehensive tool support.

This document describes the hardware interfaces for an FTXL device and the development boards for which the FTXL Developer's Kit provides reference designs.

See the *FTXL User's Guide* for a description of the architecture of an FTXL device and how to develop the software for an FTXL device.

Audience

This document assumes that the reader has a good understanding of the LONWORKS platform, FPGA device design, and programming for the Altera Nios II processor.

Related Documentation

In addition to this manual and the *FTXL User's Guide* (078-0363-01A), the FTXL Developer's Kit includes recent editions of the following manuals:

- *Neuron C Programmer's Guide* (078-0002-02G). This manual describes the key concepts of programming in Neuron® C Version 2 and describes how to develop a LONWORKS application.
- *Neuron C Reference Guide* (078-0140-02E). This manual provides reference information for writing programs that use the Neuron C language.
- *NodeBuilder Errors Guide* (078-0193-01B). This manual describes error codes issued by the Neuron C compiler.

The FTXL Developer's Kit also includes the reference documentation for the FTXL LonTalk API, which is delivered as a set of HTML files.

After you install the FTXL software, you can view all of these documents from the Windows **Start** menu: select **Programs → Echelon FTXL Developer's Kit → Documentation,** then select the document that you want to view.

The following manuals are available from the Echelon Web site (www.echelon.com) and provide additional information that can help you to develop applications for an FTXL Transceiver:

- *Introduction to the LONWORKS System* (078-0183-01A). This manual provides an introduction to the ANSI/CEA-709.1 (EN14908) Control

Networking Protocol, and provides a high-level introduction to LONWORKS networks and the tools and components that are used for developing, installing, operating, and maintaining them.

- *LONMARK Application Layer Interoperability Guidelines.* This manual describes design guidelines for developing applications for open interoperable LONWORKS devices, and is available from the LONMARK® Web site, www.lonmark.org.
- *FT 3120 / FT 3150 Smart Transceiver Data Book* (005-0139-01D). This manual provides detailed technical specifications on the electrical interfaces, mechanical interfaces, and operating environment characteristics for the FT 3120®, FT 3150®, and FTXL 3190 Smart Transceivers.

All of the FTXL documentation, and related product documentation, is available in Adobe® PDF format. To view the PDF files, you must have a current version of the Adobe Reader®, which you can download from Adobe at: www.adobe.com/products/acrobat/readstep2.html.

Related Altera Product Documentation

For information about the Altera Nios II family of embedded processors and associated tools, see the Altera Nios II Literature page: www.altera.com/literature/lit-nio2.jsp.

Table 1 lists Altera product documents that are particularly useful for the FTXL Developer's Kit.

Table 1. Related Altera Documentation

Product Category	Documentation Titles
Quartus® II software	Introduction to Quartus II Software Quartus II Quick Start Guide Quartus II Development Software Handbook v7.2
Nios II processor	Nios II Hardware Development Tutorial Nios II Software Development Tutorial (included in the online help for the Nios II EDS integrated development environment) Nios II Flash Programmer User Guide Nios II Processor Reference Handbook Nios II Software Developer's Handbook

Product Category	Documentation Titles
Cyclone® II and Cyclone III FPGA and device configuration	Cyclone II Device Handbook Cyclone III Device Handbook Configuration Handbook
USB-Blaster™ download cable	USB-Blaster Download Cable User Guide
Software licensing	Quartus II Installation & Licensing for Windows AN 340: Altera Software Licensing

Related devboards.de Product Documentation

The FTXL Developer's Kit uses the devboards.de DBC2C20 Altera Cyclone II Development Board for its examples and reference designs. For information about the DBC2C20 Altera Cyclone II Development Board, including the most current data sheet for the board, see the DBC2C20 page: www.devboards.de/index.php?mode=products&kategorie=14.

Table of Contents

Welcome iii
Audience iii
Related Documentation iii
Related Altera Product Documentation iv
Related devboards.de Product Documentation v

FTXL Hardware Overview 1
Overview 2
The FTXL Developer's Kit 3
The FTXL Development Process 3
Hardware Design 4
FPGA Design 5
Software Design 5

FTXL Developer's Kit Hardware 7
Overview of the FTXL Developer's Kit Hardware 8
The DBC2C20 Development Board 8
Buttons and LEDs 9
Jumper Settings 11
Connectors and Headers 11
The FTXL Adapter Board 13
Jumper Settings 13
Connectors and Headers 13
The FTXL Transceiver Board 15
LEDs 16
Jumper Settings 16
Connectors and Headers 16

FTXL Transceiver Hardware Interface 19
Overview of the Hardware Interface 20
DC-DC Converter 20
Control Signal Buffer 20
Data Bus Isolation 20
Pull-Up Resistors for Communications Lines 21
The Parallel Communications Interface 21
Token Passing and Handshaking 23
Transferring Data 23
FTXL Transceiver Pin Characteristics 24
I/O Pins 24
IRQ Pin 24
Reset Pin 25
Reset Function 25
Power-Up Sequence 26
Software Controlled Reset 27
Watchdog Timer 27
LVI Considerations 27
Reset Processes and Timing 27
Service Pin 27
Clock Pins 27
FPGA Pin Assignments for the FTXL Transceiver 28
Control Flow: Host Receiving Data from the FTXL Transceiver 29
Control Flow: Host Sending Data to the FTXL Transceiver 32

FPGA Design for the FTXL Transceiver 37
Overview 38
Using the Reference Design 38
Developing a New FPGA Design 38
FPGA Device Requirements 39
Nios II Processor 40
FPGA Configuration Device 40
FTXL Components 41
FTXL Parallel Interface Delay 42
FTXL Parallel I/O Transceiver Interface 43
FTXL Service LED 46
FTXL Service Pin 47
FTXL Transceiver Interrupt 47
FTXL Transceiver Reset 47
Phase-Locked Loop 48
DBC2C20 Components 48
Timers 48
External Memory 48
Addressing, Size, and IRQ Requirements 49
FTXL Hardware Abstraction Layer 50
Other Hardware Design Considerations 51

Working with the Altera Development Environments 53
Development Tools 54
Using a Device Programmer for the FPGA Device 55
Setting Component Search Paths 56
Adding FTXL Components to an Existing Design 57
Modifying the SOPC Builder Design 57
Modifying the Quartus II Design 60
Building the Application Image 61
Loading the Application Image into the FPGA Device 62

Using the Bring-Up Application to Verify FTXL Hardware Design 63
Overview 64
Application Framework 64
Interrupt Functions from the FTXL HAL 64
FTXL Transceiver Interface 65
Reset Signal 65
Status Signals 66
Data Register 66
Interrupt 68
Working with the Nios IDE for the Bring-Up Application 69
Creating a New Application Project 69
Building the Application Image 70
Running the Application from the Nios IDE 70
Running the Tests 70
Reset Test 71
Token Passing Test 73
Data Passing Test 76
Interrupt Test 78
Service Pin and LED Test 79
Designing Additional Tests 80

Index 81

1

FTXL Hardware Overview

This chapter provides an overview of the FTXL Developer's Kit and the development process for developing a device based on the FTXL Transceiver Chip.

Overview

Echelon's Free Topology Smart Transceivers provide a well-tested and cost-effective platform for many distributed control applications that are built on LONWORKS technology. For high value sensors, smart actuators, or terminal equipment controllers, an FT Smart Transceiver provides a well matched cost-to-capability ratio. For more complex applications, the Echelon FTXL Transceiver Chip provides an alternate processing platform for high-performance LONWORKS applications.

The FTXL solution includes the following elements:

- The FTXL LonTalk protocol stack
  The FTXL stack is a C++ implementation of the ANSI/CEA 709.1-B protocol stack that has been ported to run on the Altera Nios II processor, implemented on an Altera Cyclone II FPGA.
- FTXL Transceiver Chip
  The FTXL 3190 Smart Transceiver Chip is an FT 3120 Smart Transceiver that includes a firmware image that allows it to run as a layer 2 parallel interface network transceiver.
- FTXL Design Components
  The FTXL Developer's Kit includes the FTXL design components for the Altera SOPC Builder tool and Quartus II software.
- FTXL Reference Design
  The reference design includes an Altera Quartus II project that targets a specific Cyclone II development board, and provides the necessary VHDL modules and configuration files to build an example Nios II target for the FTXL LonTalk stack and FTXL Transceiver.

The FTXL Transceiver Chip can be configured to run at any of the following clock frequencies, depending on the requirements of the FTXL device:

- 5 MHz
- 10 MHz
- 20 MHz
- 40 MHz

The FTXL LonTalk protocol stack provides support for the following configurations:

- Up to 4096 addresses
- Up to 200 receive transactions
- Up to 2500 transmit transactions
- Up to 4096 network variables
- Up to 8192 alias table entries

The FTXL Developer's Kit

The FTXL Developer's Kit is a development toolkit that contains the hardware designs, software designs, and documentation needed for developing applications that use an FTXL Transceiver. The kit includes the following components:

- Hardware and software design files for the FPGA design, including Quartus II files, SOPC Builder files, and Nios IDE files
- Hardware component files for the FPGA development board
- The FTXL LonTalk protocol stack, delivered as a C object library
- Software source files for the LonTalk application programming interface (API)
- A set of example programs that demonstrate how to use the FTXL LonTalk stack and LonTalk API to communicate with a LONWORKS network
- The LonTalk Interface Developer utility, which defines parameters for your FTXL application program and generates required device interface data for your device
- Documentation, including this *FTXL Hardware Guide*, the *FTXL User's Guide*, and HTML documentation for the LonTalk API

The FTXL Developer's Kit is available as a free download from www.echelon.com/ftxl. See the *FTXL User's Guide* for information about the software components of the FTXL Developer's Kit.

The FTXL Developer's Kit also refers to three hardware development boards that are available from devboards GmbH, www.devboards.de. You can also contact EBV Elektronik GmbH, www.ebv.com. The FTXL Developer's Kit uses these boards for its examples and reference designs. These boards are:

- The *DBC2C20 Altera Cyclone II Development Board*, which provides the FPGA device and peripheral I/O
- The *FTXL Adapter Board*, which primarily provides voltage regulation between the DBC2C20 development board and the FTXL Transceiver Board
- The *FTXL Transceiver Board*, which includes the FTXL Transceiver Chip and a LONWORKS network connector

See Chapter 2, *FTXL Developer's Kit Hardware*, on page 7, for more information about the hardware for the FTXL Developer's Kit.

The FTXL Development Process

An FTXL device is comprised of the following basic elements:

- An Echelon FTXL Transceiver Chip that communicates with a LONWORKS network
- An FPGA device, running an Altera Nios II processor, that runs the FTXL application program

- RAM, read/write non-volatile memory (such as flash) to store configuration data, and non-volatile memory (such as flash) to store the FTXL application
- The associated FPGA design and printed-circuit board (PCB) design for the device

Thus, the development process for an FTXL device includes the following tasks:

1. Gather the requirements for the device
2. Based on those requirements, determine the necessary functionality of the FPGA device, including the Nios II processor, on-chip memory, and any intellectual property (IP) cores
3. Implement the FPGA design using the Altera Quartus II software
4. Choose the physical FPGA device, along with its corresponding configuration device, and load the design into the device
5. Design and implement a prototype PCB design for the FPGA device and FTXL Transceiver (this step can be deferred if you use the reference designs for the DBC2C20 development board for prototyping)
6. Design the FTXL application program, using the FTXL LonTalk protocol stack
7. Load the software into the FPGA device
8. Test the completed FTXL device
9. Integrate the FTXL device into a LONWORKS network
10. Design and implement the final PCB design for the FPGA device and FTXL Transceiver

This book describes many of these tasks. See the appropriate Altera documentation for additional design considerations for an FPGA device; see the *FT 3120 / FT 3150 Smart Transceiver Data Book* for design considerations for an FT 3120 Smart Transceiver, which shares electrical and physical characteristics with the FTXL Transceiver; and see the *FTXL User's Guide* for information about software design for an FTXL device.

Hardware Design

A minimal hardware design for an FTXL device includes the following elements:

- An Echelon FTXL 3190 Smart Transceiver Chip
- An Echelon FT-X1 or FT-X2 Communication Transformer
- A crystal oscillator for the FTXL Transceiver Chip
- Associated circuitry for communications, as described in the *FT 3120 / FT 3150 Smart Transceiver Data Book*
- A charge pump DC-DC converter (or similar device) to allow the FTXL Transceiver Chip and FPGA device to share a common power supply
- A FPGA device, such as an Altera Cyclone II or Cyclone III device
- An FPGA serial configuration device

- External memory (such as external RAM) for the FTXL application program
- Non-volatile memory (such as flash memory) for network configuration data
- Associated user I/O, such as a service pin and LED, reset button and LED, and other I/O for the device
- A power supply

A more robust or complex design includes additional hardware components, such as additional user I/O, support for a USB or other network interface, signal processors, or other coprocessors.

Because an FTXL device is a communications device, its design must include considerations for electromagnetic compatibility (EMC), including electrostatic discharge (ESD), radio frequency (RF) immunity, and resistance to electromagnetic interference (EMI).

See the *FT 3120 / FT 3150 Smart Transceiver Data Book* for overall design considerations for an FT Smart Transceiver, including the FTXL Transceiver.

FPGA Design

A minimal FPGA design for an FTXL device includes the following elements:

- An Altera Nios II processor
- One or more phase-locked loop (PLL) components
- Definitions for the FTXL parallel communications interface (see *The Parallel Communications Interface* on page 21)
- A definition for the FTXL Transceiver reset signal
- A definition for the FTXL Transceiver interrupt signal
- A definition for the FTXL service pin
- A definition for the FTXL service LED
- Definitions for user I/O
- An interface for both on-chip and external memory

Because an FPGA device can include multiple processors and many intellectual property (IP) cores, a more robust or complex design can include any number of additional design elements. These additional design elements, in turn, help determine the specific type of FPGA device that your FTXL device requires.

Chapter 4, *FPGA Design for the FTXL Transceiver*, on page 37, describes requirements for the FPGA design for an FTXL device.

Software Design

Software design for an FTXL device requires a host application program that runs on the Nios II processor and uses the FTXL LonTalk protocol stack to manage the FTXL Transceiver for communications with a LONWORKS network.

The LonTalk application programming interface (API) provides essential functions for managing an FTXL Transceiver. This API shares many features

and functions with the LonTalk Compact API, so that it is possible to migrate a ShortStack™ device to use an FTXL Transceiver.

An FTXL host program uses an embedded operating system (generally, a real-time operating system (RTOS)) for intra-processor communications and task management. In addition to the LonTalk API, the FTXL Developer's Kit provides an operating system abstraction layer (OSAL) so that your host program can use any RTOS that meets your system's requirements. The example programs that are included with the FTXL Developer's Kit use the Micrium µC/OS-II operating system.

See the *FTXL User's Guide* for more information about software design for an FTXL device.

2

FTXL Developer's Kit Hardware

This chapter describes the three development boards that comprise the hardware for the FTXL Developer's Kit.

Overview of the FTXL Developer's Kit Hardware

The FTXL Developer's Kit requires the three hardware development boards listed in **Table 2**. These boards are available from devboards GmbH, www.devboards.de. You can also contact EBV Elektronik GmbH, www.ebv.com.

Table 2. FTXL Developer's Kit Hardware

Board Name	Description	devboards Order Code
DBC2C20 Altera Cyclone II Development Board	Development board that includes the Cyclone II FPGA device and user I/O	DBC2C20USBB or DBC2C20USBB-IPN
FTXL Adapter Board	Adapter board between the 3.3 V FPGA board and the 5 V FTXL Transceiver board	DBE-ADAP
FTXL Transceiver Board	Development board for the FTXL Transceiver Chip	DBE-FT-PAR

The DBC2C20 development board includes an Altera USB-Blaster download cable. Contact your Altera representative for information about acquiring a Nios II development license.

For information about the DBC2C20 development board, see the *Datasheet DBC2C20 Cyclone II Development Board* document, available from the devboards Web site.

The DBC2C20 Development Board

To work with the Nios II processor for an FTXL device, you can use any of the many available tools that support the Nios II family of embedded processors. However, this document describes only the devboards GmbH DBC2C20 Altera Cyclone II Development Board which is part of the FTXL Developer's Kit, as described in **Table 3** on page 9.

The FTXL Developer's Kit was built for the DBC2C20 development board. You could use another development board, such as the Altera Nios II Development Board, Cyclone II Edition, but because the connectors on the FTXL Adapter Board were designed to match connectors on the DBC2C20 development board, you must create ribbon-cable connections to match a different development board. In addition, you must create your own hardware and software projects for a different development environment.

The DBC2C20USBB-IPN package includes a 9.6 VA, 800 mA power supply that is appropriate for European installations; for other geographies, you can use any input power supply with a 2.1 mm pin, center-negative barrel connector, from 7.5 V to 24 V, such as the 9 V Echelon 78010R power supply.

Table 3. Hardware Development Platform for the Nios II Processor

devboards DBC2C20 Altera Cyclone II Development Board
The DBC2C20 Cyclone II Development Board includes an Altera Cyclone II EP2C20 FPGA with 20 000 logic elements (LEs) that provides flexibility and performance for a wide range of applications. The board also includes: • 16 MB SDRAM • 8 MB flash memory • 16 Mbit EPCS16 configuration device • 1 MB SRAM • Twenty-four 3.3 V I/O ports • For communication tasks, one RS-232, four RS-485, and two Controller Area Network (CAN) transceivers are available • Two 10/100 Mbps Ethernet PHYs are available for Ethernet-based communication • A 24 V, 16-bit wide I/O port can connect the DBC2C20 board directly to industrial control systems • Two low-voltage differential signaling (LVDS) ports, available on RJ-45 connectors, can be used for high-speed board communication • For visualization tasks, an LVDS-based thin-film transistor (TFT) interface is available

Buttons and LEDs

The DBC2C20 development board includes four pushbuttons (**P24**, **P25**, **P26**, and **P28**) and eight light-emitting diodes (LEDs – **D17** through **D24**). However, the FTXL Developer's Kit uses only one button and one LED, as shown in **Figure 1** on page 10 and listed in **Table 4** on page 10.

Board Top View

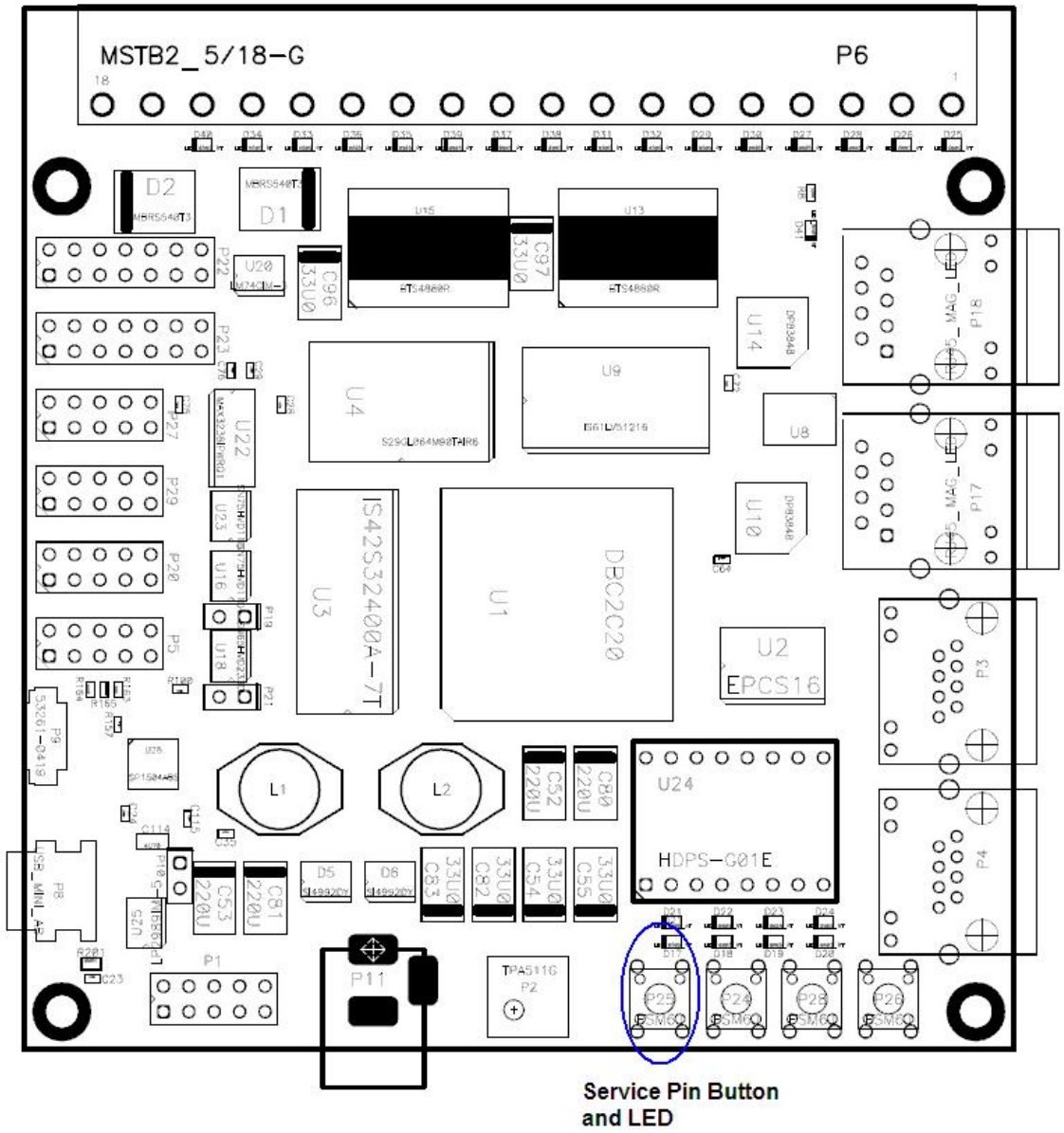


Figure 1. Service Pin Button and LED on the DBC2C20 Development Board

Table 4. FTXL Developer's Kit Button and LED on the DBC2C20 Development Board

FTXL Function	DBC2C20 Function	DBC2C20 Name	Cyclone II Pin Assignment
Service Pin Button	Button 0	P25	U1
Service Pin LED	LED 4	D17	U8

The FTXL Developer's Kit does not use the two-digit seven-segment display (**U24**) or the navigation key (**P2**) on the DBC2C20 development board.

Jumper Settings

The DBC2C20 development board includes three sets of jumpers (**P10**, **P19**, and **P21**). For the FTXL Developer's Kit, all of these jumpers remain unmounted.

Connectors and Headers

The DBC2C20 development board includes 15 connectors and headers. The FTXL Developer's Kit uses only the connectors that are shown in **Figure 2** and listed in **Table 5**.

