Channel Adapter
and
Channel Adapter Plus
FPGA Core Reference Manual

797 North Grove Rd, Suite 101
Richardson, TX 75081
Phone: (972) 671-9570
www.redrapids.com

Table of Contents

1.0 Introduction 1
1.1 Contents and Structure 1
1.2 Related Publications 1
1.3 Conventions 1
1.4 Manual Compatibility 2
1.5 Revision History 2
2.0 FPGA Core Library 3
3.0 DCM Core (adc_dcm_mXXX) 5
4.0 ADC Capture FPGA Core (adc_capture_mXXX) 6
4.1 ADC Capture FPGA Core - User Signals 6
4.1.1 Resets and Clocks 6
4.1.2 Control Interface 6
4.1.3 Data and Status 7
4.2 ADC Capture FPGA Core - Hardware Signals 7
5.0 QDR SRAM Interface FPGA Core (qdrsram_iface) 8
5.1 SRAM Interface FPGA Core - User Signals 8
5.1.1 Resets and Clocks 8
5.1.2 Auto Calibration Port 9
5.1.3 Write Port 9
5.1.4 Read Port 9
5.2 SRAM Interface FPGA Core - Hardware Signals 11
6.0 IDELAYCTRL FPGA Core 12
7.0 Local Bus Interface FPGA Core (Local_Bus_Interface_Wrapper) 13
7.1 Local Bus Interface FPGA Core - User Signals 13
7.1.1 Resets and Clocks 13
7.1.2 Local Bus Interface 13
7.1.3 Interrupts 14
7.1.4 DMA Control and Status 14
7.2 Local Bus Interface FPGA Core - Hardware Signals 16
7.3 Local Bus Interface Simulation Model 16
7.3.1 LBI_access.vhd 16
7.3.2 Local_Bus_Interface.vhd 17
7.3.3 Local Bus Interface Simulation Model Operation 18
8.0 FPGA and Configuration PROM Programming 21
8.1 Setting up a Xilinx ISE Project for the Local Bus Example 21
8.2 Creating XSVF Programming Files 22
8.2.1 Creating PROM File for M314 & M320 22
8.2.2 Creating XSVF File for M314 & M320 23
8.2.3 Creating PROM File for M316 23
8.2.4 Creating XSVF File for M316 24
8.3 Loading XSVF Files 25
8.4 Programming via the JTAG connector 25
8.5 Custom FPGA Development Projects 25

List of Figures

Figure 1-1 Placeholder 2
Figure 2-1 Core Interconnect 3
Figure 2-2 Signal Acquisition Function 4
Figure 7-1 JTAG Chain 21

List of Tables

Table 4-1 ADC User Interface Reset and Clock Signals 6
Table 4-2 ADC User Interface Control Register Interface Signals 6
Table 4-3 ADC User Data and Status Signals 7
Table 4-4 Model 314 Receiver Interface Signals 7
Table 4-5 Model 320 Receiver Interface Signals 7
Table 5-1 SRAM User Interface Reset and Clock Signals 8
Table 5-2 SRAM User Interface Control Register Interface Signals 9
Table 5-3 SRAM User Data and Status Signals 9
Table 5-4 SRAM Interface Signals 11
Table 7-1 Local Bus User Interface Reset and Clock Signals 13
Table 7-2 DMA Engine Local Bus Interface 14
Table 7-3 Local Bus Interface Interrupt Signals 14
Table 7-4 Local Bus Interface DMA Control and Status Signals 15
Table 7-5 PCI Bridge Signals 16
Table 7-6 LBI_access Data Types 17
Table 7-7 LBI_access Procedures 17
Table 7-8 Local Bus Interface Simulation Model Additional I/O 17
Table 7-9 Local Bus Interface Simulation Model Generic Inputs 18

1.0 Introduction

1.1 Contents and Structure

This manual describes the FPGA cores supplied with the Channel Adapter and Channel Adapter Plus hardware. This publication supplements the supporting documents listed in section 1.2 to provide a complete description of the capabilities and operation of the product. The manual is focused on the FPGA developer, other Channel Adapter manuals provide information on the hardware and software aspects of the product.