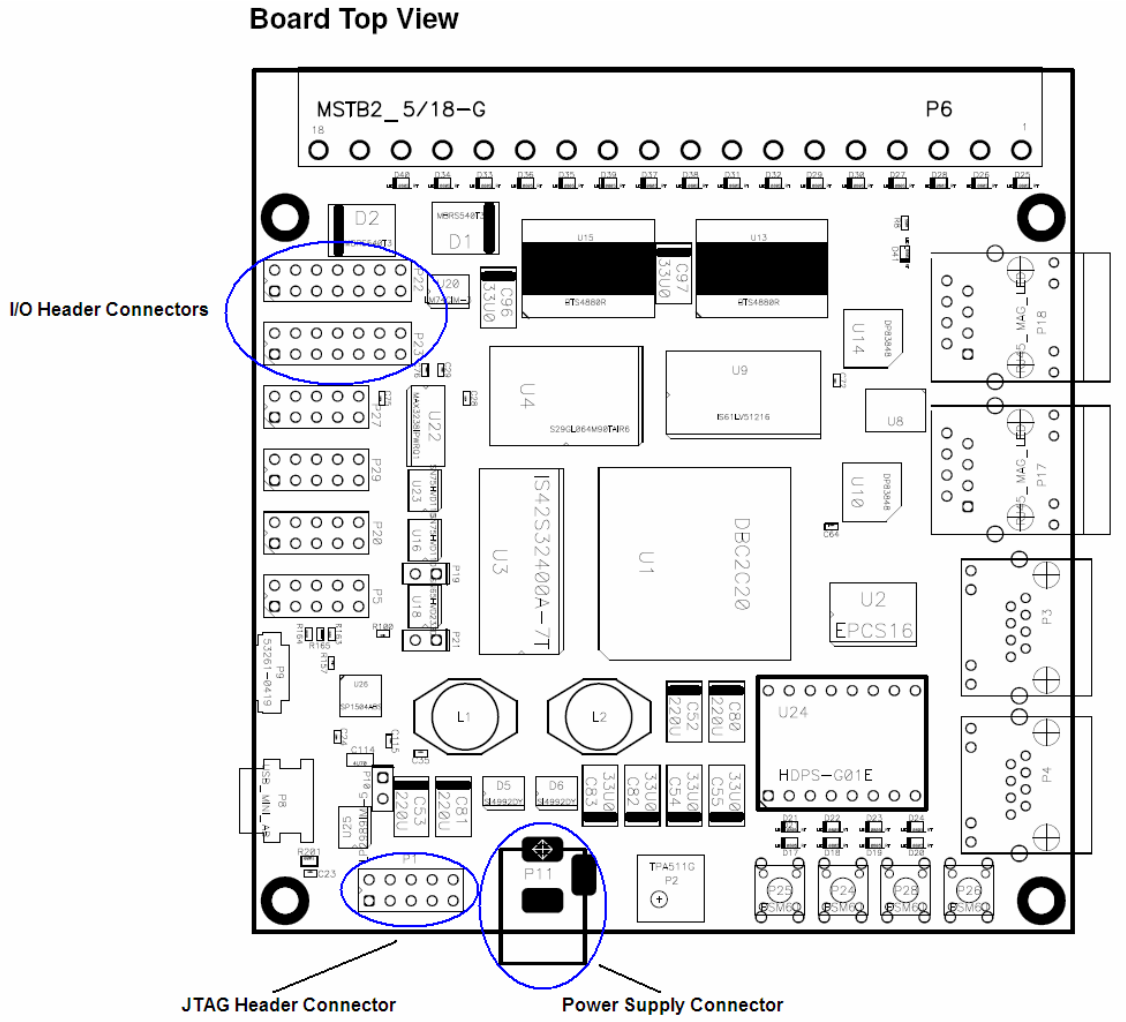


Figure 2. Connectors and Headers on the DBC2C20 Development Board

Table 5. FTXL Developer's Kit Connectors and Headers on the DBC2C20 Development Board

FTXL Function	DBC2C20 Function	DBC2C20 Name
Header for programming the Cyclone II FPGA and flash memory	JTAG Connector for the Altera USB-Blaster download cable	P1

FTXL Function	DBC2C20 Function	DBC2C20 Name
Main power	Power supply connector	P11
FTXL Transceiver Chip I/O	3.3 V I/O Connector	P22
FTXL Transceiver Chip I/O	3.3 V I/O Connector	P23

Figure 3 shows the connections for the **P22** and **P23** headers. The names in parentheses are the Cyclone II pin assignments for the I/O lines. The FTXL Developer's Kit does not use pins 8-14 (PIO17-PIO23) on the **P22** header.

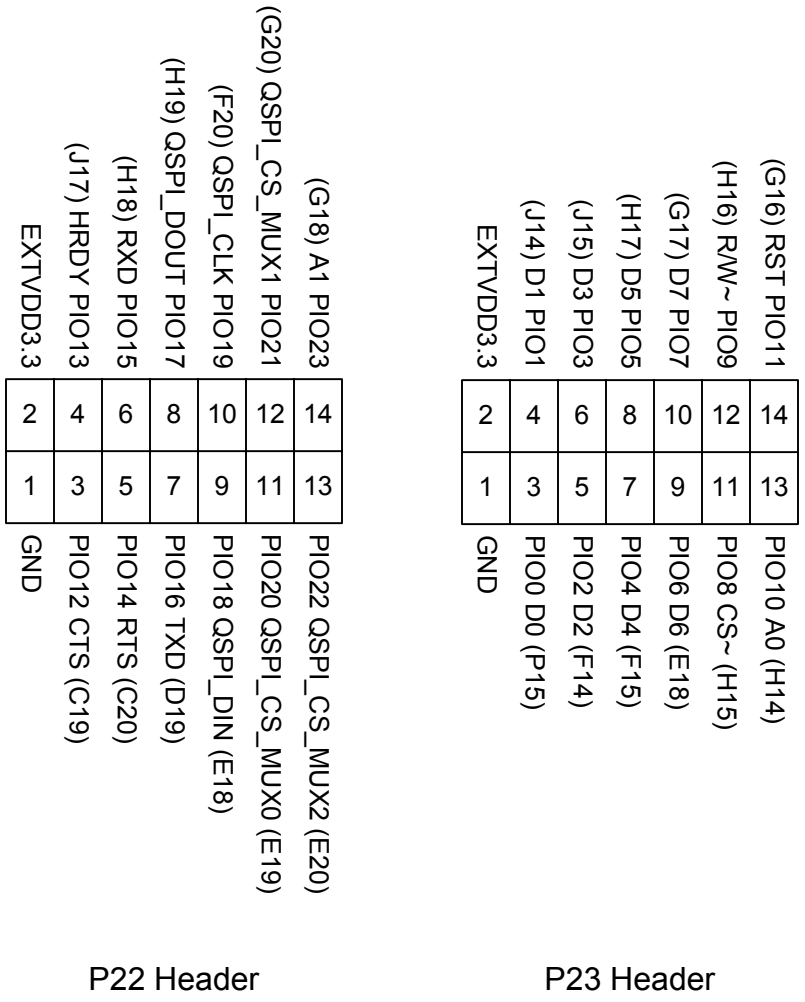


Figure 3. FTXL Transceiver Chip I/O Header Connections for P22 and P22 Headers

For more information about the pins used for the FTXL Transceiver, see *FTXL Transceiver Pin Characteristics* on page 24.

The FTXL Adapter Board

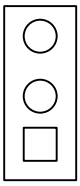
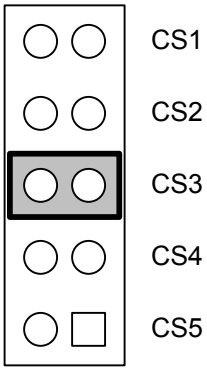
The primary function of the FTXL Adapter Board is to provide the 5 V power to the FTXL Transceiver Board from the 3.3 V power of the DBC2C20 development board. The FTXL Adapter Board also provides access to all of the FTXL Transceiver I/O lines through headers on the board.

Connect the FTXL Adapter Board to the DBC2C20 development board by joining the FTXL Adapter Board's **J7** and **J5** connectors to the DBC2C20 development board's **P22** and **P23** headers.

Jumper Settings

The FTXL Adapter Board includes two sets of jumpers (**J4** and **J9**). However, the FTXL Developer's Kit uses only the **J9** jumper, as described in **Table 6**.

Table 6. FTXL Adapter Board Jumper Settings

Function	Jumper	Description
Interface Selector (J4)	 J4	This jumper is not used for the FTXL Developer's Kit. Leave this jumper unmounted.
Chip Select Signal Selector (J9)	 J9	This jumper selects which Chip Select signal to use in the **J2** connector. You can set this jumper in any position, but the setting for the **J9** jumper on the FTXL Adapter Board must match the jumper setting for the **J7** jumper on the FTXL Transceiver Board. The factory shipped default setting for this jumper is to mount it across pins 5 and 6, as shown.

Connectors and Headers

The FTXL Adapter Board includes eight connectors and headers, of which the FTXL Developer's Kit uses the ones listed in **Table 7** on page 14.

The FTXL Developer's Kit does not use the **J3** or **J10** headers.

Table 7. FTXL Adapter Board Connectors and Headers

FTXL Developer's Kit Function	FTXL Adapter Board Function	FTXL Adapter Board Name
Connects FTXL Adapter Board with FTXL Transceiver Board for FTXL Transceiver Chip I/O	Hirose stacking header	J1
Connects FTXL Adapter Board with FTXL Transceiver Board for FTXL Transceiver Chip I/O	Hirose stacking header	J2
Provides access to FTXL Transceiver Chip I/O	Header and connector for FTXL Transceiver Chip I/O with DBC2C20 development board	J5
Provides access to FTXL Transceiver Chip I/O	Header and connector for FTXL Transceiver Chip I/O with DBC2C20 development board	J7

In addition, the FTXL Adapter Board provides an area for prototyping or measurement through headers **J6** and **J8**, which correspond functionally to headers **J5** and **J7**. See **Figure 3** on page 12 for the pin assignments of the DBC2C20 development board headers **P22** and **P23**, which directly connect to the FTXL Adapter Board headers **J5** and **J7**.

Figure 4 on page 15 shows the connections for the **J1** and **J2** Hirose stacking headers. Note that the FTXL Developer's Kit does not use the connections for pins 36-44 or pin 81 on the **J2** header.

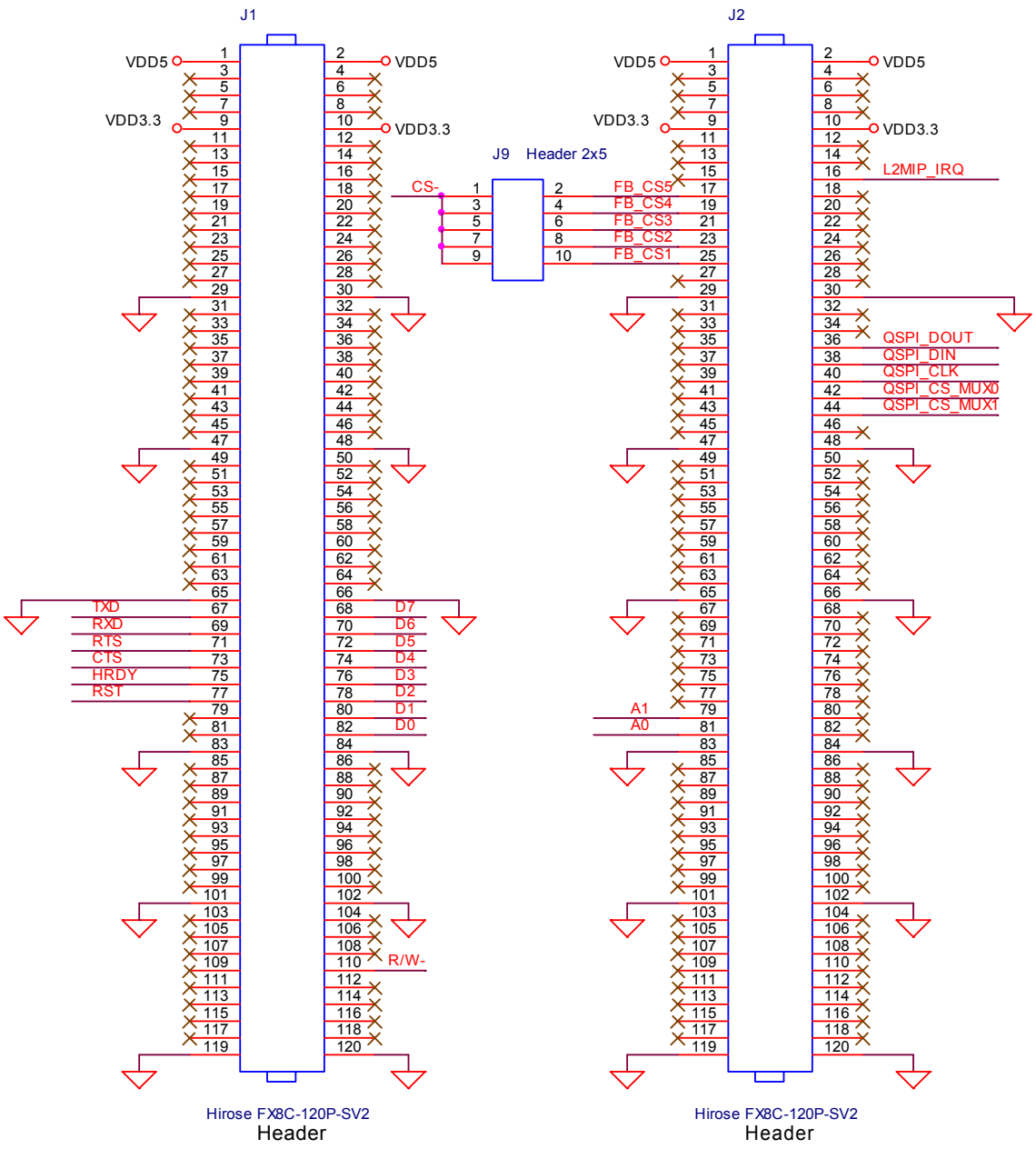


Figure 4. FTXL Adapter Board Hirose Stacking Headers J1 and J2

The FTXL Transceiver Board

The FTXL Transceiver Board is the development board for the FTXL Transceiver Chip. It also provides LONWORKS network connectivity.

The FTXL Transceiver Chip on the FTXL Transceiver Board runs at 20 MHz.

Connect the FTXL Transceiver Board to the FTXL Adapter Board by joining the two boards' Hirose stacking headers.

LEDs

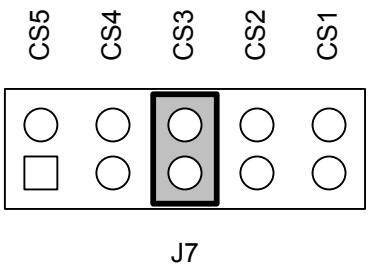
The FTXL Transceiver Board includes two LEDs (**D11** and **D14**). These LEDs are active while the FTXL Transceiver Chip is sending or receiving network data.

User interaction with the FTXL Transceiver is controlled from the DBC2C20 development board.

Jumper Settings

The FTXL Transceiver Board includes one jumper set (**J7**), as described in **Table 8**.

Table 8. FTXL Transceiver Board Jumper Settings

Function	Jumper	Description
Chip Select Signal Selector (J7)	 J7	This jumper selects which Chip Select signal to use in the **J4** connector. You can set this jumper in any position, but the setting for the **J7** jumper on the FTXL Transceiver Board must match the jumper setting for the **J9** jumper on the FTXL Adapter Board. The factory shipped default setting for this jumper is to mount it across pins 5 and 6, as shown.

Connectors and Headers

The FTXL Transceiver Board includes five connectors and headers, as listed in **Table 9**.

Table 9. FTXL Transceiver Board Connectors and Headers

FTXL Developer's Kit Function	FTXL Transceiver Board Function	FTXL Transceiver Board Name
Connects FTXL Transceiver Board with FTXL Adapter Board for FTXL Transceiver Chip I/O	Hirose stacking header and receptacle	J1 and J3
Connects FTXL Transceiver Board with FTXL Adapter Board for FTXL Transceiver Chip I/O	Hirose stacking header and receptacle	J2 and J4
LONWORKS Network Connector	Network Connector	J6

Figure 5 shows the connections for the **J3** and **J4** Hirose stacking headers; **Figure 6** on page 18 shows the connections for their corresponding **J1** and **J2** receptacles.

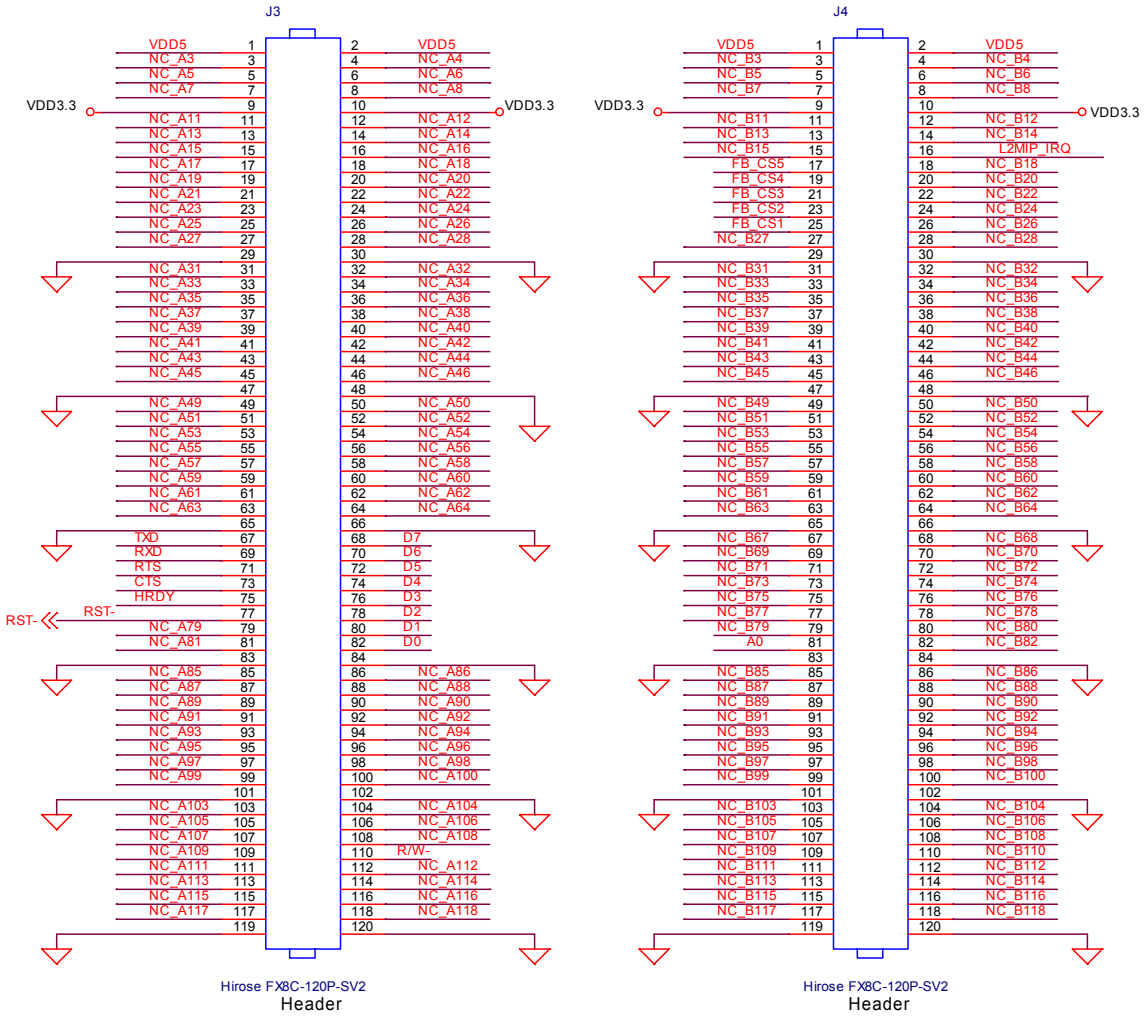


Figure 5. FTXL Transceiver Board Hirose Stacking Headers J3 and J4

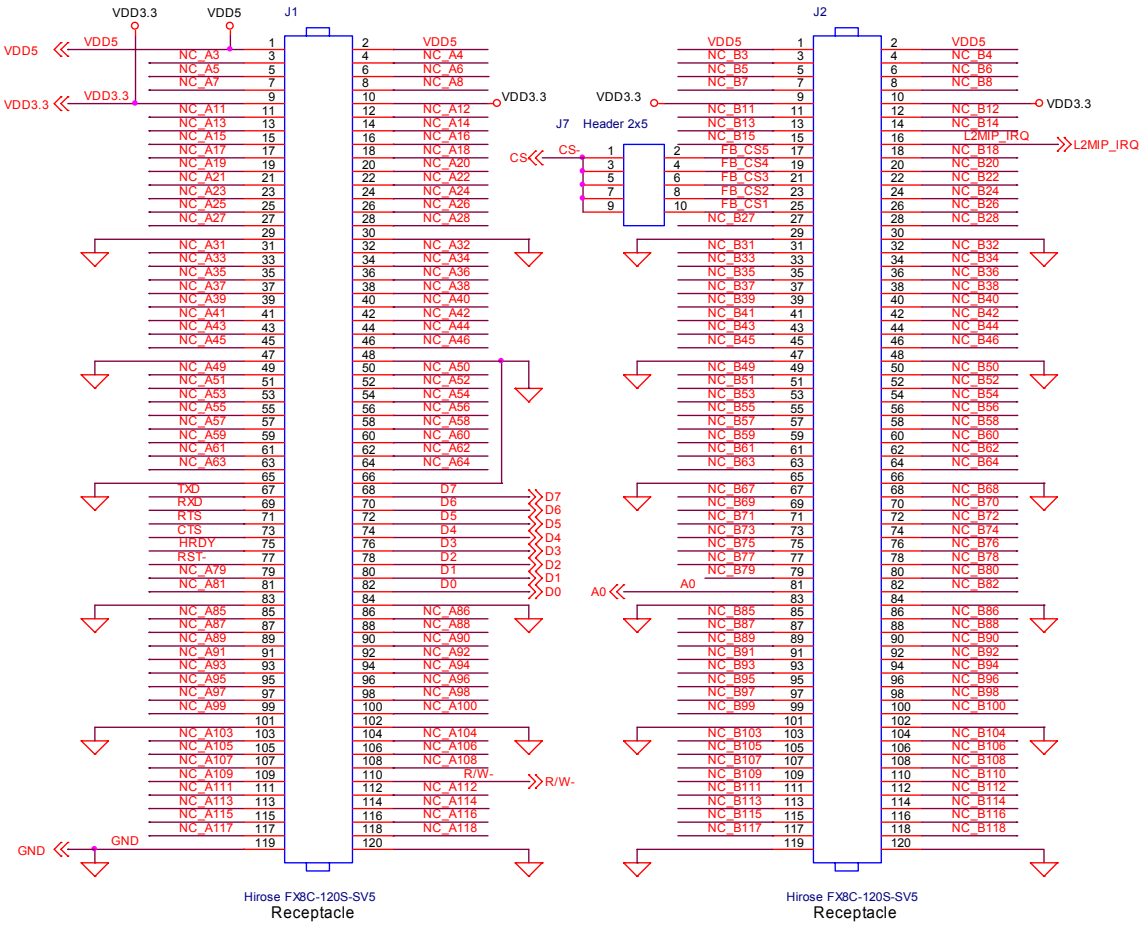


Figure 6. FTXL Transceiver Board Hirose Stacking Header Receptacles J1 and J2

3

FTXL Transceiver Hardware Interface

This chapter describes the hardware interface for the FTXL Transceiver Chip, which is primarily comprised of the parallel communications interface.

Overview of the Hardware Interface

The hardware interface for an FTXL Transceiver is comprised of the parallel communications interface, the pin assignments and characteristics for the FTXL Transceiver Chip, and the pin assignments and characteristics for the FPGA device. This chapter describes the hardware interface.

The FTXL 3190 Free Topology Smart Transceiver shares electrical and physical characteristics with the FT 3120 Smart Transceiver. For information about the hardware interface for an FT 3120 Smart Transceiver, see the *FT 3120 / FT 3150 Smart Transceiver Data Book*.

DC-DC Converter

Because the Cyclone II FPGA requires 3.3 V input voltage, and the FTXL Transceiver Chip requires 5 V input voltage, your hardware design must include either a separate power supply for each of the two parts, or a DC-DC step-up or step-down converter chip with a common power supply for both parts.

Important: The Cyclone II FPGA pins are not 5V-tolerant.

The FTXL Developer's Kit reference design uses the power supply on the DBC2C20 development board, and the FTXL Adapter Board uses a Texas Instruments™ TPS60110 step-up, regulated charge pump DC-DC converter to provide the necessary 5 V for the FTXL Transceiver Chip from the DBC2C20 development board's 3.3 V power supply.

Control Signal Buffer

For an FPGA device that does not have 5V-tolerant input pins, you need to buffer the control signals between the FTXL Transceiver and the FPGA device. The control signals are the CS~, R/W~, A0, IRQ, and RESET~ signals described in *The Parallel Communications Interface* on page 21.

The FTXL Adapter Board uses an NXP® Semiconductor 74AHC541 three-state octal buffer/line driver to provide buffering for the FTXL control signals.

The FTXL RESET~ signal also requires an additional 100 Ω resistor in-series between the FTXL Transceiver and the buffer/line driver device. See the *FT 3120 / FT 3150 Smart Transceiver Data Book* for other considerations for the RESET~ signal, such as providing a comparator circuit that can monitor when the power-supply voltage goes out of tolerance.

Data Bus Isolation

For an FPGA device that does not have 5V-tolerant input pins, you need to provide isolation for the data bus signals between the FTXL Transceiver and the FPGA device. The data bus signals are the D0..D7 signals described in *The Parallel Communications Interface* on page 21.

The FTXL Adapter Board uses an NXP Semiconductor 74AHC245 three-state octal bus transceiver to provide signal isolation for the FTXL data bus.

Pull-Up Resistors for Communications Lines

For the parallel communications interface, you must add 10 kΩ pull-up resistors to all communication lines between the FPGA device and the FTXL Transceiver Chip (the IO0-IO10 pins of the FTXL Transceiver). These pull-up resistors prevent invalid transactions on start-up and reset of the FPGA device or the FTXL Transceiver Chip. Certain I/O pins can revert to a floating state without a pull-up resistor, which can cause unpredictable results.

The Parallel Communications Interface

An FT 3120 Smart Transceiver can connect to application-specific external hardware through 11 pins, named IO0-IO10. The Smart Transceiver design allows these pins to be configured as an *I/O object* that provides programmable access to an I/O driver for a specified on-chip I/O hardware configuration and a specified input or output waveform definition.

The FTXL 3190 Smart Transceiver Chip is an FT 3120 Smart Transceiver with a firmware image that is configured to use the IO0-IO10 pins as a *parallel I/O object*, which defines a bidirectional, half-duplex, 8-bit data port and a 3-bit control port for connecting to the FPGA device. Because the FTXL Transceiver communicates with an external host processor, the FPGA device, the parallel I/O interface is configured in the Smart Transceiver's slave B mode.

Table 10 summarizes the pin assignments for the FTXL Transceiver, and **Figure 7** on page 23 shows the parallel interface for the FTXL Transceiver.

Table 10. FTXL Transceiver Pin Assignments for the Parallel Interface

FTXL Transceiver Pin Number	FTXL Transceiver Pin Name	Signal Name	Direction
4	IO0	D0/HS	Input and output
3	IO1	D1	Input and output
2	IO2	D2	Input and output
43	IO3	D3	Input and output
42	IO4	D4	Input and output
36	IO5	D5	Input and output
35	IO6	D6	Input and output
32	IO7	D7	Input and output
31	IO8	CS~	Input
30	IO9	R/W~	Input
27	IO10	A0	Input

FTXL Transceiver Pin Number	FTXL Transceiver Pin Name	Signal Name	Direction
Note: Signal direction is from the point of view of the FTXL Transceiver Chip.			

When configured in slave B mode, the Smart Transceiver defines a 3-bit control port:

- IO8 is the chip select pin (CS~), and when asserted (driven low), specifies that a byte-transfer operation is in progress. This pin is driven by the FPGA device.
- IO9 is the read/write control pin (R/W~), and determines the direction of the bidirectional data bus. When asserted (driven low), this pin indicates a write operation; when deasserted (driven high), it specifies a read operation. This pin is driven by the FPGA device.
- IO10 is the address pin (A0), and controls the function of the IO0 pin, which can be part of the data I/O (as D0) or can be the handshake signal (as HS). This pin is driven by the FPGA device.

The FTXL LonTalk protocol stack running on the Nios II processor in the FPGA device manages the control port for the communications protocol with the FTXL Transceiver.

Pins IO0-IO7 form the bidirectional data bus (D0-D7) when the IO10 (A0) pin is low, or when the IO10 (A0) pin is high and the IO9 (R/W~) pin is low. The IO0 pin is the HS (handshake) acknowledgment signal to the master when the IO10 (A0) pin is high and IO9 (R/W~) is high. **Table 11** summarizes the states of the control port for determining the function of the IO0 (D0/HS) pin.

Table 11. Controlling the Function of the IO0 Pin

IO0 Function	CS~ State	A0 State	R/W~ State
D0	Low	Low	High or Low
D0	Low	High	Low
HS	Low	High	High

The HS line is driven by the FTXL Transceiver. When it is high, it specifies that the FTXL Transceiver is busy with an I/O operation, and cannot accept new read/write requests. When it is low, the FPGA device can access the data bus. It is possible for the state of the HS line to change before the CS~ pin becomes inactive.

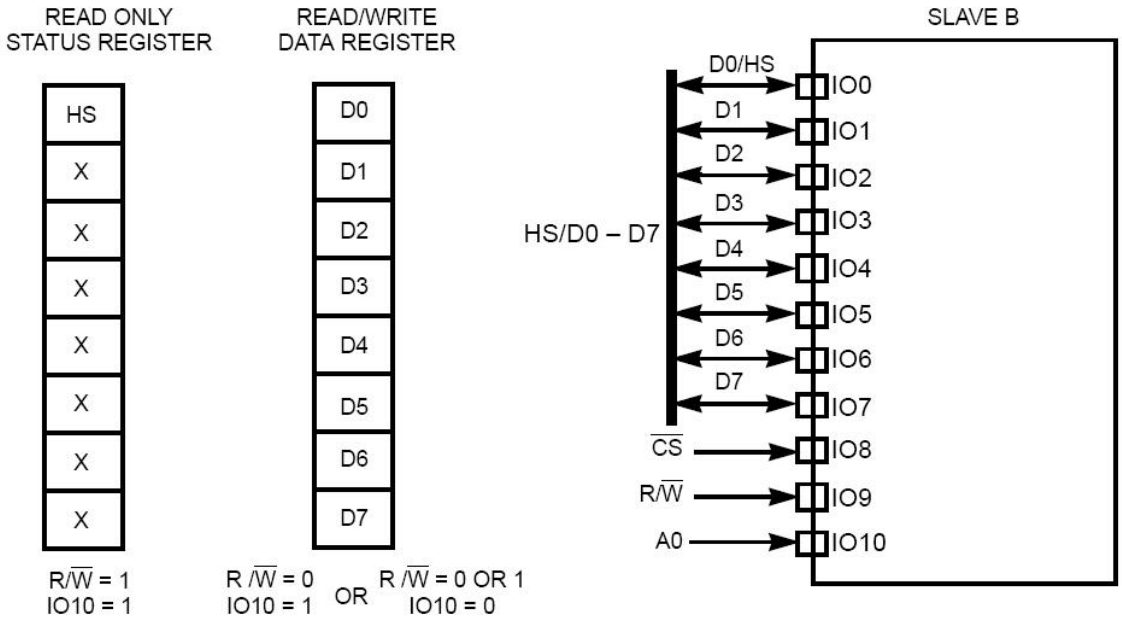


Figure 7. The FTXL Transceiver Parallel Interface

From the point of view of the host processor, the FTXL Transceiver appears as a memory-mapped parallel I/O device with eight data bits and three control bits. The FTXL LonTalk protocol stack communicates with the FTXL Transceiver through two logical registers: an 8-bit read/write data register and a 1-bit read-only status register. The FTXL LonTalk protocol stack reads the status register's HS bit before every read or write.

Token Passing and Handshaking

To eliminate the possibility of data bus contention, the FTXL LonTalk protocol stack (the master) and the FTXL Transceiver pass a virtual token between them to indicate which one of them can write to the data bus. Only the owner of the token can write to the data bus. After the initial device synchronization (or device resynchronization), the FTXL LonTalk protocol stack owns the token.

Whenever the FTXL LonTalk protocol stack reads data from the bus or writes data to the bus, the FTXL Transceiver sets the HS signal high. Conversely, whenever the FTXL Transceiver reads or writes data, it sets the HS signal low. That is, when the FTXL LonTalk protocol stack owns the token, it waits for the HS signal from the FTXL Transceiver before it writes data to the bus, and when the FTXL Transceiver owns the token, the host program monitors the low transition of the HS signal before it reads the bus.

The FTXL link-layer serial driver, included with the FTXL LonTalk protocol stack, manages the token and the handshake protocol. Token ownership and handshaking are transparent to your FTXL application.

Transferring Data

The data transfer operation between the FTXL LonTalk protocol stack (the master) and the FTXL Transceiver uses the virtual write token passing protocol.

The FTXL LonTalk protocol stack and the FTXL Transceiver pass the write token alternatively between themselves on the bus in an infinite ping-pong fashion. The owner of the token has the option to write a series of data bytes, or alternatively, pass the write token without any data.

The owner of the token can transfer up to 255 bytes of data. The FTXL LonTalk protocol stack reads the HS bit of the status register prior to reading or writing each data byte. The token owner keeps possession of the token until all data bytes have been written, after which it passes the token to the attached device.

The other device can then repeat the same process or it can pass the token back without any data.

The FTXL LonTalk protocol stack reads or writes data by first asserting the CS~ signal, then deasserting it. For a host read operation, the assertion causes the FTXL Transceiver to put data on the bus so that the FTXL LonTalk protocol stack can read it. For a host write operation, the assertion causes the FTXL Transceiver to read data on the bus and store it in its input buffer. In both cases, the data is latched on the rising edge of the CS~ signal.

FTXL Transceiver Pin Characteristics

The FTXL Transceiver includes the following sets of pins:

- Eight I/O pins
- An interrupt request (IRQ) pin
- A reset pin
- A service pin
- Clock pins
- LONWORKS network I/O pins

The following sections describe the characteristics of the I/O, IRQ, Reset~, Service~, and clock pins. See the *FT 3120 / FT 3150 Smart Transceiver Data Book* for information about other pins, including the requirements for the V_{DD} and ground pins.

I/O Pins

The I/O pins (IO0-IO10) have a standard sink capability (1.4 mA @ 0.4 V), and have TTL level inputs with hysteresis.

Because the CS~ line (IO8) is asynchronous, it should be kept as noise-free as possible. For example, you should add a 100 pF debounce capacitor to this line.

Important: To ensure that noise-levels for all communications lines between the FPGA device and the FTXL Transceiver are kept to a minimum, you should ensure that the FPGA device and FTXL Transceiver are separated by no more than 10 cm on the FTXL device's PCB.

IRQ Pin

The IRQ pin (pin 24) is an output pin. For an FT 3120 Smart Transceiver, this pin is the CP3 Sleep~ pin. However, for an FTXL 3190 Free Topology Smart

Transceiver, this pin is not part of the communications port, and does not function as a sleep control pin, but acts as in interrupt control pin.

The FTXL LonTalk protocol stack uses the IRQ pin to receive an indication from the FTXL Transceiver that the network is ready, either for uplink or for downlink. The FTXL LonTalk protocol stack asserts the IRQ pin high to cause the interrupt.

The *downlink ready* interrupt allows the FTXL Transceiver to inform the host when it has read the first byte of the transfer. This interrupt accounts for the latency of the parallel interface, that is, between a host write for a downlink transfer and the FTXL Transceiver read for the transfer. This latency is on the order of 110 microseconds (for an FTXL Transceiver running at 20 MHz), but it could be longer if the FTXL Transceiver is busy processing an incoming network frame. The FTXL LonTalk protocol stack initiates a downlink transfer by writing only the length byte; it then lets the interrupt service routine handle the rest of the transfer.

The *uplink ready* interrupt is an indication from the FTXL Transceiver that uplink traffic needs to be transferred. The IRQ pin is asserted only when the FTXL Transceiver does not own the write token.

The IRQ pin is deasserted during the downlink activity.

Although there are two interrupt cases, there is only a single interrupt request line. The interrupt type is determined by the FTXL LonTalk protocol stack based on the state of the FTXL Transceiver and token ownership.

Reset Pin

The RESET~ pin (pin 40) is both an input and an output. As an input, the RESET~ pin is internally pulled high by a current source acting as a pull-up resistor. The RESET~ pin becomes an output when any of the following events occur:

- Software reset
- Watchdog Timer event
- Internal low-voltage inhibit (LVI) circuit detects a low voltage
- RESET~ pin drops below the internal trip point

Important: The Nios II processor must be able to detect changes to and assert the FTXL Transceiver's RESET~ pin.

Reset Function

The reset function is a critical operation for any embedded microcontroller. In the case of an FTXL Transceiver, the reset function plays a key role in the following conditions:

- Initial V_{DD} power up
  Reset ensures proper initialization of the FTXL Transceiver.
- V_{DD} power fluctuations
  Reset manages proper recovery of the FTXL Transceiver after V_{DD} stabilizes.

- Program recovery
  If an application experiences unexpected behavior because of address or data corruption, a reset can recover.
- V_{DD} power down
  Reset ensures proper shutdown.
- Memory maintenance
  Reset helps protect the EEPROM from major corruption.

The FTXL Transceiver has four mechanisms to initiate a reset:

- The RESET~ pin is asserted (pulled low) and then deasserted (returned high).
- A software reset command from the parallel interface driver within the FTXL LonTalk protocol stack.
- A watchdog timeout occurs during application execution (the timeout period is approximately 840 ms when running at 10 MHz and approximately 210 ms at 40 MHz; this figure scales inversely with clock frequency).
- The LVI circuit detects a drop in the power supply below a set level.

During any of the reset functions, when the RESET~ pin is asserted (in the low state), the FTXL Transceiver enters the following states:

- The oscillator continues to run
- All processor functions stop
- The SERVICE~ pin goes to high impedance
- The I/O pins go to high impedance

When the RESET~ pin is released back to a high state, the FTXL Transceiver begins its initialization procedure.

Power-Up Sequence

During power up sequences, the RESET~ pin should be held low until the power supply is stable, to prevent start-up malfunctioning. Likewise, when powering down, the FTXL Transceiver RESET~ pin should be asserted before the power supply goes below the minimum operating voltage of the FTXL Transceiver.

Important: If proper reset recovery circuitry is not used, the FTXL Transceiver can go applicationless. The applicationless state occurs when the checksum error-verification routine detects a corruption in memory; this corruption can be falsely detected because of an improper reset sequence or noise on the power supply. In general, the applicationless state is an unrecoverable state, so be sure that your device's reset circuitry is correct.

The total capacitance directly connected to the RESET~ pin, including stray and external device input capacitance, must not exceed 1000 pF. This limit ensures that the FTXL Transceiver can successfully output a reset down to below 0.8 V. The 100 pF minimum capacitance is required for noise immunity.

Because the FTXL LonTalk protocol stack internally handles most reset events, you do not generally need an external Reset button or switch connected to the RESET~ pin.

Software Controlled Reset

When the CPU watchdog timer expires, or a software command to reset occurs, the RESET~ pin is asserted (pulled low) for 256 CLK1 clock cycles. The RESET~ pin external capacitor ($100 \leq C_E \leq 1000$ pF) begins charging and provides the required duration of reset.

Watchdog Timer

The FTXL Transceiver is protected against malfunctioning software or memory faults by three watchdog timers, one for each processor that makes up the Neuron core. If the system software fails to reset these timers periodically, the entire FTXL Transceiver automatically resets. The watchdog period is approximately 840 ms when running at 10 MHz, approximately 210 ms at a 40 MHz input clock rate, and scales inversely with the input clock rate.

LVI Considerations

The FTXL Transceiver includes an internal LVI to ensure that it only operates above the minimum voltage threshold. See the *FT 3120 and FT 3150 Smart Transceiver Datasheet* for LVI trip points.

Reset Processes and Timing

During the reset period, the I/O pins are in a high-impedance state. The data lines are undetermined but driven high or low, so that they do not float and draw excess current. The SERVICE~ pin is high impedance during reset. After the RESET~ pin is released, the FTXL Transceiver performs hardware and firmware initialization before communicating with the host processor.

Service Pin

The service pin (pin 5) function for an FTXL device is controlled by the host processor. The SERVICE~ pin on the FTXL Transceiver is not used.

Clock Pins

The FTXL Transceiver operates with an input clock of 5, 10, 20, or 40 MHz. The FTXL Transceiver divides the input clock by a factor of two to provide a symmetrical on-chip system clock. The input clock can be generated either by an external free-running oscillator or by the on-chip oscillator in the FTXL Transceiver using an external parallel-mode resonant crystal.

The accuracy of the input clock frequency of the FTXL Transceiver must be ±200 ppm or better; this requirement can be met with a suitable crystal, but cannot generally be met with a ceramic resonator.

The FTXL Transceiver includes an oscillator that can be used to generate an input clock using an external crystal. For 5 MHz, 10 MHz, and 20 MHz, either an external clock source or the on-chip crystal oscillator can be used. For 40 MHz operation, an external oscillator must be used.

When an externally generated clock is used to drive the CLK1 CMOS input pin of the FTXL Transceiver, CLK2 must be left unconnected or used to drive no more than one external CMOS load. The accuracy of the clock frequency must be ±0.02% (200 ppm) or better, to ensure that devices can correctly synchronize their bit clocks.

Figure 8 shows the crystal oscillator circuit. Use the load capacitance and resistor values recommended by the manufacturer of the crystal for this circuit. A 60/40 duty cycle or better is required when using an external oscillator. An external oscillator must provide CMOS voltage levels to the CLK1 pin.

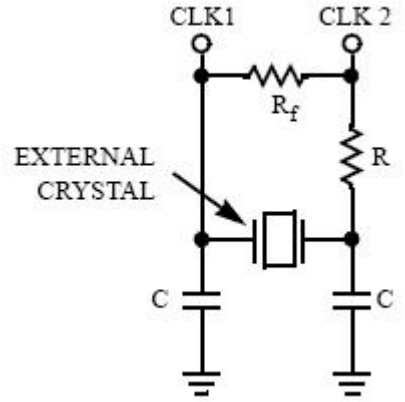


Figure 8. FTXL Transceiver Clock Generator Circuit

The FTXL Transceiver was designed to run at frequencies up to 40 MHz using an external clock oscillator. External oscillators generally take several milliseconds to stabilize after power-up. The FTXL Transceiver operating at 40 MHz must be held in reset until the externally-generated CLK input is stable, so an external power-on-reset-pulse stretching LVI chip/circuit is required. See the specification for the oscillator for more information about startup stabilization times.

FPGA Pin Assignments for the FTXL Transceiver

The standard design that is included with the FTXL Developer's Kit defines the pins for communications with the FTXL Transceiver Chip, as listed in **Table 12**.

For your own FPGA design, you can assign the pins to any appropriate pins using the Quartus II Pin Planner tool.

Table 12. FPGA Pin Assignments for the FTXL Transceiver

FPGA Design Pin Name	FPGA Pin Number	Direction	Edge Capture	Corresponding FTXL Transceiver Pin
FTXL_IRQ	E20	Input	Rising edge	Pin 24 (CP3/IRQ)
FTXL_SERVICE_PIN	U1	Input	Either edge	Pin 5 (SERVICE~)

FPGA Design Pin Name	FPGA Pin Number	Direction	Edge Capture	Corresponding FTXL Transceiver Pin
FTXL_RESET	G16	Bidirectional	Falling edge	Pin 40 (RESET~)
FTXL_SERVICE_LED	U8	Output	N/A	N/A
FTXL_AO	H14	Output		Pin 27 (IO10/A0)
FTXL_CS	H15	Output		Pin 31 (IO8/CS~)
FTXL_RW	H16	Output		Pin 30 (IO9/R/W~)
FTXL_D0	P15	Bidirectional	Either edge	Pin 4 (IO0/D0/HS)
FTXL_D1	J14	Bidirectional		Pin 3 (IO1/D1)
FTXL_D2	F14	Bidirectional		Pin 2 (IO2/D2)
FTXL_D3	J15	Bidirectional		Pin 43 (IO3/D3)
FTXL_D4	F15	Bidirectional		Pin 42 (IO4/D4)
FTXL_D5	H17	Bidirectional		Pin 36 (IO5/D5)
FTXL_D6	E18	Bidirectional		Pin 35 (IO6/D6)
FTXL_D7	G17	Bidirectional		Pin 32 (IO7/D7)

Notes:

- Signal direction is from the point of view of the FPGA device.
- All pins use the 3.3 V low-voltage transistor-transistor logic (LVTTL) I/O standard.
- The FTXL_AO and the FTXL_D[7..0] pins share an Avalon tri-state bridge.

The FPGA design also defines the pin assignments for the clock signals, SDRAM controller, and CFI flash interface controller. These pin assignments apply specifically to the DBC2C20 development board; your design will have different hardware requirements, and thus define different pin assignments for these functions.

Control Flow: Host Receiving Data from the FTXL Transceiver

When the FTXL Transceiver is ready to send an uplink message to the host program, it asserts the IRQ pin. The host then asserts the A0 pin to read the handshake bit, and deasserts the pin to read data bit 0. The transceiver receives the write token after the host writes a message (or a null token). The transceiver informs the host that it has data to send by asserting IRQ. The host asserts CS~

and A0, and waits for the assertion of D0/HS, and then deasserts A0, asserts CS~ and reads D0-D7 to receive the data.

Figure 9 shows an overview example logic analyzer trace[1] of the timing control flow when the host receives data from the FTXL Transceiver. In this example, the host receives a query status request.

See Appendix A, *Using the Bring-Up Application to Verify FTXL Hardware Design*, on page 63, for more detailed information about verifying the communications between the host processor and the FTXL Transceiver.

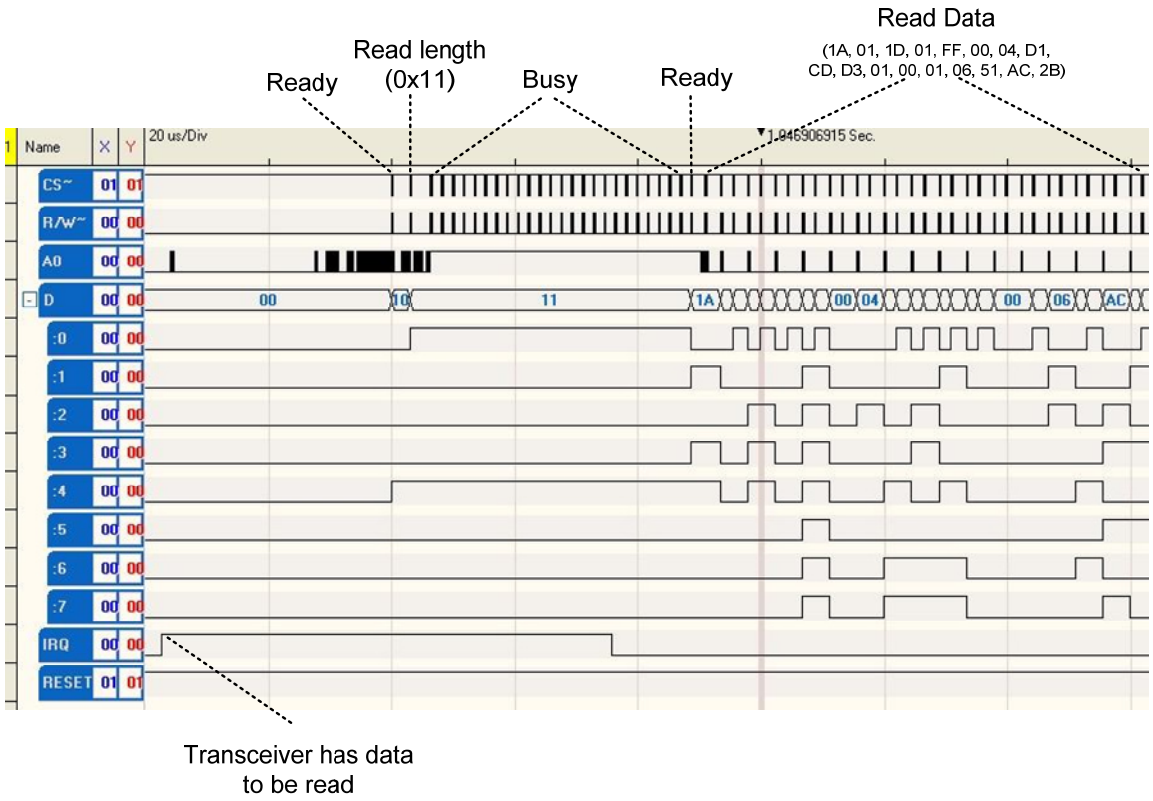


Figure 9. Overview Timing Diagram: Host Receives Data from the FTXL Transceiver

Figure 10 on page 31 shows a more detailed timing diagram for reading the length byte. The diagram also shows the read handshake.

[1] The logic analyzer traces were generated using the TechTools DigiView™ Logic Analyzer.