Section	Description
Section 1	Introductory information about the manual.
Section 2	Overview of the Channel Adapter FPGA core library.
Section 3	Description of the DCM core.
Section 4	Description of the ADC Capture core.
Section 5	Description of the QDR SRAM core.
Section 6	Description of the IDELAYCTRL core.
Section 7	Description of the Local Bus Interface core.
Section 8	Instructions for programming the FPGA and PROM.

The latest product documentation and software is available for download from the Red Rapids web site (www.redrapids.com) by following the Technical Support link.

1.2 Related Publications

Author	Number	Title
Red Rapids	REF-314-000	Model 314 Hardware Reference Manual
Red Rapids	REF-316-000	Model 316 Hardware Reference Manual
Red Rapids	REF-320-000	Model 320 Hardware Reference Manual
Red Rapids	REF-321-000	Model 321 Hardware Reference Manual
Red Rapids	REF-314-001	Model 314 Installation Guide
Red Rapids	REF-316-001	Model 316 Installation Guide
Red Rapids	REF-320-001	Model 320 Installation Guide
Red Rapids	REF-321-001	Model 321 Installation Guide
Red Rapids	REF-320-002	Channel Adapter Software Reference Manual
PCI SIG	PCI Rev 2.2	PCI Local Bus Specification

1.3 Conventions

This manual uses the following conventions:

- Hexadecimal numbers are prefixed by “0x” (e.g. 0x00058C).
- *Italic* font is used for names of registers.
- **Bold** font is used for names of directories, files and OS commands.
- Palatino font is used to designate source code.
- Active low signals are followed by ‘#’, For example, TRST#.

☞	Text in this format highlights useful or important information.

!	Text shown in this format is a warning. It describes a situation that could potentially damage your equipment. Please read each warning carefully.

The following are some of the acronyms used in this manual.

- **ADC** Analog to Digital Converter
- **API** Application Program Interface
- **CMC** Common Mezzanine Card
- **CPCI** CompactPCI
- **DAC** Digital to Analog Converter
- **DCM** Digital Clock Manager
- **DMA** Direct Memory Access
- **GPIO** General Purpose Input/Output
- **HAL** Hardware Abstraction Layer
- **IDELAY** Virtex-4 Input Delay Element
- **IDELAYCTRL** Virtex-4 Input Delay Control Element
- **IOB** Virtex-4 Input/Output Block
- **MSPS** Mega Samples per Second
- **PCI** Peripheral Component Interconnect
- **PMC** PCI Mezzanine Card
- **QDR** Quad Data Rate
- **SFDR** Spur Free Dynamic Range
- **SINAD** Signal-to-Noise and Distortion
- **SNR** Signal-to-Noise Ratio
- **TCXO** Temperature Compensated Crystal Oscillator

1.4 Manual Compatibility

The applicable hardware part numbers are defined as follows:

- Model 314-XXX (Channel Adapter 14/125)
- Model 316-XXX (Channel Adapter Plus 8/1500)
- Model 320-XXX (Channel Adapter 12/213)
- Model 321-XXX (Channel Adapter Plus 16/130)

1.5 Revision History

Version	Date	Description
R02	11/17/06	Updated SRAM core and Section 8.
R01	11/10/06	Updates.
R00	10/17/06	Initial release.

2.0 FPGA Core Library

The Channel Adapter product family is supported by a library of FPGA cores that interface user application logic to external hardware functions. VHDL source code is provided for all of the cores except the Local Bus Interface. This core is supplied as a Xilinx NGC file, which is a netlist that contains both logical design data and constraints. A VHDL wrapper is included to integrate this core with user logic.