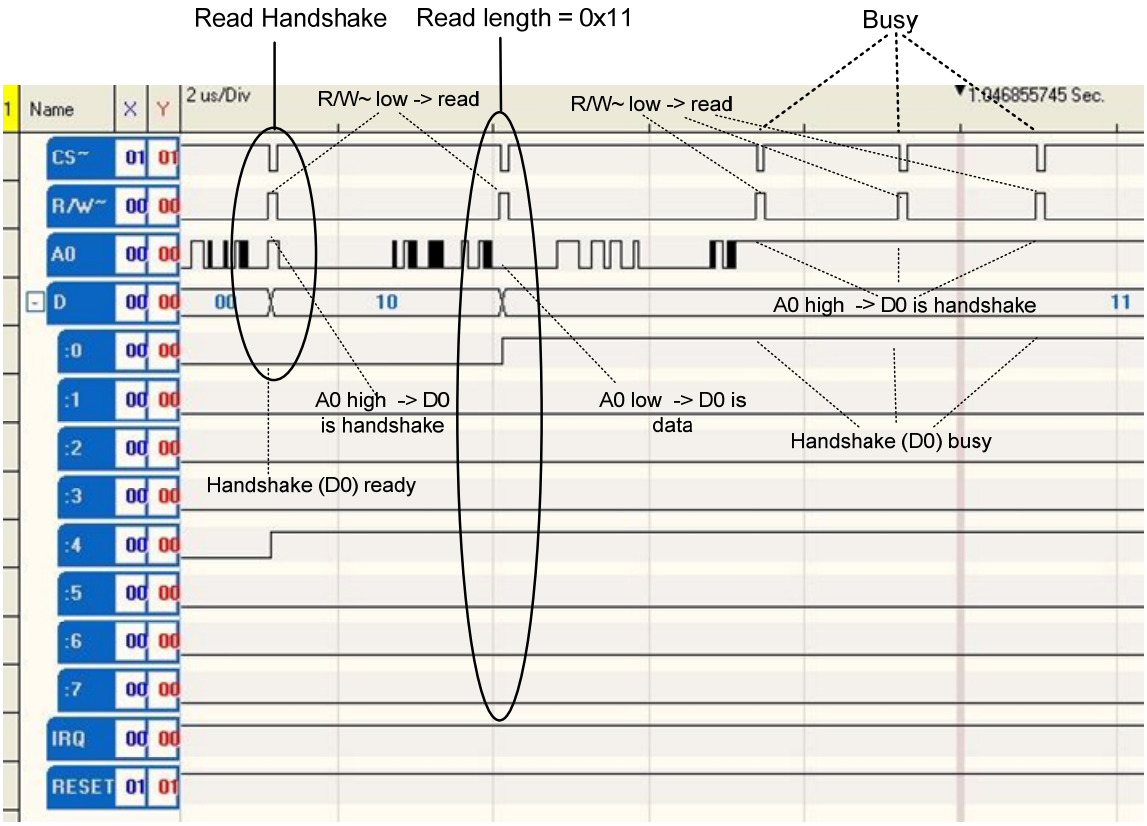


Figure 10. Timing Diagram for Reading the Length Byte

Figure 11 shows a detailed timing diagram for reading the data. The figure also shows the read handshake for each byte of data. However, the figure shows only the first three bytes of the data.

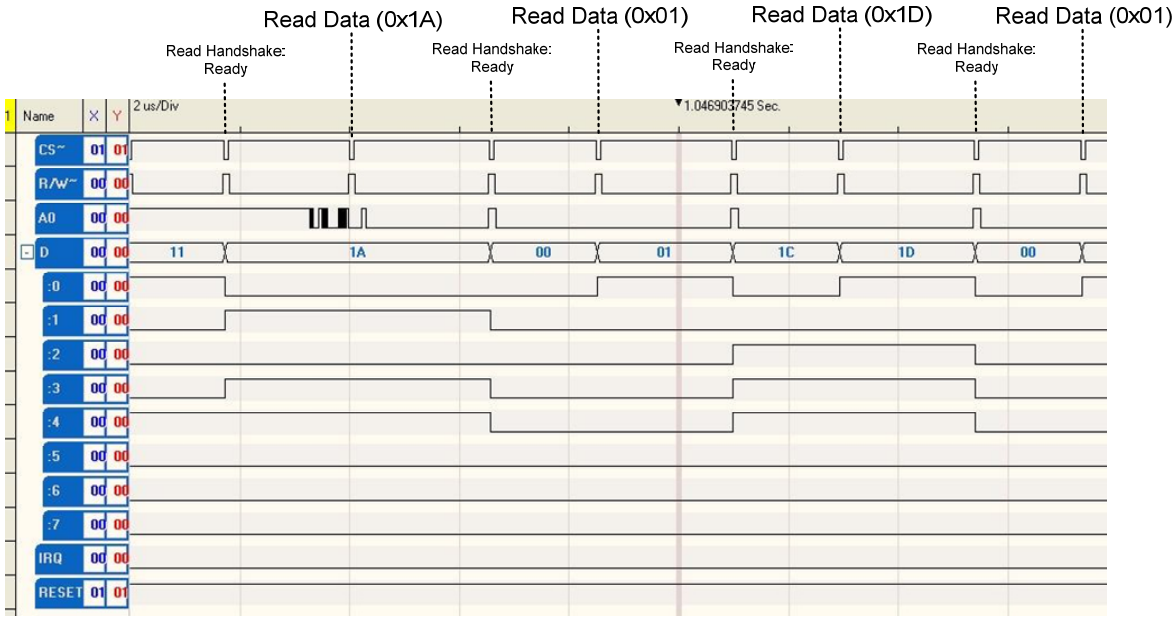


Figure 11. Timing Diagram for Reading Data

Control Flow: Host Sending Data to the FTXL Transceiver

When the host program is ready to send a downlink message to the FTXL Transceiver, it asserts the A0 pin. It receives the write token after the transceiver has sent a complete message, or passed a null token. The transceiver asserts IRQ after it has received the first byte of the message (the length byte), and is ready to receive the rest of the bytes (in fast-I/O mode). It does not assert IRQ if the host sends the null token.

Note: The transceiver never holds onto the token; when it gets the token, it either writes a message or passes the token back to the host.

Figure 12 shows an overview example logic analyzer trace of the timing control flow when the host sends data to the FTXL Transceiver. In this example, the host sends a service-pin message.

See Appendix A, *Using the Bring-Up Application to Verify FTXL Hardware Design*, on page 63, for more detailed information about verifying the communications between the host processor and the FTXL Transceiver.

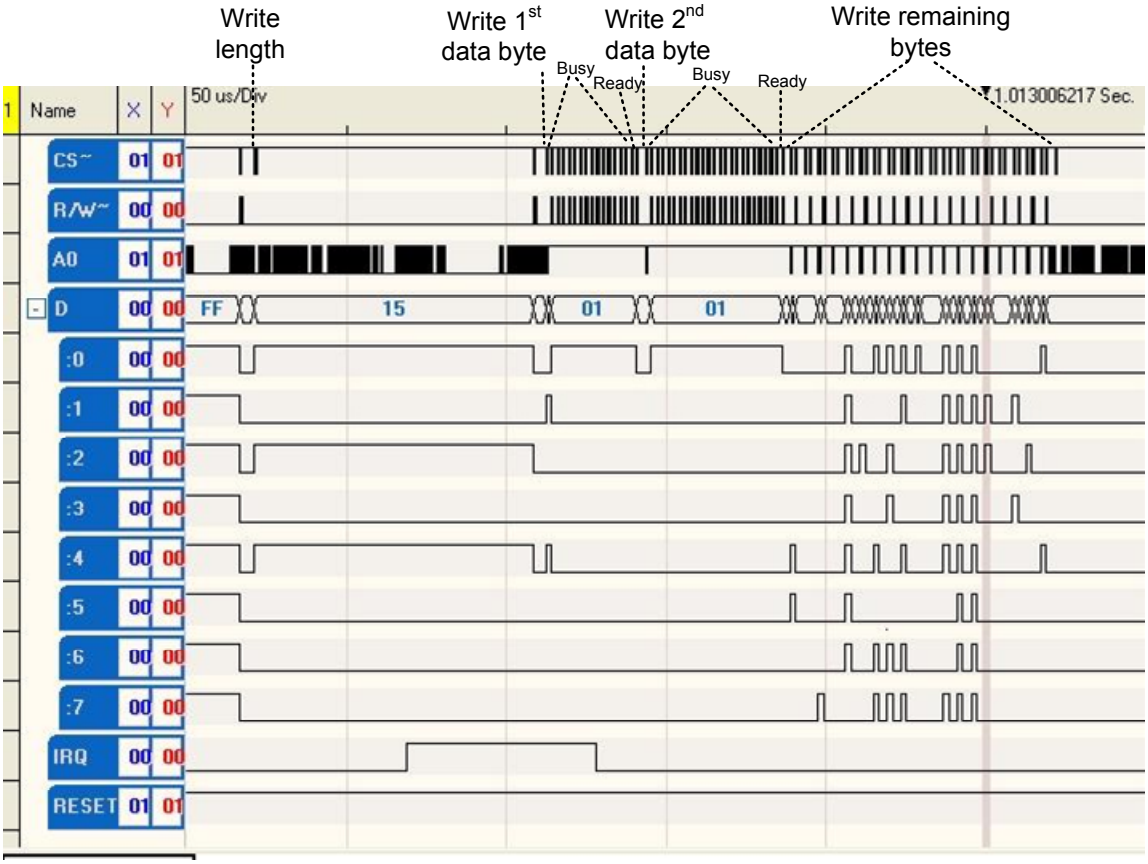


Figure 12. Overview Timing Diagram: Host Sends Data to the FTXL Transceiver

Figure 13 on page 33 shows a more detailed timing diagram for writing the length byte for the service-pin message. The diagram also shows the read handshake.

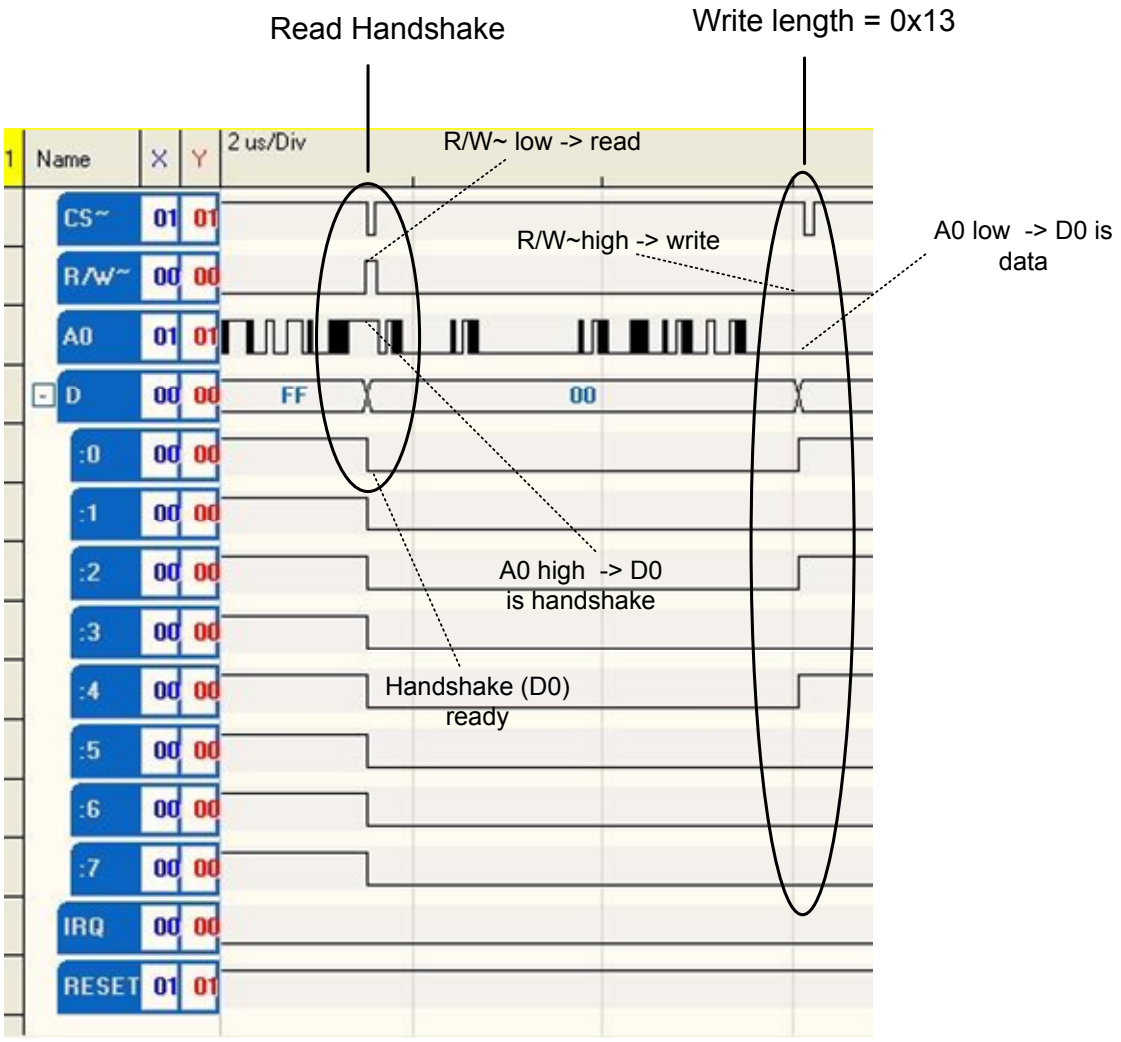


Figure 13. Timing Diagram for Writing the Length Byte

Figure 14 on page 34 shows a detailed timing diagram for writing the first two bytes of the data. The figure also shows the read handshake for each byte of data.

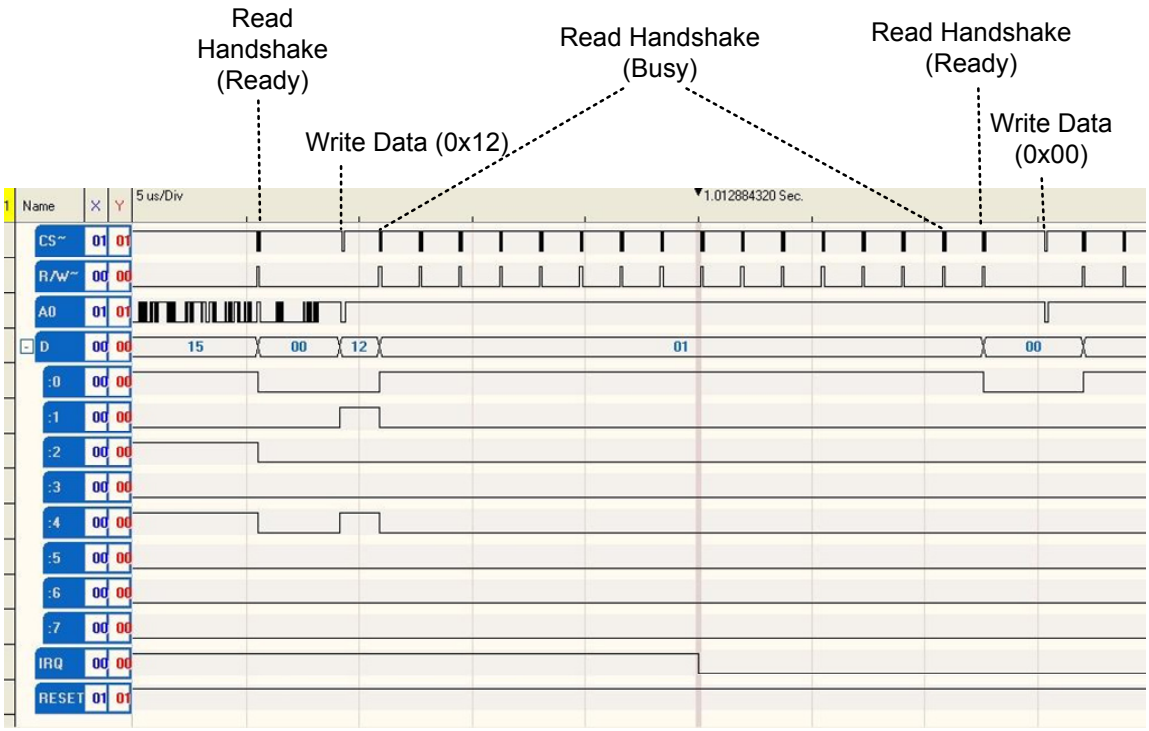


Figure 14. Timing Diagram for Writing the First Two Bytes of Data

Figure 15 shows a detailed timing diagram for writing the remaining bytes of the data for the service-pin message.

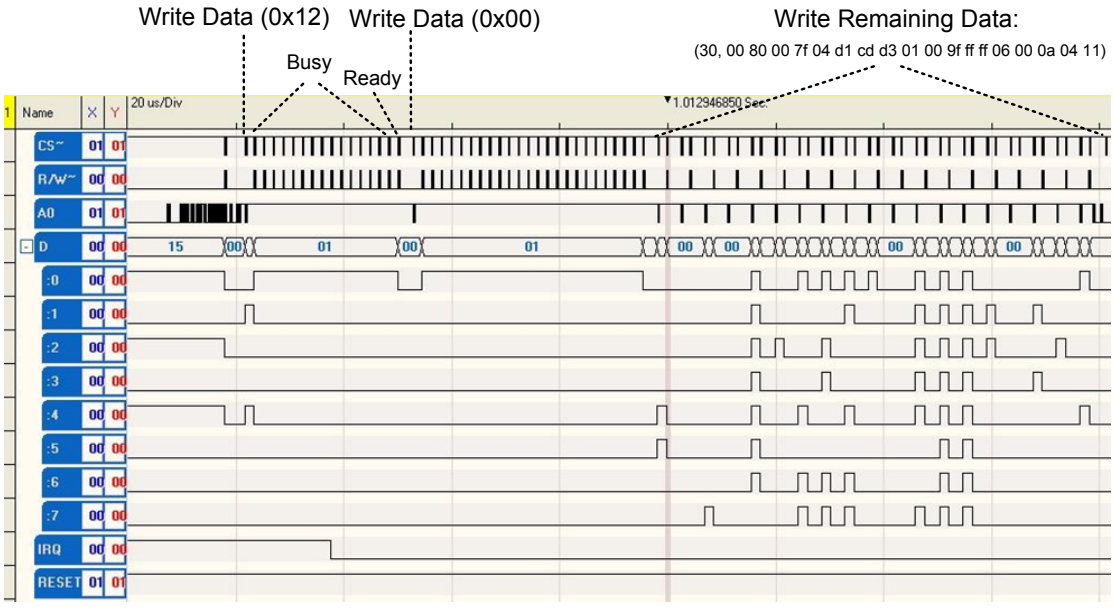


Figure 15. Timing Diagram for Writing Remaining Data

Figure 16 on page 35 shows an overview timing diagram for writing all of the data bytes for the service-pin message, including the last byte.

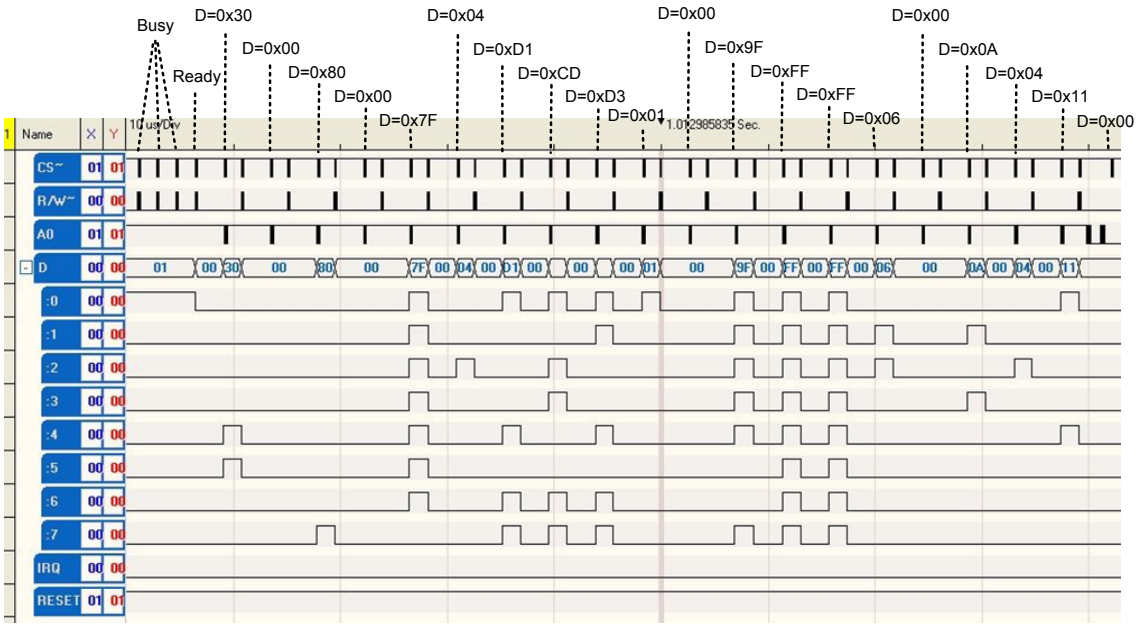


Figure 16. Timing Diagram for Writing a Service-Pin Message

4

FPGA Design for the FTXL Transceiver

This chapter describes FPGA design considerations for an FTXL device.

Overview

The hardware for an FTXL device consists primarily of an FTXL Transceiver Chip and an Altera FPGA device. When designing your FTXL device, you can use the reference design that is included with the FTXL Developer's Kit or you can create your own FPGA design and include the required FTXL components.

Using the Reference Design

The FTXL Developer's Kit provides a reference design for the FPGA hardware design. You can use this design as is for an FTXL device that uses the same underlying hardware as the DBC2C20 development board, or you can modify the design for a different development board or production device.

The reference design is located in the [*NiosEDS*]**\examples\vhdl\DBC2C20_FTXL\Standard** directory, where [*NiosEDS*] is the directory to which you installed the Nios II Embedded Design Suite (EDS), usually **C:\altera\72\nios2eds**. You should create a backup copy of this design before modifying it.

To work with the reference design, you must use the Altera Quartus II software, version 7.2 or later (either the Web Edition or the Subscription Edition). Within the Quartus II software, you must:

- Set up the USB Blaster download cable so that you can load the design into the FPGA device on the DBC2C20 development board; see *Using a Device Programmer for the FPGA Device* on page 55.
- Add the FTXL and DBC2C20 components to the global or project search path; see *Setting Component Search Paths* on page 56.
- If you modify the reference design, recompile the project; see *Building the Application Image* on page 61.
- Load the design into the FPGA device on the DBC2C20 development board; see *Loading the Application Image into the FPGA Device* on page 62.

Developing a New FPGA Design

During development of a new FPGA design for an FTXL Transceiver, your design needs to address the following considerations:

- Which type of physical FPGA device to use
- Which type of Nios II processor to use
- Which FPGA configuration device to use
- How to design for the FTXL parallel interface
- How to modify the FTXL hardware abstraction layer (HAL) for the software project

This chapter addresses these considerations for an FTXL device.

To develop a new FPGA design, you must use the Altera Quartus II software, version 7.2 or later (either the Web Edition or the Subscription Edition). Within the Quartus II software, you must:

- Add the FTXL components to the global or project search path; see *Setting Component Search Paths* on page 56.
- Add the FTXL components to the project; see *Adding FTXL Components to an Existing Design* on page 57.
- Compile the project; see *Building the Application Image* on page 61.
- Load the design into the FPGA device; see *Loading the Application Image into the FPGA Device* on page 62.

In addition, your design also needs to address other hardware design considerations for the FTXL 3190 Smart Transceiver Chip. These other considerations are not described in this book; see the *FT 3120 / FT 3150 Smart Transceiver Data Book* for information about these other considerations.

FPGA Device Requirements

The FTXL Transceiver reference design can use a Cyclone II or Cyclone III FPGA device for its Nios II host processor. **Table 13** lists the FPGA devices supported as host FPGA devices for an FTXL Transceiver.

Table 13. Supported Cyclone II and Cyclone III FPGA Devices

Cyclone II FPGA Device	Cyclone III FPGA Device	Supported for FTXL?
EP2C5	EP3C5	Yes (see note)
EP2C8	EP3C10	Yes
EP2C15	EP3C16	Yes
EP2C20	EP3C25	Yes
EP2C35	EP3C40	Yes
EP2C50	EP3C55	Yes
EP2C70	EP3C80	Yes
	EP3C120	Yes
Note: A Cyclone II EP2C5 FPGA device or a Cyclone III EP3C5 FPGA device might not have a sufficient number of logic elements (LEs) available for an FTXL device design. The EP2C5 FPGA provides 4608 LEs and the EP3C5 provides 5136 LEs, whereas a simple FTXL device (such as the FTXL Transceiver reference design) could require more than 3500 LEs.		

The FTXL Developer's Kit reference design uses an EP2C20 Cyclone II FPGA device. Which FPGA device your FTXL device uses depends on your requirements: including the number of logic elements (LEs), embedded

multipliers, embedded memory blocks, phase-locked loops (PLLs), and high-speed differential I/O channels.

See the Altera *Cyclone II Device Handbook* or *Cyclone III Device Handbook* for more information about these FPGA devices.

Although the FTXL Transceiver has not been tested with an Altera Stratix®, Stratix GX, or Arria™ GX FPGA device, there is no restriction within the FTXL hardware or software design that prevents your FTXL device from using one of these types of FPGA device. Likewise, there is no restriction on porting your FTXL device to an Altera Hardcopy® II or Hardcopy Stratix structured ASIC device.

Nios II Processor

The FTXL Developer's Kit reference design uses a Nios II/s processor with the following characteristics:

- Embedded multipliers
- No hardware divide
- Reset vector set to the cfi_flash at offset 0x0
- Exception vector set to the sdram at offset 0x20
- 4 KB cache for the instruction master
- Bursts disabled for the instruction master
- No tightly coupled instruction master ports
- Default settings for the data master
- No advanced features selected
- JTAG Level 1
- No custom instructions

Your FTXL device design can use either a Nios II/s or Nios II/f processor with similar settings as those of the FTXL Developer's Kit. However, the Nios II/e processor might not have sufficient resources for an FTXL device.

See the Altera *Nios II Processor Reference Handbook* for more information about the Nios II processor.

FPGA Configuration Device

The FTXL Developer's Kit reference design hardware uses an EPCS16 serial configuration device. Your FTXL device design can use any configuration device that is appropriate for your design's FPGA device.

In the Quartus II Device and Pin Options dialog, the configuration device uses the active serial configuration and generates compressed bit streams, as shown in **Figure 17** on page 41. The configuration device uses default settings.

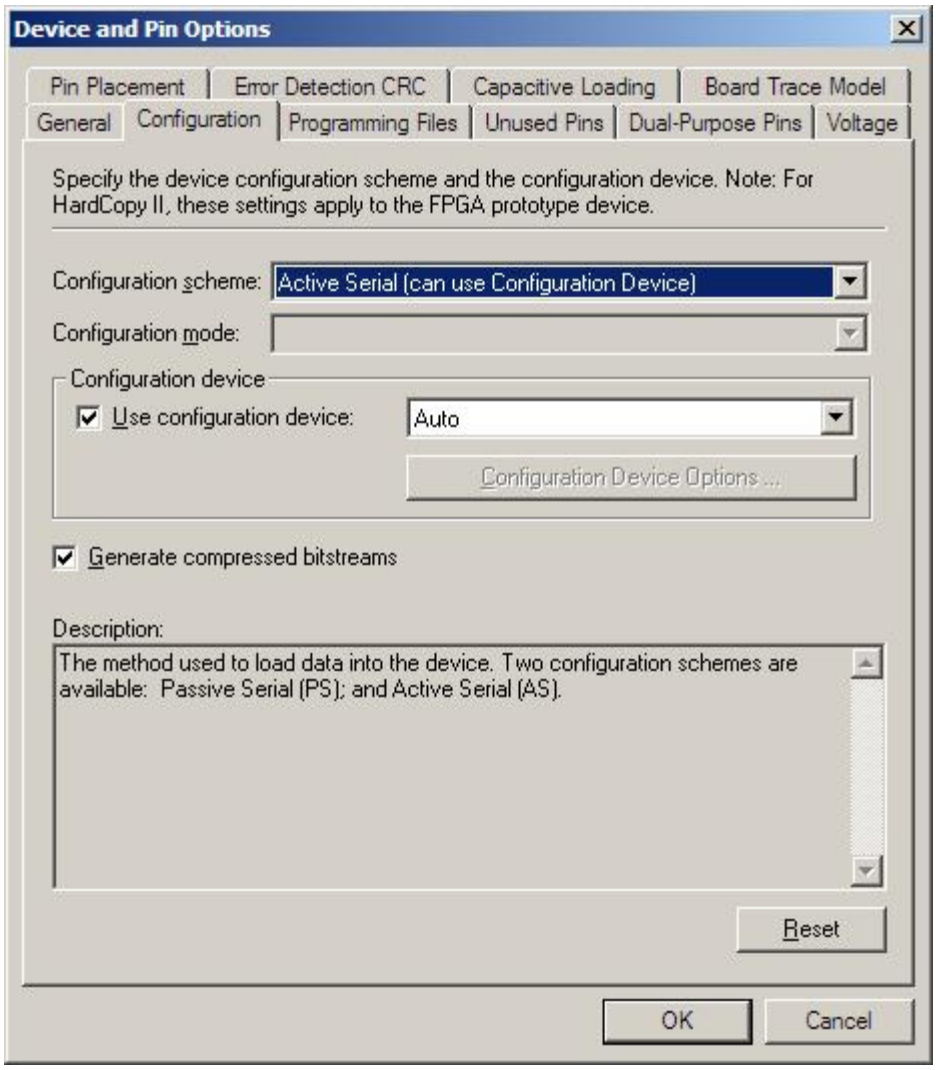


Figure 17. Quartus II Device and Pin Options Dialog

See the Altera *Configuration Handbook* for more information about FPGA configuration devices.

FTXL Components

The FTXL Developer's Kit includes components for the Altera SOPC Builder tool and the Quartus II software, as listed in **Table 14**. These components are installed to the [*NiosEDS*]**\components\FTXL** directory. An FTXL device must include all of these components.

Table 14. FTXL Components

Component Name	Class Name	File Names	Description
FTXL Parallel Interface Delay	FTXL_PIO_Delay	FTXL_PIO_Delay.bsf FTXL_PIO_Delay.bdf	Parallel interface timing delay circuit

Component Name	Class Name	File Names	Description
FTXL Parallel I/O Transceiver Interface	FTXL_PIO	FTXL_PIO_hw.tcl	Parallel I/O interface to the FTXL Transceiver
FTXL Service LED	FTXL_SERVICE_LED	FTXL_SERVICE_LED.vhd FTXL_SERVICE_LED_hw.tcl	Service LED for the FTXL device
FTXL Service Pin	FTXL_SERVICE_PIN	FTXL_SERVICE_PIN.vhd FTXL_SERVICE_PIN_hw.tcl	FTXL service pin input and interrupt
FTXL Transceiver Interrupt	FTXL_IRQ	FTXL_IRQ.vhd FTXL_IRQ_hw.tcl	Interrupt signal from the FTXL Transceiver
FTXL Transceiver Reset	FTXL_RESET	FTXL_RESET.vhd FTXL_RESET_hw.tcl	Bidirectional reset signal used to reset and detect reset on the FTXL Transceiver

None of these components has any parameters that you can set within the Altera SOPC Builder tool.

The following sections describe each of the FTXL components.

FTXL Parallel Interface Delay

Because the Cyclone II FPGA device has low delay times for its signals, the required timing for the parallel interface control port is achieved by adding a separate time-delay function to the Quartus II design. **Figure 18** shows the Quartus II design view of the FTXL parallel interface delay component.

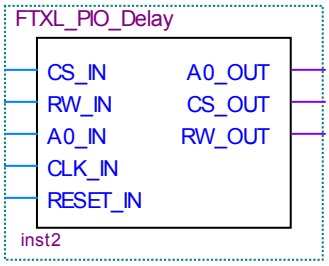


Figure 18. Quartus II Component for the FTXL Parallel Interface Delay

The delay circuit includes six D flip-flops, which provide the required delay for the three control signals:

- The CS~ signal is delayed by three clock cycles. It is also inverted by a NAND gate to ensure that it is active low.

- The A0 signal is delayed by two clock cycles.
- The R/W~ signal is delayed by one clock cycle.

Figure 19 shows the circuit detail for this component.

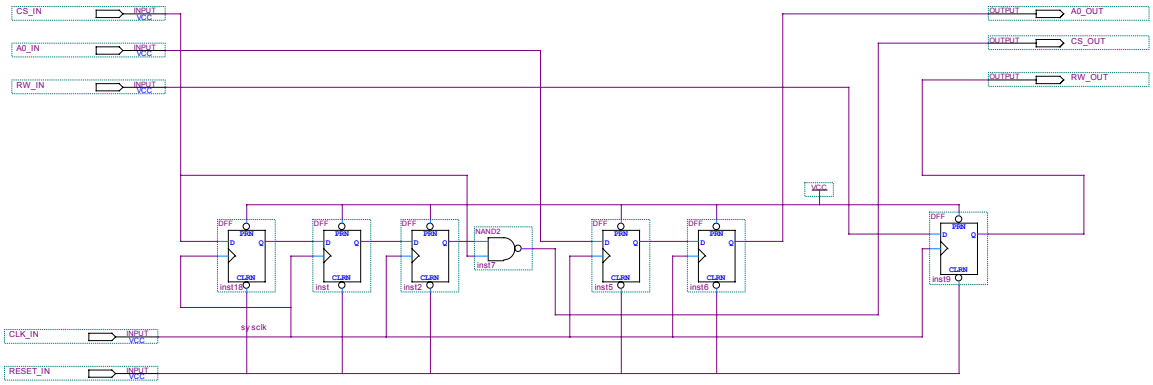


Figure 19. FTXL Parallel Interface Timing Delay Component

FTXL Parallel I/O Transceiver Interface

The FTXL parallel interface consists of a memory-mapped bus interface for an 8-bit data bus and three 1-bit control signals, as described in *The Parallel Communications Interface* on page 21.

The FTXL parallel I/O transceiver interface component defines the following signals, as shown in **Figure 20** on page 44:

- clk_out: Component clock signal
- clk: Nios II processor clock signal
- address: FTXL address (A0) signal, that controls the function of the IO0 pin, which can be part of the data I/O (as D0) or can be the handshake signal (as HS)
- data: FTXL data bus
- RW: FTXL read/write (R/W~) signal, which determines the direction of the bidirectional data bus
- CS: FTXL chip select (CS~) signal, which specifies whether a byte-transfer operation is in progress

Figure 20 on page 44 shows the definitions for the signals that comprise the FTXL parallel interface. The figure shows part of the Quartus II Component Editor dialog for this component, open to the **Signals** tab.

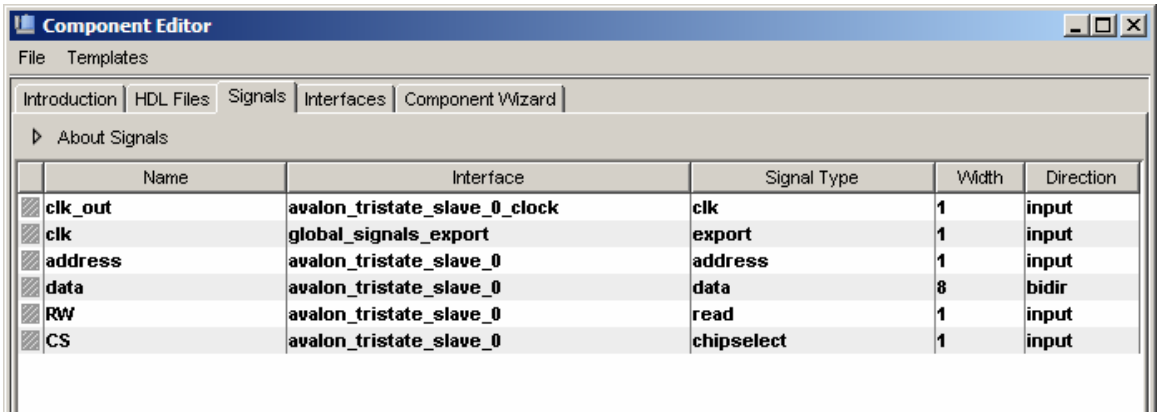


Figure 20. Quartus II Component Editor Dialog for FTXL Parallel I/O Signals

The data and control parts of the interface share a common Avalon tri-state bridge component. **Figure 21** on page 45 shows part of the Quartus II Component Editor dialog for this component, open to the **Interfaces** tab, with most of the interface definitions expanded. **Figure 22** on page 46 shows the same dialog, with the last interface expanded.

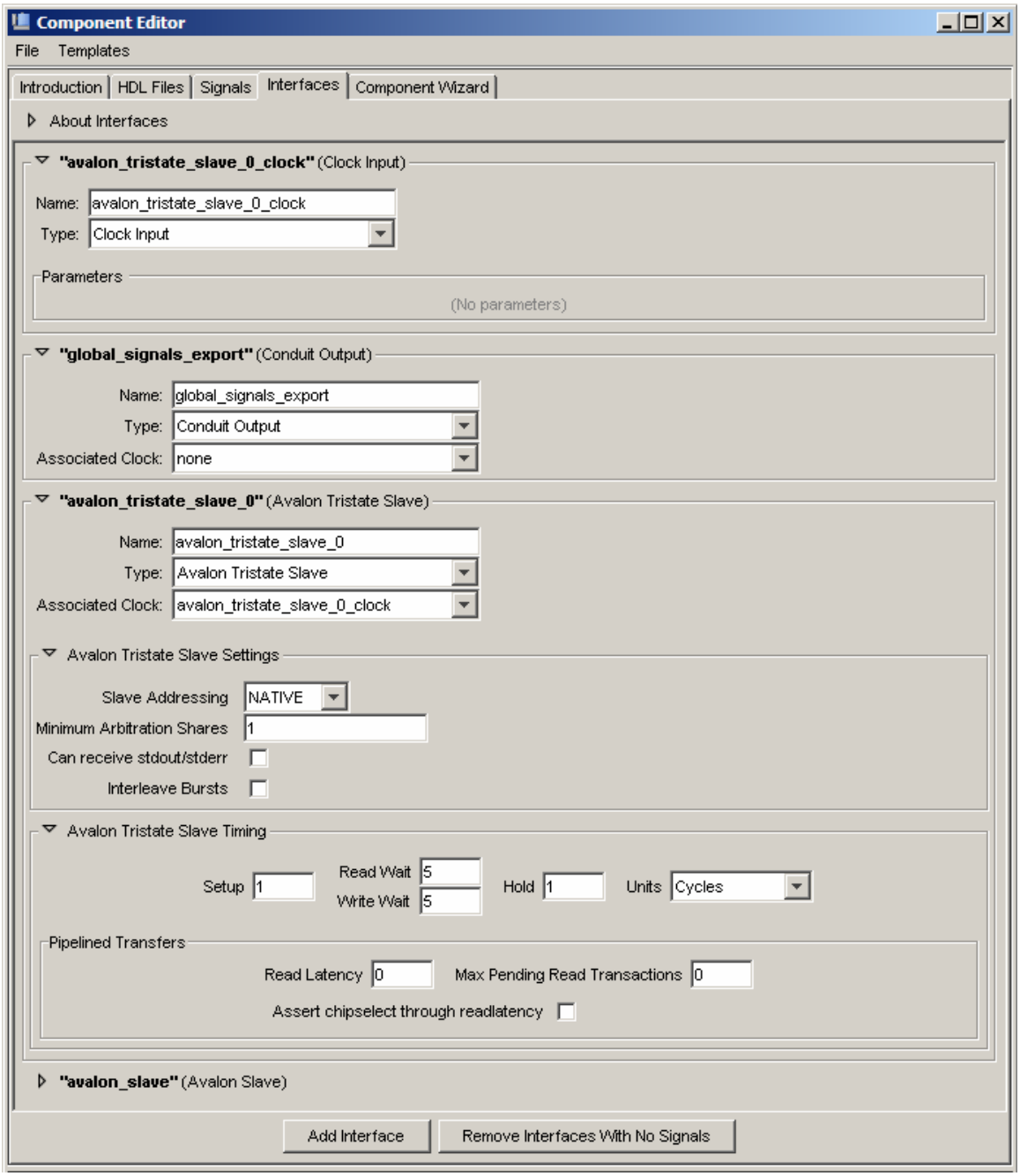


Figure 21. Quartus II Component Editor Dialog for FTXL Parallel I/O Interfaces (Part 1)

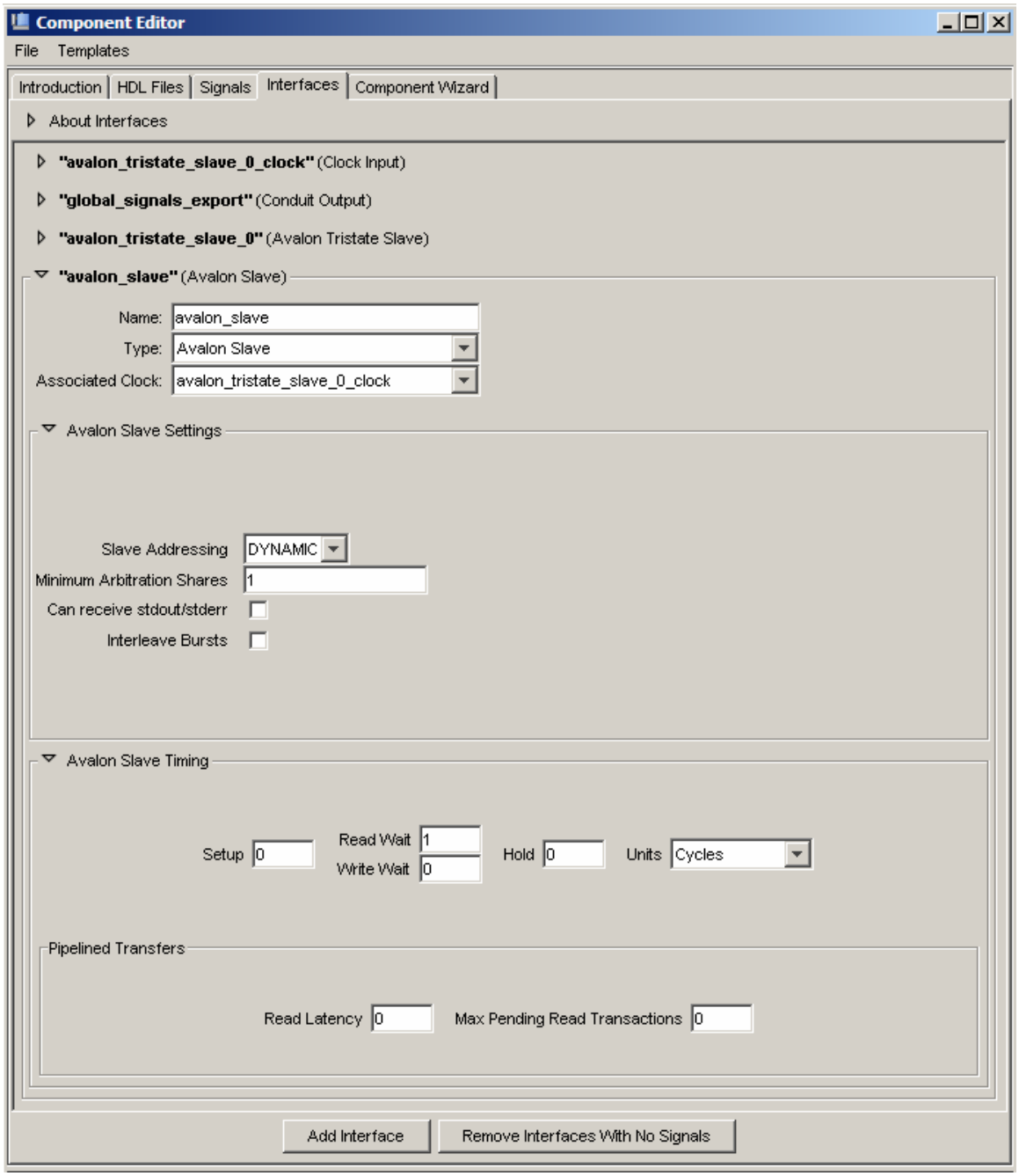


Figure 22. Quartus II Component Editor Dialog for FTXL Parallel I/O Interfaces (Part 2)

FTXL Service LED

The FTXL service LED component defines the signals needed for an FTXL device service-pin LED, including:

- clk: Nios II processor clock signal
- reset_n: Nios II processor reset signal
- address: 2-bit address signal for the LED

- chipselect: Select signal to enable access to the LED
- write_n: Write-select signal to write to the LED
- writedata: Data signal to the LED
- out_port: Output signal from the LED

FTXL Service Pin

The FTXL service pin component defines the signals needed for an FTXL device service-pin button, including:

- clk: Nios II processor clock signal
- reset_n: Nios II processor reset signal
- address: 2-bit address signal for the service pin button
- chipselect: Select signal to enable access to the service pin button
- write_n: Write-select signal to write to the service pin button
- writedata: Data signal to the service pin button
- readdata: Data signal from the service pin button
- irq: Interrupt signal from the service pin button
- in_port: Input signal to the service pin button

FTXL Transceiver Interrupt

The FTXL Transceiver interrupt component defines the signals needed for the FTXL Transceiver interrupt pin, including:

- clk: Nios II processor clock signal
- reset_n: Nios II processor reset signal
- address: 2-bit address signal for the interrupt pin
- chipselect: Select signal to enable access to the interrupt pin
- write_n: Write-select signal to write to the interrupt pin
- writedata: Data signal to the interrupt pin
- readdata: Data signal from the interrupt pin
- irq: Interrupt signal from the interrupt pin
- in_port: Input signal to the interrupt pin

FTXL Transceiver Reset

The FTXL Transceiver reset component defines the signals needed for the FTXL Transceiver reset pin, including:

- clk: Nios II processor clock signal
- reset_n: Nios II processor reset signal
- address: 2-bit address signal for the reset pin

- chipselect: Select signal to enable access to the reset pin
- write_n: Write-select signal to write to the reset pin
- writedata: Data signal to the reset pin
- readdata: Data signal from the reset pin
- bidir_port: Bidirectional signal for the reset pin

Phase-Locked Loop

If your FPGA design includes a phase-locked loop (PLL) component, be sure to connect one of its clock-out signals to the CLK_IN signal of the FTXL parallel I/O transceiver interface component. The CLK_IN signal requires the following characteristics for the PLL clock-out signal:

- Ratio: 1/1
- Phase shift: 0
- Clock duty cycle: 50%

If your FPGA design does not include a PLL component, be sure to connect the CLK_IN signal to an external oscillator with the same characteristics and the same clock speed as the Nios II processor clock.

DBC2C20 Components

Although the reference designs that are provided with the DBC2C20 development board include several SOPC Builder components, the FTXL Developer's Kit reference design uses only one of the SOPC Builder components for the DBC2C20 development board: the DBC2C20 SRAM interface. This component is installed to the [*NiosEDS*]**\components\DBC2C20_sram_interface** directory.

You do not need to install any of the SOPC Builder components for the DBC2C20 development board from the CD-ROM that accompanies the DBC2C20 development board.

Your FTXL device can include any appropriate SRAM components, but a design that uses the DBC2C20 development board must use the DBC2C20 SRAM interface component.

Timers

The FTXL Developer's Kit reference design includes a system timer (a 1 ms timer) and a high-resolution timer (a 1 µs timer). The high-resolution timer is not used by the FTXL software, but is included for applications that require a timer with higher resolution than the system clock.

External Memory

The FTXL Developer's Kit reference design includes the following external memory:

- 1 MB SRAM

- 8 MB CFI flash memory
- 16 MB SDRAM

These numbers correspond to the external memory provided by the DBC2C20 development board. Your FTXL device design can include as much external memory as required for your device.

Addressing, Size, and IRQ Requirements

For the FTXL Developer's Kit reference design, the address assignments for the instruction master and data master components were assigned by allowing the Altera SOPC Builder tool to allocate and assign the address locations.

Table 15 lists the size requirements of each of the major components of the FTXL Developer's Kit reference design. The size specifications for general components are the default sizes; the size specifications for the components whose names begin with **FTXL_** define the requirements for the hardware interface.