Channel Adapter FPGA Core Library					
VHDL File Name (.vhd)	**314**	**316**	**320**	**321**	**Description**
adc_dcm_m314	✓				DCM configured to generate Model 314 clocks.
adc_dcm_m316		✓			DCM configured to generate Model 316 clocks.
adc_dcm_m320			✓		DCM configured to generate Model 320 clocks.
adc_dcm_m321				✓	DCM configured to generate Model 321 clocks.
adc_capture_m314	✓				Interface to Model 314 ADC input.
adc_capture_m316		✓			Interface to Model 316 ADC input.
adc_capture_m320			✓		Interface to Model 320 ADC input.
adc_capture_m321				✓	Interface to Model 321 ADC input.
tel_ad	✓				Interface to Model 314 telemetry ADC input.
qdrsram_iface	✓	✓	✓	✓	Interface to QDR SRAM.
idelay_ctl_block	✓	✓	✓	✓	Calibrates the SRAM interface IDELAY values.
Local_Bus_Interface_Wrapper	✓	✓	✓	✓	Interface to PCI bridge chip with DMA controller.

Note: A check mark signifies the relevance of each core to a corresponding product number (e.g. Model 314).

Figure 2-1 illustrates a typical interconnect between the FPGA application logic and the various cores that are available. The local bus interface core is required in virtually every application. A complete design example using this core is supplied with the product, refer to the *Channel Adapter Software Reference Manual* for further details.

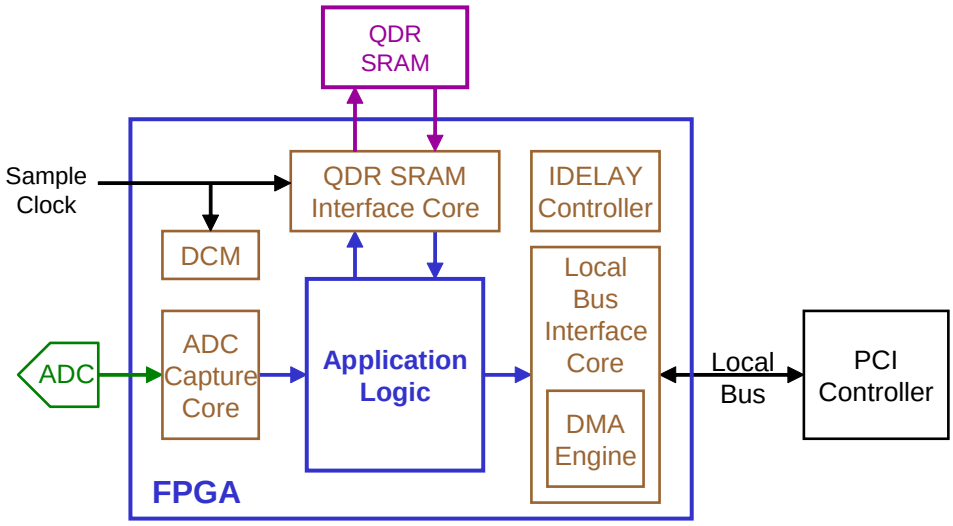

Figure 2-1 Core Interconnect

Configuring the Channel Adapter for basic signal acquisition requires only three FPGA cores as shown in Figure 2-2. The application logic consists of nothing more than a FIFO and an address decoder. This minimal design is a good starting point for users to become familiar with the operation of these primary cores before tackling more complex functions.

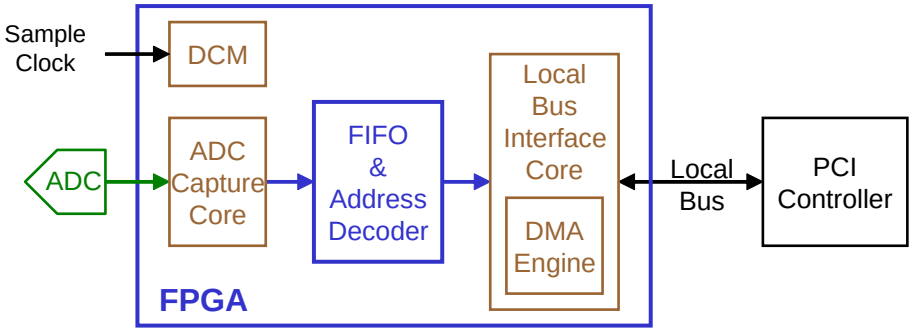

Figure 2-2 Signal Acquisition Function

The on-board sample clock will produce a continuous acquisition data rate that may be greater than the available PCI bus bandwidth. This problem can be eliminated by simply capturing snapshots of data into the FIFO. Otherwise the application logic must include a data reducing function (filter) to lower the sustained rate.