Table 15. Size Requirements for Components in the FTXL Reference Design

Component	Component Name	Size (Bytes)
FTXL Parallel I/O Transceiver interface	FTXL_PIO_inst	8
JTAG UART	jtag_uart	8
System ID	sysid	8
FTXL service LED	FTXL_SERVICE_LED_inst	16
FTXL service pin	FTXL_SERVICE_PIN_inst	16
FTXL Transceiver interrupt	FTXL_IRQ_inst	16
FTXL Transceiver reset	FTXL_RESET_inst	16
High-resolution timer	high_res_timer	32
System timer	sys_timer	32
EPCS serial flash controller port	epcs_controller	2 KB
JTAG debug module	cpu.jtag_debug_module	2 KB
SRAM interface for the DBC2C20 development board	DBC2C20_sram_interface_0	1 MB
CFI flash interface for the DBC2C20 development board	cfi_flash	8 MB

Component	Component Name	Size (Bytes)
SDRAM interface for the DBC2C20 development board	sdram	16 MB

Table 16 lists the interrupt request (IRQ) numbers for each of the components that can generate an interrupt. Your FTXL FPGA design can specify different IRQ numbers, as long as the components retain the same relative interrupt levels.

Table 16. IRQ Specifications for the FTXL Reference Design

Component	Component Name	IRQ Number
System timer	sys_timer	0
JTAG UART	jtag_uart	1
High-resolution timer	high_res_timer	2
EPCS serial flash controller port	epcs_controller	3
FTXL Transceiver interrupt	FTXL_IRQ_inst	11
FTXL service pin	FTXL_SERVICE_PIN_inst	13

FTXL Hardware Abstraction Layer

The FTXL software design provides a hardware abstraction layer (HAL) that acts as an interface between the Nios IDE-generated system library (primarily the **system.h** file) and the FTXL software. The FTXL HAL provides a number of abstract functions and macro definitions that allow portability among various FPGA designs.

When you create or modify the FPGA design and regenerate the Nios system library, you must modify the FTXL HAL to use the component names defined in your Nios system library. Thus, the FTXL LonTalk protocol stack library works with any underlying hardware, and you do not need to modify your FTXL host program when you modify the hardware design.

Recommendation: The FTXL Developer's Kit reference design defines a number of component names (names that begin with **FTXL_**) that you should retain in your design so that you do not need to modify the FTXL HAL. However, if you do modify these names, you must modify the FTXL HAL appropriately. If you add new components, you should define a new HAL for those components.

Important: Because the FTXL LonTalk protocol stack library calls the functions that the FTXL HAL defines, you can change the behavior of these functions, but not the function names of the programmatic interface.

Within the FTXL software, the FTXL HAL is defined in two files: **FtxlHal.h** and **FtxlHal.c**.

Other Hardware Design Considerations

The FTXL 3190 Free Topology Smart Transceiver Chip shares electrical and physical characteristics with the FT 3120 Smart Transceiver Chip. For information about hardware design for an FT 3120 Smart Transceiver, including PCB design considerations, EMC considerations, electrical characteristics of the chip, and networking requirements, see the *FT 3120 / FT 3150 Smart Transceiver Data Book*.

For other FPGA design considerations, see the Altera *Cyclone II Device Handbook* or the *Cyclone III Device Handbook*.

5

Working with the Altera Development Environments

This chapter describes how to use the Altera Complete Design Suite to build the hardware design and load it into the FPGA device.

Development Tools

To develop your FTXL application, you use version 7.2 or later of the Altera Complete Design Suite, as listed in **Table 17**. You can obtain the Altera Complete Design Suite on DVD-ROM from Altera Corporation, or you can download the Web Edition of the tools from https://www.altera.com/support/software/download/nios2/dnl-nios2.jsp.

Table 17. Altera Complete Design Suite

Quartus II Design Software for Windows
The Quartus II design software provides a suite of tools for system-level design, embedded software programming, FPGA and CPLD design, synthesis, place-and-route, verification, and device programming. Quartus II software supports all of Altera's current device families. The Quartus II Web Edition is a subset of the Quartus II design software that provides support for selected Altera processors. Both the Quartus II design software and the Quartus II Web Edition include the SOPC Builder tool, which is an automated system development tool that dramatically simplifies the task of creating high-performance system-on-a-programmable-chip (SOPC) designs.
ModelSim®-Altera VHDL & Verilog HDL Simulation Tool
The ModelSim-Altera software is an Altera-specific version of the Model Technology™ ModelSim simulation software, which supports behavioral simulation and testbenches for VHDL or Verilog hardware description languages (HDLs). The ModelSim-Altera software is included with Altera software subscriptions.
MegaCore IP Library
The MegaCore IP library includes some of Altera's most popular intellectual property (IP) cores, including a finite impulse response (FIR) compiler, a numerically controlled oscillator (NCO) compiler, a fast Fourier transform (FFT) compiler, several DDR SDRAM controllers, a QDRII SDRAM controller, an RLDRAM II controller, and a lightweight serial interconnect protocol. The MegaCore IP library is included with Altera software subscriptions.
Nios II Embedded Design Suite
The Nios II integrated development environment (IDE) is a graphical user interface (GUI) within which you can accomplish all Nios II embedded processor software development tasks, including editing, building, managing, and debugging embedded software programs. The Nios II IDE is included with Altera software subscriptions.

For more information about installing the Altera Complete Design Suite, see *Quartus II Installation & Licensing for Windows*, available from the Quartus II Development Software Literature page at www.altera.com/literature/lit-qts.jsp.

Using a Device Programmer for the FPGA Device

To load your hardware design, software application, and the FTXL LonTalk protocol stack, into the FPGA device, you can use a device programmer, such as the Altera USB-Blaster download cable, as described in **Table 18**.

Table 18. Device Programmer for the Nios II Processor

Altera USB-Blaster Download Cable
The USB-Blaster download cable interfaces to a standard PC USB port. This cable drives configuration or programming data from the PC to the device. For more information about the USB-Blaster, see the *USB-Blaster Download Cable User Guide*.

The Windows driver for the USB-Blaster is in the [*Altera*]**\quartus\drivers\usb-blaster** directory, where [*Altera*] is the directory in which you installed the Altera Complete Design Suite, usually **C:\altera\72**.

To set up the programming hardware in the Quartus II software:

1. Start the Quartus II software.
2. Select **Tools** → **Programmer** to open the Chain Description File (*.cdf) view for the project.
3. Click **Hardware Setup** to open the Hardware Setup window.
4. If you have already installed the Windows drivers for the USB-Blaster, it should appear in the **Available hardware items** area of the Hardware Setup window.
5. If the programming hardware that you want to use does not appear in the **Available hardware items** area of the Hardware Setup window, click the **Add Hardware** button to open the Add Hardware dialog.
    a. Select the appropriate programming cable or programming hardware from the **Hardware Type** dropdown list box.
    b. Select the appropriate port, baud rate, and server information, if necessary.
    c. Click **OK**.
6. Select the programming hardware that you want to use from the **Currently selected hardware** dropdown list box.
7. Click **Close** to close the Hardware Setup window.
8. Select **JTAG** from the **Mode** dropdown list box of the Chain Description File view for the project.
9. Select **File** → **Close** to close the Chain Description File view for the project.

You can save the Chain Description File (*.cdf) for use with other projects.

Setting Component Search Paths

To work with an FPGA design that includes FTXL components, you must add the components to the Quartus II and SOPC Builder library paths. In addition, for the FTXL reference design, you must add the DBC2C20 components to the library paths.

In the Quartus II software, you can add the FTXL components to the global library paths so that all projects can access the FTXL components, or you can add the FTXL components to the library path for a specific project.

To add FTXL components to the Quartus II global library path:

1. Start the Quartus II software.
2. Open a Quartus II project, such as the FTXL reference design.
3. Select **Tools** → **Options** to open the Options window.
4. In the Category area on the left side of the Options window, select **Global User Libraries (All Projects)**.
5. In the Global User Libraries (All Projects) page, click the browse button (**...**) to the right of the **Library name** field to open the Select Directory dialog.
6. In the Select Directory dialog, select the [*Altera*]**\nios2eds\components\FTXL** directory, and click **Open**.
7. In the Global User Libraries (All Projects) page, click **Add** to add the FTXL components to the Libraries area. Click **OK** to save the updated library path setting and close the window.

If you want to add the DBC2C20 components to the global library path, specify the [*Altera*]**\nios2eds\components\DBC2C20_sram_interface** directory in step 6.

To add FTXL components to a specific Quartus II project library path:

1. Start the Quartus II software.
2. Open a Quartus II project, such as the FTXL reference design.
3. Select **Assignments** → **Settings** to open the Settings window.
4. In the Category area on the left side of the Settings window, select **Libraries**.
5. In the Libraries page, click the browse button (**...**) to the right of the **Project library name** field to open the Select Directory dialog.
6. In the Select Directory dialog, select the [*Altera*]**\nios2eds\components\FTXL** directory, and click **Open**.
7. In the Libraries page, click **Add** to add the FTXL components to the Libraries area. Click **OK** to save the updated library path setting and close the window.

If you want to add the DBC2C20 components to the project library path, specify the [*Altera*]**\nios2eds\components\DBC2C20_sram_interface** directory in step 6.

You do not need to add components to the project library path if they are already in the global library path.

To add FTXL components to the SOPC Builder library path:

1. Start the Quartus II software.
2. Open a Quartus II project, such as the FTXL reference design.
3. Select **Tools → SOPC Builder** to open the Altera SOPC Builder tool.
4. In the Altera SOPC Builder tool, select **Tools → Options** to open the Options dialog.
5. In the Options dialog, select **IP Search Path** from the Category area.
6. In the IP Search Path Options page of the Options dialog, click **Add** to open the Open dialog.
7. In the Open dialog, select the [*Altera*]**\nios2eds\components\FTXL** directory and click **Open**.
8. Click **Finish** to save the updated search paths and close the Options dialog.

If you want to add the DBC2C20 components to the SOPC Builder search path, specify the [*Altera*]**\nios2eds\components\DBC2C20_sram_interface** directory in step 7. You might also have to specify the [*Altera*]**\quartus\bin** directory to allow the SOPC Builder to find standard Altera MegaCore IP Library functions.

Adding FTXL Components to an Existing Design

To add FTXL components to an existing FPGA device design, you must modify your SOPC Builder design for the project and you must modify your Quartus II block design file (*.bdf) design for the project.

After you modify both designs, you must regenerate the SOPC Builder design and recompile the Quartus II project. See *Building the Application Image* on page 61 for more information.

Modifying the SOPC Builder Design

Before you begin, ensure that the SOPC Builder IP search path includes the FTXL components; see *Setting Component Search Paths* on page 56.

This section assumes that you have already added a Nios II processor and related hardware components to the FPGA device design within the SOPC Builder tool.

Recommendation: Use the default component names (*ComponentClass*_inst) whenever possible so that you do not need to modify the FTXL HAL file, **FtxlHal.c**.

To add FTXL components to the design:

1. Start the Quartus II software.
2. Open your Quartus II project to which you plan to add the FTXL components.
3. Select **Tools → SOPC Builder** to open the Altera SOPC Builder tool.
4. In the Altera SOPC Builder tool, select the **System Contents** tab.
5. Add an Avalon tristate bridge:

a. Expand the **Bridges and Adapters** folder.

b. Expand the **Memory Mapped** folder.

c. Select **Avalon-MM Tristate Bridge**.

d. Click **Add** to open the MegaWizard for the component.

e. In the MegaWizard for the Avalon-MM Tristate Bridge, select **Registered** on the Incoming Signals page. Click **Finish** to add the component to the design.

6. Add the FTXL Parallel I/O Transceiver Interface component:

a. Expand the **FTXL** folder.

b. Select **FTXL Parallel I/O Transceiver Interface**.

c. Click **Add** to open the MegaWizard for the component.

d. In the MegaWizard for the FTXL Parallel I/O Transceiver Interface, there are no parameters to set. Click **Finish** to add the component to the design.

e. Connect **FTXL_PIO_inst.avalon_tristate_slave_0** to **tristate_bridge.tristate_master**.

7. Add the FTXL Service LED component:

a. Expand the **FTXL** folder.

b. Select **FTXL Service LED**.

c. Click **Add** to open the MegaWizard for the component.

d. In the MegaWizard for the FTXL Service LED, there are no parameters to set. Click **Finish** to add the component to the design.

8. Add the FTXL Service Pin component:

a. Expand the **FTXL** folder.

b. Select **FTXL Service Pin**.

c. Click **Add** to open the MegaWizard for the component.

d. In the MegaWizard for the FTXL Service Pin, there are no parameters to set. Click **Finish** to add the component to the design.

e. If necessary, modify the assigned IRQ number for the component; see *Addressing, Size, and IRQ Requirements* on page 49 for recommendations about the IRQ assignments.

9. Add the FTXL Transceiver Interrupt component:

a. Expand the **FTXL** folder.

b. Select **FTXL Transceiver Interrupt**.

c. Click **Add** to open the MegaWizard for the component.

d. In the MegaWizard for the FTXL Transceiver Interrupt, there are no parameters to set. Click **Finish** to add the component to the design.

e. If necessary, modify the assigned IRQ number for the component; see *Addressing, Size, and IRQ Requirements* on page 49 for recommendations about the IRQ assignments.

10. Add the FTXL Transceiver Reset component:
    a. Expand the **FTXL** folder.
    b. Select **FTXL Transceiver Reset**.
    c. Click **Add** to open the MegaWizard for the component.
    d. In the MegaWizard for the FTXL Transceiver Reset, there are no parameters to set. Click **Finish** to add the component to the design.
11. If necessary, set the base addresses of the newly added components. Altera recommends that you let the SOPC Builder tool assign addresses: select **System → Auto-Assign Base Addresses**.
12. Edit the Avalon MM-Tristate Bridge component for the FTXL Parallel I/O Transceiver Interface component to verify that none of the signals are shared:
    a. Right-click the **tristate_bridge** component and select **Edit** to open the MegaWizard for the Avalon MM-Tristate Bridge.
    b. Verify that the bridge is registered on the Incoming Signals page.
    c. Verify that none of the signals for **FTXL_PIO_inst.avalon_tristate_slave_0** are selected on the Shared Signals page.
    d. Click **Finish** to close the MegaWizard for the Avalon MM-Tristate Bridge.

After you add the FTXL components, the design should look similar to **Figure 23**.

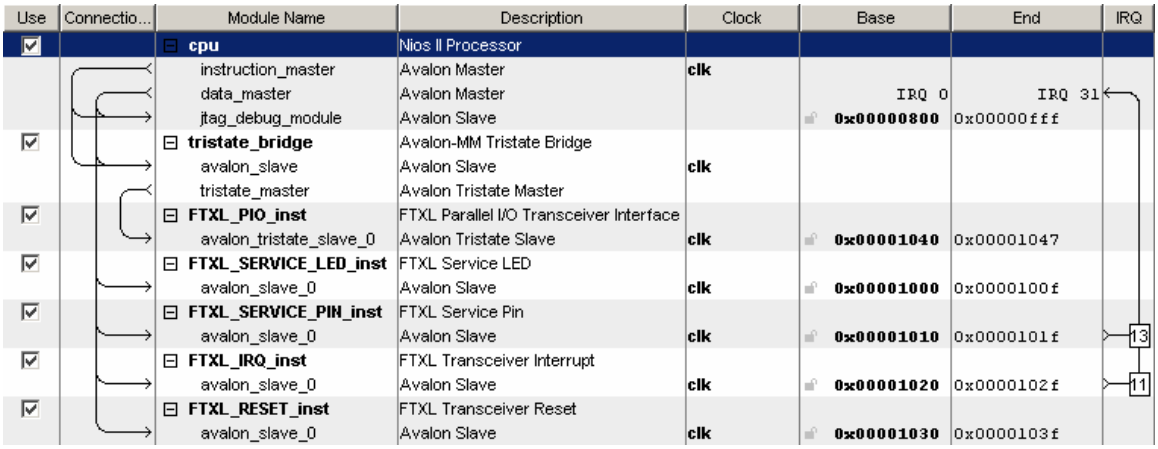


Figure 23. FTXL Components Added to the SOPC Builder Design

When you are satisfied with the SOPC Builder design, you must regenerate it: click **Generate** at the bottom of the SOPC Builder window.

Modifying the Quartus II Design

Before you begin, ensure that the Quartus II global library or project library search path includes the FTXL components; see *Setting Component Search Paths* on page 56.

After you generate the SOPC Builder design, you can update your Quartus II design, including updating the symbol block for the Nios II processor in the block design file (*.bdf) for the project. The updated Nios II block symbol should include the signals shown in **Figure 24** and **Figure 25**.

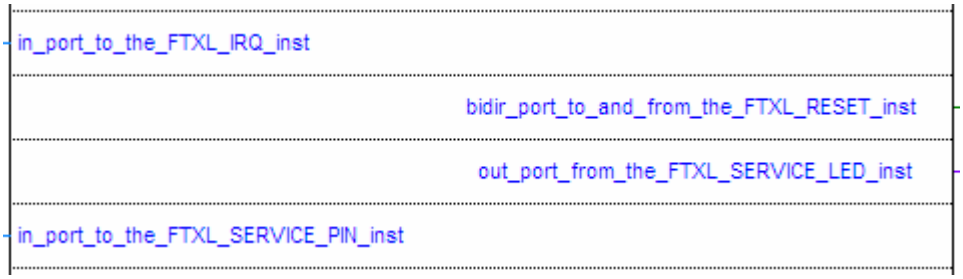


Figure 24. FTXL Signals within the Nios II Processor

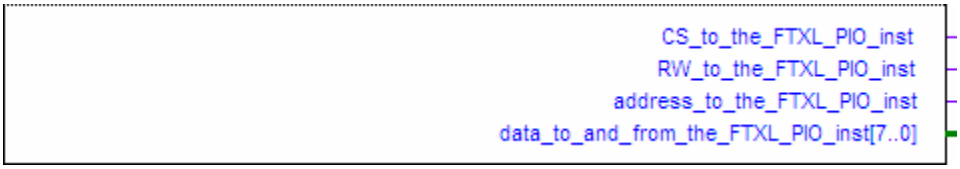


Figure 25. FTXL Parallel I/O Signals within the Nios II Processor

Within the block design file (*.bdf) for the project, you need to add a symbol block for the FTXL Parallel Interface Delay component:

1. Right-click the canvas and select **Insert** → **Symbol** to open the Symbol dialog.
2. In the Symbol dialog, expand the [*Altera*]**\nios2eds\components\ftxl** folder. If this folder does not display, be sure that you have added the FTXL components to the SOPC Builder library path, as described in *Setting Component Search Paths* on page 56.
3. Select **FTXL_PIO_Delay** from the [*Altera*]**\nios2eds\components\ftxl** folder. **Figure 18** on page 42 shows this component's block symbol.
4. Click **OK** to close the Symbol dialog and to add the symbol to the canvas.

After you add the **FTXL_PIO_Delay** symbol to the canvas, you need to connect its signals to the FTXL Parallel I/O signals within the Nios II processor:

- Connect **CS_IN** to **CS_to_the_FTXL_PIO_inst**
- Connect **AO_IN** to **address_to_the_FTXL_PIO_inst**
- Connect **RW_IN** to **RW_to_the_FTXL_PIO_inst**
- Connect **RESET_IN** to **reset_n**

In addition, you need to connect **CLK_IN** to the system clock signal for the Nios II processor (for example, the clock output of a PLL). **Figure 26** on page 61 shows the main connections for the **FTXL_PIO_Delay** component.

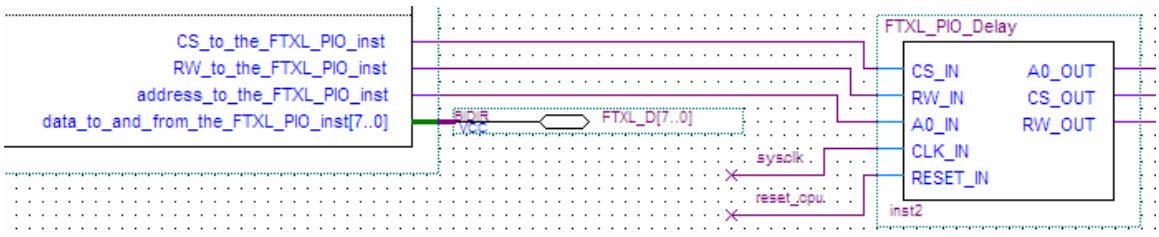


Figure 26. Connections for FTXL_PIO_Delay Component

Finally, you need to add pins to the symbol blocks for the FTXL components:

1. Right-click each component's symbol (for example, the **FTXL_PIO_Delay** symbol and the Nios II processor symbol), and select **Generate Pins for Symbol Ports**.
2. As necessary, rename the pins. The FTXL Hardware Abstraction Layer (HAL) does not use the pin names, but instead uses the signal names, so you can assign any valid names to the pins.
3. Modify pin assignments, as necessary, to match your hardware layout.

When you are satisfied with the design, you must recompile it: select **Processing → Start Compilation** to compile the design. Compilation can take a few minutes.

Building the Application Image

The FTXL Developer's Kit includes the both hardware design for the Nios II processor and the software for the FTXL LonTalk protocol stack, API, and operating system. You must separately build the hardware image and the software image. See the *FTXL User's Guide* for information about working with the FTXL software.

The FTXL Developer's Kit includes a pre-compiled hardware reference design image for the Nios II processor. To use the pre-compiled image, see *Loading the Application Image into the FPGA Device* on page 62.

You might need to rebuild the hardware reference design image, for example, if you want to modify the design, run the Nios II processor on a different device than a Cyclone II FPGA, or if your Altera tools license requires you to rebuild the image.

Before you rebuild the hardware image, ensure that the appropriate hardware components are included in the library search path for the project; see *Setting Component Search Paths* on page 56.

To rebuild the hardware image:

1. Start the Quartus II software.
2. Select **File → Open Project** to display the Open Project window.
3. In the Open Project window, select the Quartus Project File (*.qpf) for the project**,** and click **Open** to add the project file to the Project Navigator.
4. Modify the design as desired.
5. Select **Processing → Start Compilation** (or click the **Start Compilation** button on the toolbar) to compile the project. Compilation can take a few minutes.

6. Load the modified hardware design for the Nios II processor into the FPGA device, as described in *Loading the Application Image into the FPGA Device*.

Loading the Application Image into the FPGA Device

You can choose to load the hardware and software images into the FPGA device's RAM at the same time, or you can choose to load them separately. To load both images at the same time, or to load the images into the FTXL device's flash memory, use the Nios IDE; see the *FTXL User's Guide* for more information.

To load the hardware image for the Nios II processor into the FPGA device:

1. Ensure that the FPGA device is powered on and that the USB-Blaster download cable (or similar programming device) is connected to it.
2. Start the Quartus II software.
3. Select **File** → **Open Project** to display the Open Project window.
4. In the Open Project window, select the Quartus Project File (*.qpf) for the project, and click **Open** to add the project file to the Project Navigator.
5. Select **Tools** → **Programmer** to open the Chain Description File view for the project.
6. Ensure that the USB-Blaster download cable (or similar programming device) is defined in the Chain Description File for the project.

   If you have not defined the USB-Blaster download cable in the Chain Description File for the project, click **Hardware Setup**. See *Using a Device Programmer for the FPGA Device* on page 55 for more information about setting up the USB-Blaster download cable.
7. Verify that the SRAM Object File (*.sof) for the project is already listed in the Chain Description File view for the project.

   If the SRAM Object File (*.sof) for the project is not listed in the Chain Description File view for the project, click **Add File** to open the Select Programming File dialog. From the Select Programming File dialog, select the SRAM Object File (*.sof) for the project, and click **Open** to add the file to the Chain Description File view for the project.
8. Verify that the **Program/Configure** checkbox is selected for the hardware design file.
9. Click **Start** to load the hardware design into the FPGA device.
10. After the Quartus II software has finished loading the hardware image into the FPGA device, perform a reset of the FPGA device.
11. Select **File** → **Exit** if you want to close the Quartus II software window.

A

Using the Bring-Up Application to Verify FTXL Hardware Design

This chapter describes how to use the Bring-Up application that is included with the FTXL Developer's Kit to test and verify a new or modified FTXL hardware design. The Bring-Up application tests the communications interface between the FTXL 3190 Free Topology Smart Transceiver and the Nios II host processor.

Overview

This appendix describes the Bring-Up application that is included with the FTXL Developer's Kit. This application implements a series of tests that exercise the hardware interface between the Nios II processor, the FTXL 3190 Free Topology Smart Transceiver, the FTXL service pin, and the FTXL service LED. You should run these tests to verify a new or modified FTXL device's hardware design.

The Bring-Up application requires a Nios II hardware design that contains the FTXL components and a STDOUT display device. In addition, the device must have a JTAG or similar interface between the device and the PC that will run the Nios IDE. The Bring-Up application can run with or without the Micrium µC/OS-II operating system.