☞	Keep your first FPGA design simple so that you can focus on the operation of the cores without the overhead of complex application logic to consider.

3.0 DCM Core (adc_dcm_mXXX)

The DCM core is simply a standard Xilinx Digital Clock Manager primitive that has been customized with a phase offset to meet the timing requirements at the ADC Capture core interface. There is a different DCM core provided for each Channel Adapter product since each ADC has unique output timing characteristics. Refer to the Xilinx documentation for further information about using the DCM in a design.

4.0 ADC Capture FPGA Core (adc_capture_mXXX)

The ADC capture FPGA core provides access to sample data produced by the digitizer. The interface consists of two primary signal groups; the user interface and the hardware interface. The user interface signals provide the connection to application logic inside the FPGA. The hardware interface consists of the physical connections between the FPGA and the ADC.

4.1 ADC Capture FPGA Core - User Signals

The receiver user interface signals can be broken into three categories, resets/clocks, control interface, and data/status. Each signal group is discussed in more detail in the following sections.

4.1.1 Resets and Clocks

There is one reset and one clock associated with the receiver user interface as described in Table 4-1.

Table 4-1 ADC User Interface Reset and Clock Signals

Signal Name	Direction	Description
reset	Input	Logic one places all registers in reset.
clk	Input	Phase adjusted ADC sample clock.

The reset signal is used to clear all of the registers and state machines in the interface.

The clk input is supplied by a DCM that phase adjusts the ADC sample clock input to meet timing constraints on the data inputs. A pre-configured DCM core is available for each Channel Adapter product to achieve the correct clock settings. The clock is used in combination with the registers located in the IOBs to capture the ADC data.

4.1.2 Control Interface

The control interface provides the vehicle to enable and disable ADC data capture and to inject a synthetic data stream (binary count) for testing purposes. A list of signals is provided in Table 4-2.

Table 4-2 ADC User Interface Control Register Interface Signals

Signal Name	Direction	Description
enable	Input	Logic zero masks the sync input (see below).
sync	Input	Logic high starts the capture of ADC data if the enable is active (logic one).
sim_enable	Input	Logic one replaces the ADC input with a binary counter to provide a deterministic data pattern.

The combination of an enable and sync input allows individual channels to respond to a common synchronization trigger without interrupting other channels. This feature can be useful in applications that dynamically allocate resources.

4.1.3 Data and Status

The data and status pins listed in Table 4-3 provide the operating interface to the front-end of every Channel Adapter. Data from the ADC is provided on data_out coincident with the clk signal described in Section 4.1.1.

Table 4-3 ADC User Data and Status Signals

Signal Name	Direction	Description
data_valid16	Output	Logic one indicates that data is valid on the lower 16 bits of data_out (15:0).
data_valid32	Output	Logic one indicates that data is valid on the lower 32 bits of data_out (31:0).
data_valid64	Output	Logic one indicates that data is valid on the lower 64 bits of data_out (63:0).
adc_or	Output	Over-range flag from the ADC.
data_out[63:0]	Output	Data bus carrying ADC samples.

There are multiple data_valid signals provided so that the user can extract data from the core in a size that matches the next stage of processing. Data is sign extended to the nearest 8-bit boundary and packed with the most recent sample in the lowest word of the data_out bus.

4.2 ADC Capture FPGA Core - Hardware Signals

All of the signals connecting the FPGA to the ADC are listed in Table 4-4 for the Model 314 and Table 4-5 for the Model 320. This list is provided for reference only, the core manages this interface independently.

Table 4-4 Model 314 Receiver Interface Signals

Signal Name	Direction	Description
adc_data_in[13:0]	Input	Data bus from the ADC.
adc_or	Input	LVDS negative ADC data in. Data from the ADC.

Table 4-5 Model 320 Receiver Interface Signals

Signal Name	Direction	Description
adc_data_in_p[11:0]	Input	Positive side of the LVDS ADC data bus.
adc_data_in_n[11:0]	Input	Negative side of the LVDS ADC data bus.
adc_or_p	Input	Positive side of the LVDS over-range flag.
adc_or_n	Input	Negative side of the LVDS over-range flag.

5.0 QDR SRAM Interface FPGA Core (qdrsram_iface)

The SRAM interface FPGA core simplifies user interaction with the memory by managing all of the critical timing, including a self-calibration procedure that runs each time the hardware is reset. The self-calibration procedure adjusts the IDELAY elements that are resident in each IOB to compensate for path delay differences in the clock, data, and control signals.

The architecture of the SRAM is presented to the user as a single data rate 32-bit dual-port memory instead of the more complicated 16-bit quad data rate structure of the actual device. There is no loss of memory bandwidth through the core by making this translation. The read and write ports operate independently; each has a dedicated clock, data bus, and control flow.

Unpredictable results can occur if a read and write transaction are simultaneously posted to the same address of the dual port. The user application logic should allow two SRAM clock periods for data to be written before reading from the same address.

The SRAM interface core consists of two primary signal groups; the user interface and the hardware interface. The user interface signals provide the connection to application logic inside the FPGA. The hardware interface consists of the physical connections between the FPGA and the QDR II SRAM chip.

5.1 SRAM Interface FPGA Core - User Signals

The SRAM user interface signals can be broken into three categories, resets/clocks, write port, and read port. Each signal group is discussed in more detail in the following sections.

5.1.1 Resets and Clocks

There is one reset and three clocks assigned to the SRAM user interface as described in Table 5-1.

Table 5-1 SRAM User Interface Reset and Clock Signals

Signal Name	Direction	Description
reset	Input	Active high reset.
enable	Input	Logic 1 enables SRAM DLL, Logic 0 puts SRAM in power down. Used to reset SRAM after sramclk becomes valid.
wrrclk	Input	Write clock to the user interface side of the SRAM core. All wr* signals are registered on this clock. Must not be faster than sramclk (250 MHz maximum.)
rdclk	Input	Read clock to the user side of the SRAM core. All rd* signals are registered on this clock. Must not be faster than sramclk (250 MHz maximum.)
sramclk	Input	Interface clock to the SRAM, must be between 125 – 250 MHz.
sramclk_90	Input	Interface clock to the SRAM (sramclk) phase shifted by 90 degrees. Used to center SRAM input clock with respect to SRAM inputs.

The reset signal is used to clear registers internal to the core and also reset the DLL inside the QDR II SRAM chip. Do not hold reset active for more than 200 us. The

SRAM data outputs are tristate during DLL reset, which results in an indeterminate logic state at the FPGA inputs.

Separate clocks are provided for the read and write ports to the SRAM. These can be set to any frequency less than or equal to the clocks that are distributed to the QDR II SRAM chips. The core manages the clock domain crossing.

5.1.2 Auto Calibration Port

In order to compensate for variations over physical characteristics, temperate and voltage the SRAM controller automatically centers the incoming data from the SRAM. After the SRAM is enabled an auto calibration process must be done prior to read or write operations.

Table 5-2 SRAM Auto Calibration Interface Signals

Signal Name	Direction	Description
cal_start	Input	Logic 1 start auto calibration process. Should not be asserted until sramcal / sramclk_90 are valid and the SRAM has been enabled for at least 1024 sramclk ticks.
cal_done	Output	Logic 1 indicates the calibration process is done and the read / write ports can be used.
cal_clk	Output	Phase aligned with sramclk at one fourth the rate.

5.1.3 Write Port

Writing data to the SRAM is accomplished in a single clock cycle by supplying the signals listed in Table 5-3.

Table 5-3 SRAM User Interface Control Register Interface Signals

Signal Name	Direction	Description
wren	Input	Logic one captures the current write address and data input and posts them to the write port of the SRAM.
wraddr[21:0]	Input	Address to be written. The memory is addressed on 32-bit boundaries, so the LSB must always be zero.
wrdata[31:0]	Input	Data value to be written.
wrben[3:0]	Input	Byte Enables (Active Low) B3 B2 B1 B0. Logic zero allows the specified byte to be written the SRAM.