Because the Bring-Up application is based on the FPGA hardware reference design used by the example applications that are included with the FTXL Developer's Kit, you might have to modify the **FtxlHal.c** file if your hardware design differs from the reference design (for example, if your hardware design uses different signal names or different logic).

Recommendation: While running the tests, connect a digital analyzer to the FTXL Transceiver communications pins, the service LED pin, and the service pin so that you can externally verify the signals on the pins. The test descriptions in *Running the Tests* on page 70 include logic analyzer traces that were generated using the TechTools DigiView™ Logic Analyzer.

Application Framework

The Bring-Up application consists of one C source file (**FtxlBringupApp.c**) that includes the series of tests. This file, in turn, includes the **FtxlHal.h** file and uses functions from the **FtxlHal.c** file. All of these files are included in the Bring-Up application project folder.

Interrupt Functions from the FTXL HAL

The FTXL interface defines interrupts from two sources: the FTXL Transceiver and the service pin.

The **FtxlBringupApp.c** file defines the following callback handler functions to process interrupts:

- **LonDriverTransceiverIrq()**

  Called to process an interrupt from the FTXL Transceiver.

- **LonIsrServicePin()**

  Called to process an interrupt from the service pin.

The **FtxlHal.c** file defines the following functions to handle interrupts:

- **LonRegisterIsr()**

  Initializes the interrupt system, including registering the interrupt handlers described in *Application Framework* on page 64.

- **LonEnableInterrupt()**

  Used to enable either or both of the interrupts.

- **LonDisableInterrupt()**

  Used to disable either or both of the interrupts.

FTXL Transceiver Interface

The interface between the Nios II processor and the FTXL Smart Transceiver includes the following signals:

- A bi-direction reset signal
- A set of status signals that are used to determine when the FTXL Transceiver is ready to accept data or has data to be read
- A bi-directional 8-bit data register
- An interrupt

These signals cannot be tested in isolation. For example, you cannot test the data register without first verifying that the FTXL Transceiver has been properly reset and that the status register is properly reporting its state. And you cannot completely test the status register without transferring some data. *Running the Tests* on page 70 describes the series of tests, to be performed in order, that fully test communications between the Nios II processor and the FTXL Transceiver.

The following sections describe the interface. See Chapter 3, *FTXL Transceiver Hardware Interface*, on page 19, for additional information about the FTXL transceiver interface.

Reset Signal

The interface includes a bi-direction reset signal. For most of the time, this signal is configured as an input. The input includes an edge-capture register, which the Nios II processor can read to determine whether the FTXL Transceiver has been reset. The FTXL function **LonReadTransceiverReset()** reads the reset capture register.

To reset the FTXL Transceiver, the host processor configures this signal as an output, asserts the signal, then reconfigures the signal as an input to deassert the signal. The following FTXL HAL functions configure the reset signal: **LonAssertTransceiverReset()** and **LonDeassertTransceiverReset()**. In addition, the **LonDeassertTransceiverReset()** function also clears the reset capture register.

The **LonReadTransceiverReset()** function reads the reset capture register, which is set by the FTXL Transceiver reset pin. The **LonAssertTransceiverReset()** and

LonDeassertTransceiverReset() functions write to the FTXL Transceiver reset pin.

Status Signals

The interface includes a set of status signals that are used to determine when the FTXL Transceiver is ready to accept data or has data to be read. The status signal can be read using the FTXL HAL function **LonTransceiverIsBusy()**.

When the **LonTransceiverIsBusy()** function reads the status register, the FTXL_PIO component affect FTXL Transceiver pins as listed in **Table 19**.

Table 19. Status Signals

Signal Name	FTXL Transceiver Pin	Action
CS~	31	Asserted (low)
R/W~	30	Deasserted (high)
A0	27	Asserted (high)
D0	4	Read as status (low = busy)

Data Register

The interface includes a bi-directional 8-bit data register. This register is used either to:

- Write data from the Nios II processor to the FTXL Transceiver using the FTXL HAL function **LonWriteTransceiverDataRegister()**
- Read data from the FTXL Transceiver using the FTXL HAL function **LonReadTransceiverDataRegister()**

The register must be read or written only when **LonTransceiverIsBusy()** returns **FALSE**.

When **LonWriteTransceiverDataRegister()** writes to the data register, the FTXL_PIO component affects FTXL Transceiver pins as listed in **Table 20**.

Table 20. Writing to the Data Register

Signal Name	FTXL Transceiver Pin	Action
CS~	31	Asserted (low)
R/W~	30	Asserted (low)
A0	27	Deasserted (low)
D0	4	Asserted or deasserted as data values
D1	3	

Signal Name	FTXL Transceiver Pin	Action
D2	2	
D3	43	
D4	42	
D5	36	
D6	35	
D7	32	

When **LonReadTransceiverDataRegister()** reads the data register, the FTXL_PIO component affects FTXL Transceiver pins as listed in **Table 21**.

Table 21. Reading from the Data Register

Signal Name	FTXL Transceiver Pin	Action
CS~	31	Asserted (low)
R/W~	30	Deasserted (high)
A0	27	Deasserted (low)
D0	4	Read as data values
D1	3	
D2	2	
D3	43	
D4	42	
D5	36	
D6	35	
D7	32	

As described in *Token Passing and Handshaking* on page 23, communications between the Nios II processor and the FTXL Transceiver use a token passing protocol. Initially after a reset, the Nios II processor has the token, and can perform either of the following actions:

- Write a data packet consisting of a non-zero length, one or more data bytes, and a "null" token byte (value of 0x0)
- Pass the token by writing two consecutive 0x0 data bytes

To write each byte, the application must first wait until **LonTransceiverIsBusy()** returns **FALSE**, and then call **LonWriteTransceiverDataRegister().** After the write is complete, the FTXL Transceiver has the token, and either sends a data packet or passes the token back. The application must read the data packet or token by reading first the length byte (which indicates the number of data bytes) and then reading all data bytes. If the length byte is 0x0, there is no data. To read each byte, the application must first wait until **LonTransceiverIsBusy()** returns **FALSE**, and then call **LonReadTransceiverDataRegister().**

The FTXL Transceiver enters a "fast I/O" mode during certain portions of a data transfer. During these times, the host processor must read or write the next byte within a certain amount of time, or the FTXL Transceiver assumes that it has lost synchronization with the host and resets. The timeout period is a function of the FTXL Transceiver clock speed. **Table 22** lists the timeout values. Note that the application must respond within the minimum time shown in the table to ensure that the FTXL Transceiver does not timeout.

Table 22. Fast-I/O Mode Timeout Values

Transceiver Clock Rate	Timeout
5 MHz	1.68 to 3.36 seconds
10 MHz	0.84 to 1.68 seconds
20 MHz	0.42 to 0.84 seconds
40 MHz	0.21 to 0.42 seconds

The FTXL Transceiver enters fast-I/O mode under the following conditions:

- After the application reads the first byte of a packet (the length byte), the FTXL Transceiver enters fast-I/O mode until all bytes in the packet have been read.
- After writing a zero length byte (to pass the NULL token), the FTXL Transceiver enters fast-I/O mode until the token byte has been written.
- After writing the first **data** byte of a packet, the FTXL Transceiver enters fast-I/O mode until all subsequent data bytes in the packet and the token byte have been written. Note that the FTXL Transceiver does not go into fast-I/O mode upon receiving the length byte – only after the first data byte (second byte of the frame) has been written.

Interrupt

The interface includes an interrupt signal. The FTXL Transceiver interrupts the Nios II processor under the following conditions:

- When the FTXL Transceiver has data to be sent to the host, whether it has the token or not.
- If the FTXL Transceiver has the token, and is ready to pass the NULL token.
- After the FTXL Transceiver has received a non-zero length byte from the host.

The application uses the functions described in *Interrupt Functions from the FTXL HAL* on page 64 to handle interrupts.

Working with the Nios IDE for the Bring-Up Application

Although the Bring-Up application is designed to test the hardware design, it is a software application, and thus uses the Nios II IDE. To set up the Nios II IDE for the Bring-Up application, perform the following general steps:

1. Recommended: Create a new workspace for each example application project.
2. Create a new application project based on the **FTXL Bring-Up Application** project template.
3. Build the project.

The following sections describe these steps. After you build the project, you can run it. There should be no need to load the application into the device's flash memory; instead, you can run the application from the Nios IDE.

Creating a New Application Project

Recommendation: Create the Bring-Up application project in a new workspace. To work in a new workspace, select **File → Switch Workplace** to open the Workspace Launcher window, from which you can specify a new workspace.

To create a new application project for the FTXL Bring-Up application:

1. Select **File → New → Nios II C/C++ Application** to open the New Project window.
2. From the New Project window's **Select Project Template** selection box, select the **FTXL Bring-Up Application** project.
3. Optional: Enter a project name in the **Name** field. The default name is **FTXL_BringupApp_0**.
4. Specify a location for this project (such as **C:\MyFtxl**) by selecting the **Specify Location** checkbox and specifying the location in the **Location** field. The directory name must not contain spaces. If you use the default location, your source files will be placed in the project workspace directory.
5. Specify the target hardware. Click **Browse** in the Select Target Hardware area to open the Select Target Hardware dialog.
    a. In the Select Target Hardware dialog, browse to your device's directory and select the SOPC Builder system file for the project.
    b. Click **Open** to select the file and close the Select Target Hardware dialog.
6. Click **Finish** to create the project and generate the project's system library.

Building the Application Image

To build the software image for Bring-Up application:

1. Start the Nios II EDS IDE.
2. Ensure that the workspace includes the Bring-Up application project.
3. Select **Project → Build Project** or **Project → Build All.** You can also right-click the project folder from the Nios II C/C++ Projects pane and select **Build Project**.

The first build for a new project can take a few minutes.

After you build the project, you can run it, as described in *Running the Application from the Nios IDE*.

Running the Application from the Nios IDE

To run the Bring-Up application from the Nios II EDS IDE:

1. Ensure that your device's board is powered on and that it is connected to the PC through a JTAG-type connection, such as the USB-Blaster download cable.
2. Start the Nios II EDS IDE.
3. Ensure that the workspace includes the Bring-Up application project.
4. Right-click the project from the Nios II C/C++ Projects pane and select **Run As → Nios II Hardware** or **Debug As → Nios II Hardware**. The Nios II EDS IDE recompiles the project.

The tests should start running, as described in *Running the Tests*.

Running the Tests

The Bring-Up application includes the following tests:

- Reset Test
- Token Passing Test
- Data Passing Test
- Interrupt Test
- Service-Pin and LED Test

You must run these tests in order, although after the Reset Test completes, you can press the device's service pin to run the Service-Pin and LED Test at any time.

Each of these tests leaves the FTXL Transceiver running, with the Nios II processor in possession of the token. All tests, except the Reset Test, expect the FTXL transceiver to be in this state upon entry as well. If the last test performed succeeds, you can call any of the other tests. Otherwise, you can restore the state of the FTXL Transceiver by re-running the Reset Test.

The output of each test lists the test name together with an indication of whether it passed or failed. For some of the tests, additional output describes specific error conditions or wait states.

Reset Test

The Reset Test exercises the FTXL Transceiver's reset line by asserting and deasserting it, and then verifying that both the status register and reset capture register behave as expected. All of the other tests in the suite require that the reset and status register behave properly.

Figure 27 provides an overview of the signals during the entire test. The CS~, R/W~, and A0 signals are in flux while the host polls the handshake register waiting for the FTXL Transceiver to complete its reset.

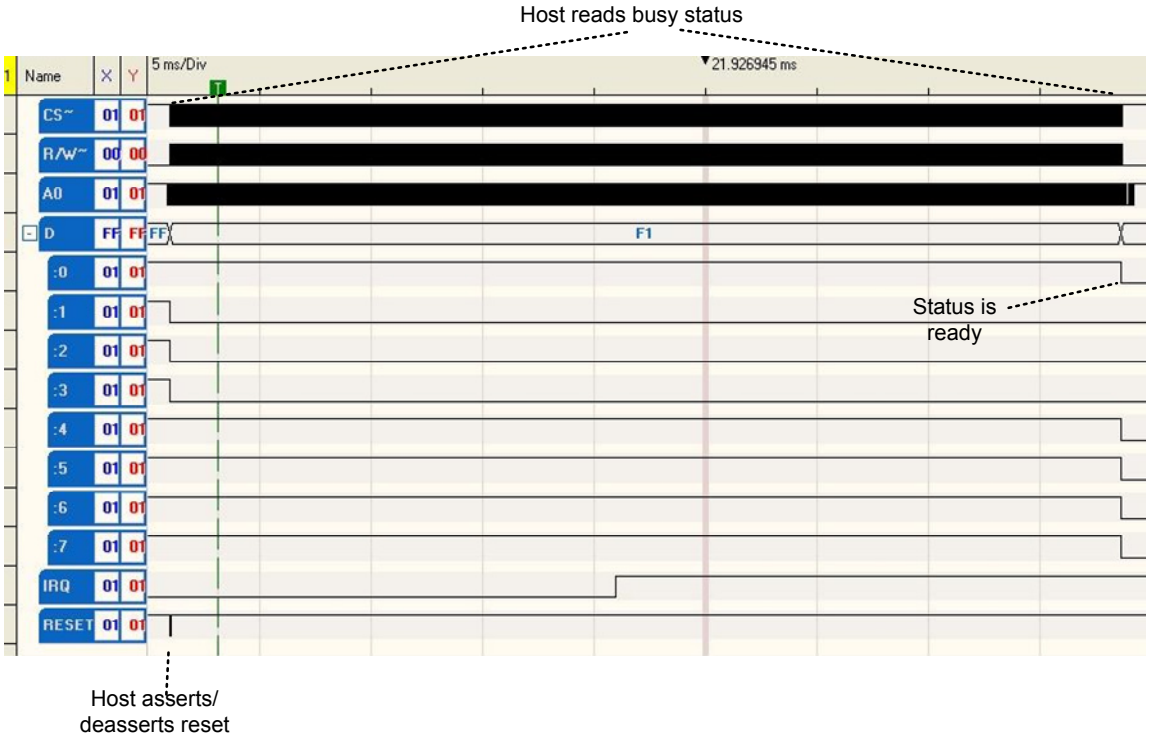


Figure 27. Reset Test Overview

Figure 28 on page 72 shows the detail of the far left-hand side of **Figure 27**, and shows the signals when the host asserts and deasserts the reset line. In this diagram, each time that the CS~ line goes low, the host reads the handshake signal.

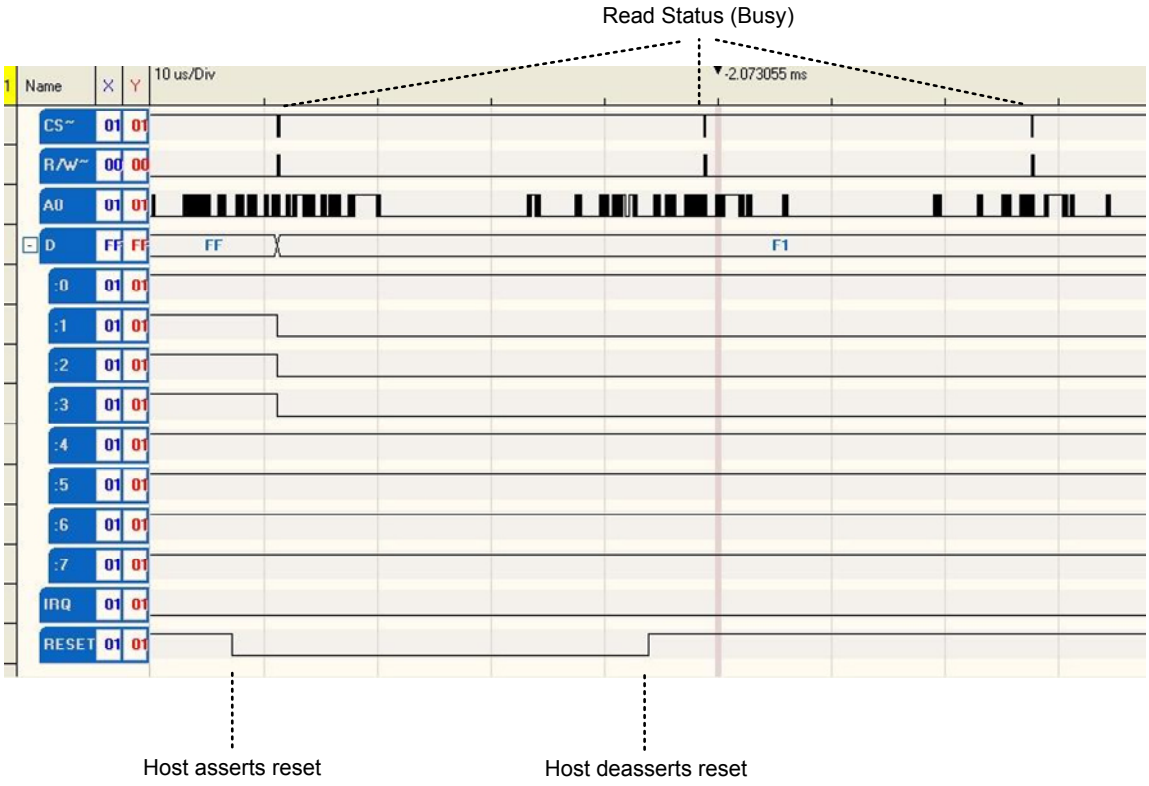


Figure 28. Host Asserts and Deasserts Reset and Checks Status

Figure 29 on page 73 shows the detail of the first read status from **Figure 28**. The figure shows the CS~, R/W~, A0, and HS signals just after the reset line has been asserted, and the host reads the status register. When CS~ is asserted (low), a byte transfer is in progress. R/W~ controls whether the transfer is a read or a write. In this case, R/W~ is deasserted (high), which indicates that this transfer is a read. The A0 signal controls whether the D0 signal should report the handshake (status) or the least significant bit of the data. In this case A0 is high, which indicates that D0 represents the handshake signal.

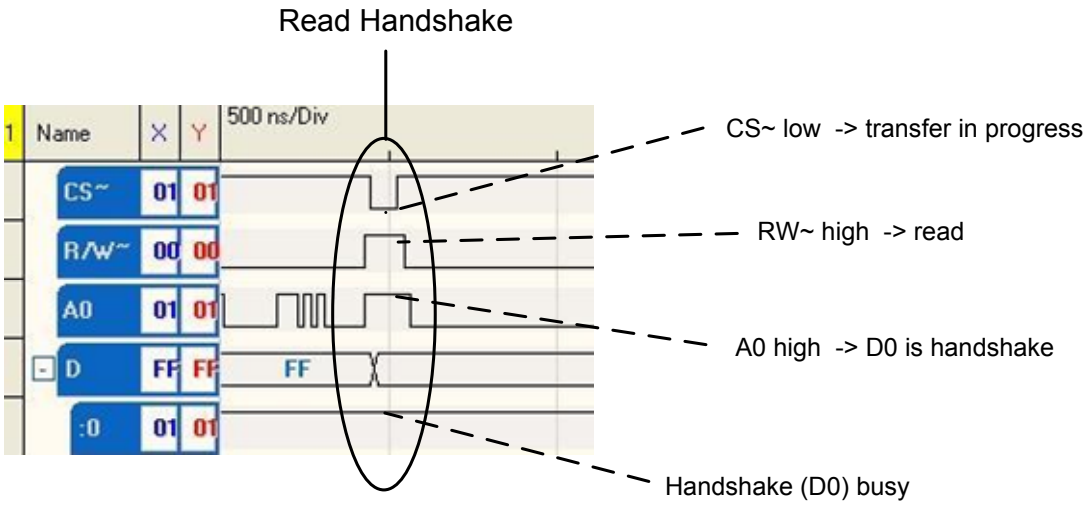


Figure 29. Reading Status Register during Reset

Token Passing Test

After the reset test runs successfully, the Token Passing Test repeatedly passes the null token between the Nios II processor and the FTXL Transceiver under program control (that is, without interrupts). This test verifies the status register to ensure that it reports busy immediately after write operations, and verifies that the reset capture register does not report reset during normal operation. Reading and writing to the data register is partially verified. However, in this test, both the application and the FTXL Transceiver write zero bytes to the data register.

This test and the remaining tests use a utility function (**waitUntilReady()**) to wait for the FTXL Transceiver to become ready. This function checks the reset capture register to detect resets, and includes a basic timeout mechanism.

Figure 30 on page 74 shows the state of the signals while writing the length byte.

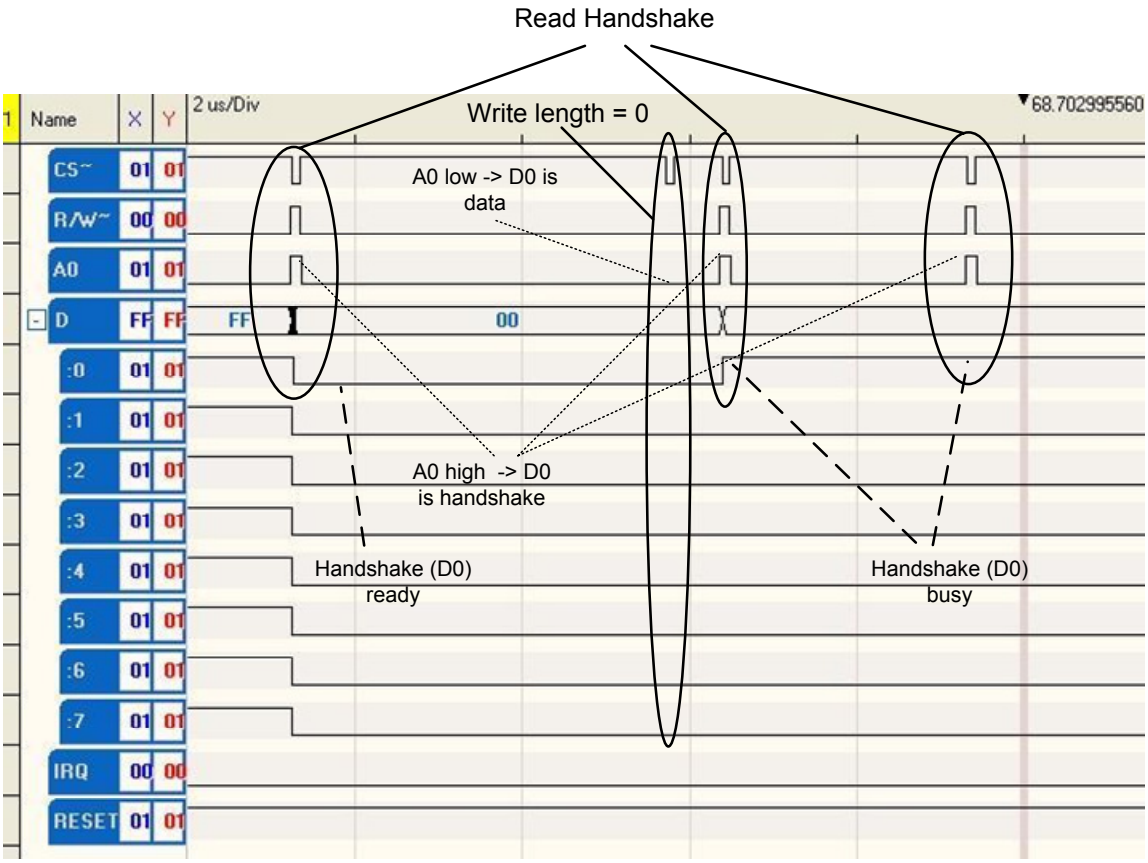


Figure 30. Writing the Length Byte (0x00)

The CS~ signal is asserted (low) every time a byte operation is in progress. The R/W~ signal is asserted (low) on a write, and deasserted (high) on a read. To read the handshake (or status) register, the A0 signal is deasserted (high), in which case D0 is the handshake signal. If A0 is asserted (low), the D0 signal is the least significant data bit. The diagram shows the following four operations:

- Read status (CS~ low, R/W~ high, A0 high). The status register (D0) is ready.
- Write data byte with value 0x0 (CS~ low, R/W~ low, A0 low, D0-D7 = 0x00).
- Read status (CS~ low, R/W~ high, A0 high). The status register (D0) is busy.
- Read status (CS~ low, R/W~ high, A0 high). The status register (D0) is busy.

Figure 31 on page 75 shows the state of the signals when the host passes the token to the FTXL Transceiver. This process consists of the following steps:

1. Read the status register and determine that the FTXL Transceiver is ready.
2. Write the length byte (0x00).
3. Repeatedly read the status register until the FTXL Transceiver is ready.
4. Write the token byte (0x00).

The figure also shows the status being read after the token is passed.

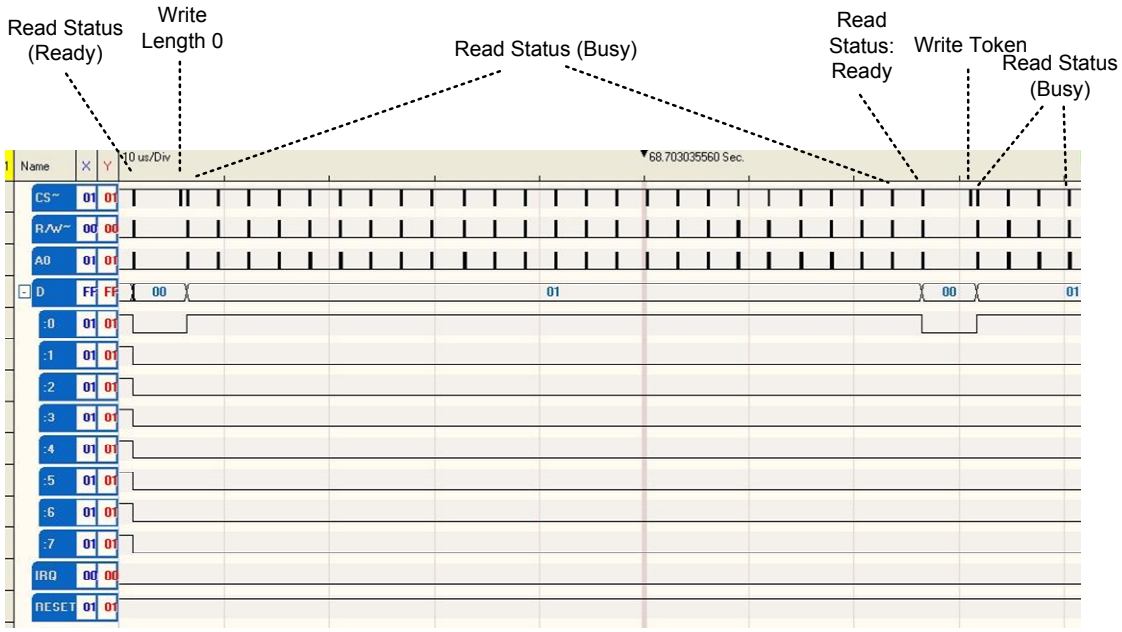


Figure 31. Passing the Token from the Host to the FTXL Transceiver

Figure 32 on page 76 shows the state of the signals while reading a null token from the FTXL Transceiver. In this figure, the host reads the status three times. The first two times the status is busy, which indicates that the transceiver has no data to be read. On the third attempt, the status is ready (D0 is low). Then, the host reads the data by deasserting A0 while asserting CS~ and R/W~. The data to be read is in D0-D7, and represents the length of the packet. In this case, the length is 0x00, and the host now has the token.

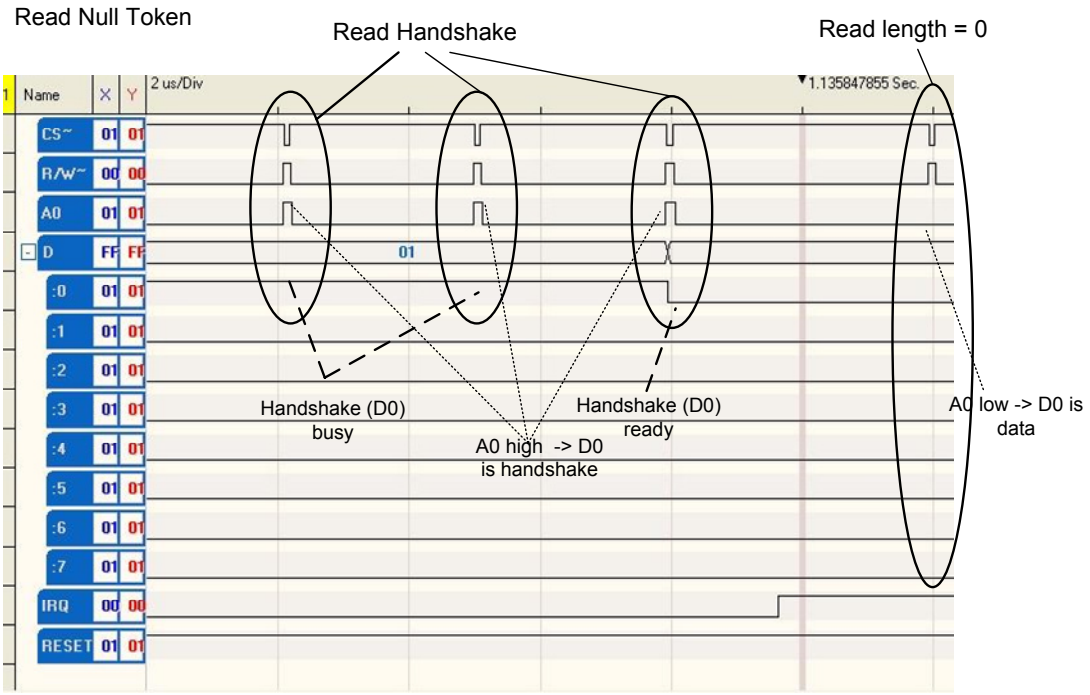


Figure 32. Reading Null Token from the Transceiver

Data Passing Test

The Data Passing Test exercises full two-way communication between the Nios II processor and the FTXL Transceiver. As with the previous test, this test is dependent on the proper function of the reset and status registers. However, in this test, both the Nios II processor and the FTXL Transceiver pass non-zero data patterns that are sufficient to test each bit of the data register.

At the end of the test, the application purposely causes a transmission timeout, which causes the FTXL Transceiver to reset, and verifies that the reset capture register detects this condition. This test, like the others, restores the state of the FTXL Transceiver so that it is ready for subsequent tests.

This test and the Interrupt Test use the following data declarations for the data to be sent and received:

```
/* The host sends downlinkMessage to the         */
/* FTXL Transceiver                              */
const byte downlinkMessage[] = {0x09, 0x52, 0x01, 0x02,
0x04, 0x08, 0x10, 0x20, 0x40, 0x80};

/* The FTXL Transceiver should respond by sending */
/* expectedReply                                  */
const byte expectedReply[] =   {0x09, 0x53, 0x69, 0xd8,
0xd7, 0x77, 0x14, 0xd7, 0x3d, 0xf6};
```

These messages are in the form of a message frame, starting with the length byte and followed by the data bytes. The null terminator is not included.

These two tests also use a utility function (**readUplinkMessage()**) to read a message from the FTXL Transceiver and verify its contents.

Figure 33 shows the signals while writing the downlink data.

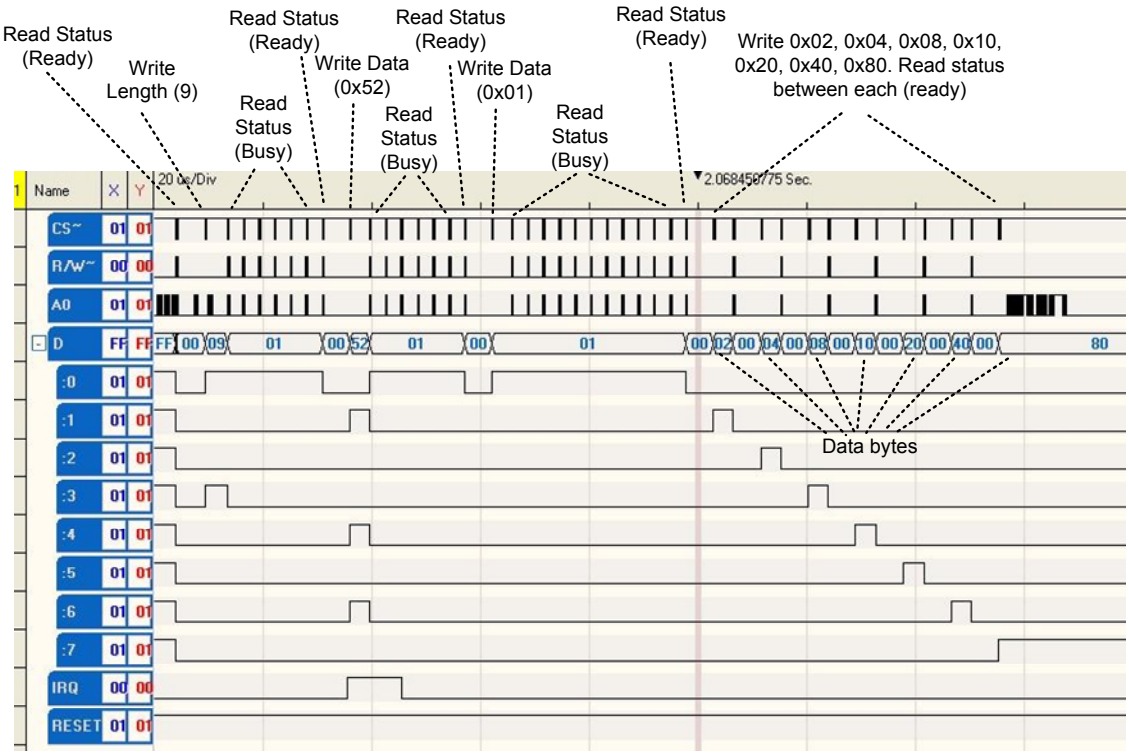


Figure 33. Writing the Downlink Data

Note: You can read the data that is being written by looking at the line labeled **D**, but because the D0 line switches it function between being a data line and being the handshake line, be sure to interpret the **D** line as data only when A0 is low.

Figure 34 on page 78 shows the signals while reading the uplink data.

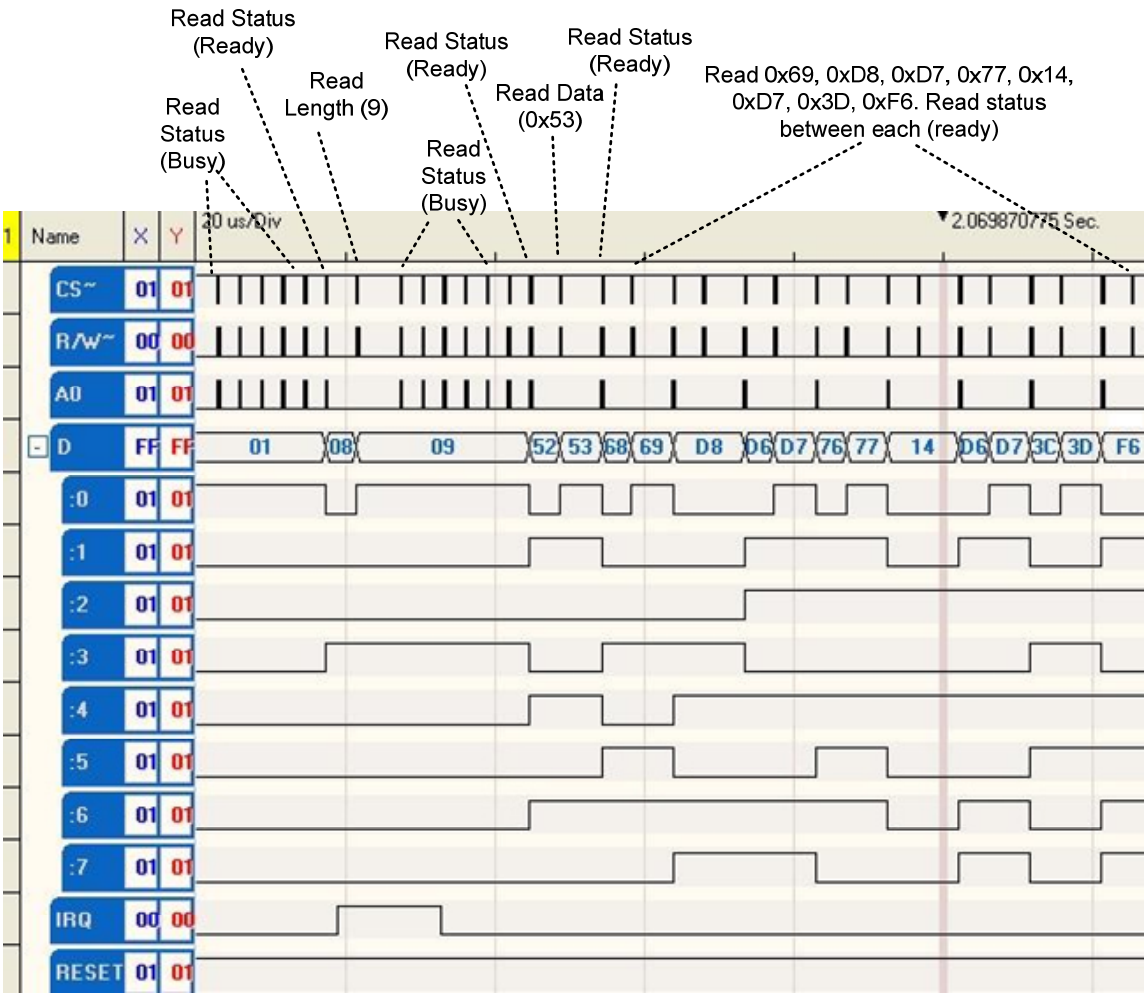


Figure 34. Reading the Uplink Data

Interrupt Test

The Interrupt Test performs the same two-way communication as the Data Passing Test, but uses an interrupt service routine rather than managing the communications under program control. The test application initiates the downlink transfer by writing the length of the downlink buffer, and enables the FTXL Transceiver interrupt. The test then waits for the response to be completed. The downlink transfer is completed by the interrupt service routine, and when the FTXL Transceiver initiates an uplink transfer, the interrupt service routine reads the message and validates the data contained in it.

In order to run this test, the application first calls the **LonRegisterIsr()** function to initialize the interrupt system.

The test function and the interrupt service routine communicate with each other using the following global variables:

- **hostHasToken** – **TRUE** if the host has the token, **FALSE** if the FTXL Transceiver has the token.

- **outputDataStream** – To write a frame to the host, the frame is first copied to this buffer (including the length byte), and then the length is written to the host. The interrupt service routine writes the rest of the data.
- **uplinkReceived** – **TRUE** when an uplink message has been received.
- **unnexpectedUplink** – **TRUE** if the uplink message does not match the expected uplink message.
- **pExpectedUplinkMsg** – Pointer to the expected contents of the next uplink message.

Figure 35 shows the signals during the Interrupt Test. On the left-hand side, you can see the downlink transfer; on the right-hand side, you can see the uplink transfer. At the bottom, you can see the interrupt line.

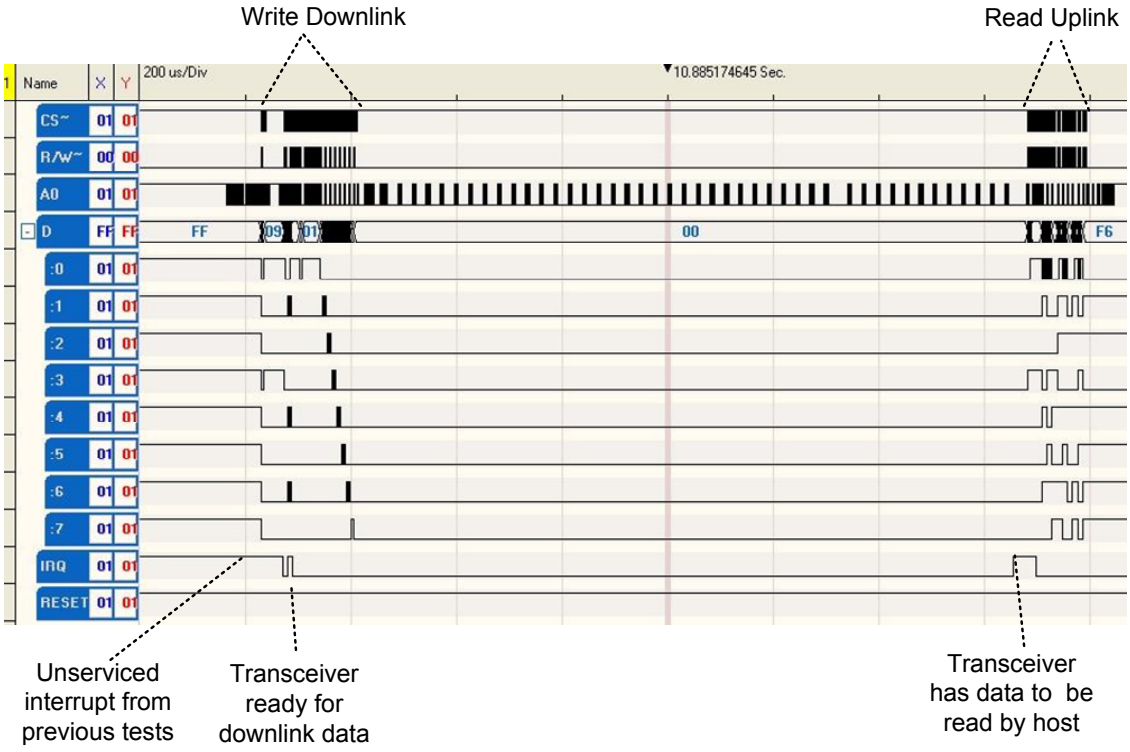


Figure 35. Signals during the Interrupt Test

Service Pin and LED Test

The service pin, service interrupt, and service LED are all tested in a single simple test. This test can be performed independently or in conjunction with the other FTXL Transceiver tests.

This test is performed entirely in the service pin interrupt routine, which illuminates the service LED each time the service pin is pressed, and clears it each time the service pin is released.

Designing Additional Tests

The tests described in *Running the Tests* on page 70 verify the FTXL hardware design, focusing on communications between the Nios II processor and the FTXL Transceiver. Other kinds of test that you should consider running include:

- One or more tests for managing the non-volatile data, including reading and writing to flash
- One or more tests for verifying communications with the LONWORKS network

These tests require a more complete (but still fairly simple) FTXL application, including the operating system.

If you use either of the standard non-volatile drivers, you can enable tracing by setting the global variable **nvdTraceEnabled** to a non-zero value. If create your own custom non-volatile data driver, be sure to add some tracing capability to it for use in the bring-up phase.

After you verify basic communication between the Nios II processor and the FTXL Transceiver, you should be ready to verify that your FTXL device can communicate on the LONWORKS network. The simplest approach is to create a new application project in the Nios IDE using the FTXL Simple template, and select your hardware design (SOPC Builder system PTF file) rather than the reference design. If your FTXL Transceiver runs at a clock rate other than 20 MHz, you must run the LonTalk Interface Developer utility and specify the correct clock rate for the device.

Within the Nios IDE:

1. Load the application into your hardware under debugger control.
2. Verify that your application is running and calls the **LonEventPump()** function properly. For example, set a breakpoint on the call to **LonEventPump()** and generate an event by pressing the device's service pin button.
3. Remove the breakpoint on the call to **LonEventPump()** and verify that your FTXL device can send a service pin message. To see the service pin message, you can use the Echelon LonScanner Protocol Analyzer or a network management tool such as the LonMaker Integration tool. Press and release the device's service pin button to send the service pin message.
4. If no service pin message is sent, verify that the clock rate used by the FTXL Transceiver matches that defined in the LonTalk Interface Developer utility. You should also verify that your channel is properly wired, and that other devices can communicate on the channel.
5. After you have successfully sent a service pin message, commission the device using a network manager such as LonMaker. Browse the device's network variables and observe that they work properly.

Index

A

A0 pin, 22
addressing requirements, 49
Altera Complete Design Suite, 54
application image
  building, 61
  loading, 62

B

bring-up application
  additional tests, 80
  building, 70
  data register, 66
  framework, 64
  interface, 65
  interrupt, 68
  interrupt functions, 64
  new project, 69
  Nios IDE, 69
  overview, 64
  reset signal, 65
  running, 70
  status signals, 66
  tests, 70
building, application image, 61
buttons, DBC2C20 development board, 9

C

clock pins, 27
communications lines, pull-ups, 21
components
  adding, 57
  DBC2C20, 48
  FTXL, 41
  requirements, 49
  search paths, 56
configuration device, FPGA, 40
connectors
  DBC2C20 development board, 11
  FTXL Adapter Board, 13
  FTXL Transceiver Board, 16
control flow
  host from FTXL Transceiver, 32
  host to FTXL Transceiver, 29
control signal buffer, 20
CS~ pin, 22

D

D0-D7 pins, 22
data bus isolation, 20
data register, bring-up application, 66
data transfer, 23
data-passing test, bring-up application, 76
DBC2C20 development board
  buttons, 9
  components, 48
  connectors, 11
  headers, 11
  jumpers, 11
  LEDs, 9
  overview, 8
DC-DC converter, 20
design, modifying, 57, 60
devboards.de GmbH, 3
developer's kit. *See* FTXL Developer's Kit
development process
  FPGA design, 5
  hardware design, 4
  overview, 3
  software design, 5
development tools, 54
device programmer, 55
documentation
  Altera, iv
  devboards, v
  Echelon, iii
downlink control flow, 32

E

EBV Elektronik GmbH, 3
external memory, 48

F

FPGA
  configuration device, 40
  design, 38
  device requirements, 39
FTXL Adapter Board
  connectors, 13
  headers, 13
  jumpers, 13
  overview, 13
FTXL Developer's Kit
  components, 41
  DBC2C20 development board, 8
  hardware, 8
  overview, 3
  reference design, 38
FTXL Transceiver Board
  connectors, 16

headers, 16
jumpers, 16
LEDs, 16
overview, 15

H

HAL, 50
handshaking, 23
hardware abstraction layer, 50
hardware interface
control signal buffer, 20
data bus isolation, 20
DC-DC converter, 20
overview, 20
pull-up resistors, 21
hardware, overview, 2
headers
DBC2C20 development board, 11
FTXL Adapter Board, 13
FTXL Transceiver Board, 16
host receive, control flow, 29
host send, control flow, 32
HS pin, 22

I

I/O pins, 24
interface. *See* parallel communications interface, *See* hardware interface
interrupt test, bring-up application, 78
interrupt, bring-up application, 68
IO0-IO10 pins, 24
IRQ pin, 24
IRQ requirements, 49

J

jumpers
DBC2C20 development board, 11
FTXL Adapter Board, 13
FTXL Transceiver Board, 16

L

LEDs
DBC2C20 development board, 9
FTXL Transceiver Board, 16
loading, application image, 62
LVI, reset, 27

M

MegaCore IP library, 54
memory, external, 48
ModelSim simulation tool, 54
modifying the design
Quartus II, 60
SOPC Builder, 57

N

Nios II Embedded Design Suite, 54
Nios II processor, 40

P

parallel communications interface
handshake, 23
overview, 21
pin assignments, 21
token passing, 23
transferring data, 23
parallel I/O transceiver interface, 43
parallel interface delay, 42
phase-locked loop, 48
pins
A0, 22
assignments, 21, 28
characteristics, 24
clock, 27
CS~, 22
D0-D7, 22
FPGA, 28
HS, 22
I/O, 24
IRQ, 24
R/W~, 22
RESET~, 25
service, 27
PLL, 48
power-up sequence, 26
pull-up resistors, 21

Q

Quartus II software, 54

R

R/W~ pin, 22
reference design, 38
reset
bring-up application, 65
LVI, 27
overview, 25
power-up sequence, 26
software controlled, 27
timing, 27
watchdog timer, 27
reset test, bring-up application, 71
RESET~ pin, 25

S

search paths, 56
service LED, 46
service pin, 27, 47
service pin test, bring-up application, 79

signals. *See* pins
software-controlled reset, 27
SOPC Builder, 54
status signals, bring-up application, 66

T

test
  data passing, 76
  interrupt, 78
  network communications, 80
  reset, 71
  service pin, 79
  token passing, 73
timers, 48
timing, reset, 27
token passing, 23
token-passing test, bring-up application, 73
transceiver interrupt, 47
transceiver reset, 47
transferring data, 23

U

uplink control flow, 29
USB-Blaster download cable, 55

W

watchdog timer, 27

www.echelon.com