5.1.4 Read Port

The SRAM signals assigned to the read port are listed in Table 5-4.

Table 5-4 SRAM User Data and Status Signals

Signal Name	Direction	Description
rden	Input	Logic one captures the current read address and posts it to the read port of the SRAM.

rdaddr[21:0]	Input	Address to be read. The memory is addressed on 32-bit boundaries, so the LSB must always be zero.
rddata[31:0]	Output	Data value returned from the SRAM address requested. Data is only valid when the rdvalid flag is logic one.
rdvalid	Output	Logic one indicates that data requested from a read operation is available on the rddata bus. This flag and the associated data are available for one clock period.
rdack	Input	Logic 1 acknodleges the data on the rddata port and allows data returned from subsequent reads to appear on the rddata port.

A read operation cannot be completed in a single clock cycle due to latencies through the QDR SRAM chip. The process begins by supplying an address along with an active enable (rden) for a single clock cycle. The core posts the read transaction to the SRAM and signals the return of the requested data with the valid flag (rdvalid).

It will take no more than six SRAM clock periods for the core to complete a read cycle. There is no need to wait for a return value before posting another transaction. Read operations can be executed continuously, but the user logic must associate each return value with the correct address.

5.2 SRAM Interface FPGA Core - Hardware Signals

All of the signals connections between the FPGA and SRAM are listed in Table 5-5. This information is provided for reference only, the core manages this interface independently.

Table 5-5 SRAM Interface Signals

Signal Name	Direction	Description
qdr_clkp	Output	QDR SRAM clock. (Period: 4.0 ns to 8.4 ns)
qdr_clkn	Output	QDR SRAM clock complement.
qdr_eclk	Input	QDR SRAM echo clock.
qdr_wn	Output	Write signal to QDR SRAM, active low.
qdr_rn	Output	Read signal to QDR SRAM, active low.
qdr_doffn	Output	QDR SRAM DLL power down, active low.
qdr_sa[21:0]	Output	QDR SRAM address bus, write address captured on rising edge, read on falling edge.
qdr_d[15:0]	Output	QDR SRAM input data bus.
qdr_q[15:0]	Input	QDR SRAM outdata data bus.
qdr_bwn[1:0]	Output	Byte write select, always set to 2b00.

6.0 IDELAYCTRL FPGA Core

The SRAM interface core relies on the IDELAY feature of the IOB to optimize timing. The IDELAYCTRL primitive must be instantiated to maintain calibration of the individual IDELAY elements. This primitive is part of the FPGA fabric and does not connect to any other logic.

The IDELAYCTRL requires a 200 MHz reference clock to operate. This clock is named idelayref_clk in our example designs. This 200 MHz clock is produced by multiplying either the Local Bus clock (or the 2X the Local Bus clock if used in a 33 MHz system) by three.

Table 5-6 IDELAYCTL Clocks and Resets

Signal Name	Direction	Description
dcm_reset	Input	Logic 1 resets the DCM. This should be done anytime the clk_sel port changes.
Idly_reset	Input	Logic 1 resets the IDELAYCTRL. This should be done after the DCM locks anytime the DCM is reset.
user_clk	Input	1X version of Local Bus clock.
user_clk_2x	Input	2X version of Local Bus clock.
clk_sel	Input	Logic 0 selects user_clk as input to DCM, logic 1 selects user_clk_2x.
idelayref_clk	Output	Selected input clock x3
locked	Output	DCM locked status, logic 1 indicates locked.
readyN	Output	Logic 0 indicates IDELAYCTLR is calibrated.

7.0 Local Bus Interface FPGA Core (Local_Bus_Interface_Wrapper)

The local bus interface FPGA core provides the functions needed to efficiently transfer data from the FPGA to host memory. It includes an integrated bus master DMA engine and handles all control of the PCI Controller chip.

The local bus interface core consists of two main signal groups; the bridge interface, and the user interface. The user interface signals are available to connect with other logic inside the FPGA. These signals comprise the local bus interface to the user application logic. The bridge interface includes all of the connections between the FPGA core and the PCI Controller chip. These signals represent actual physical connections between the devices.

The **addr_mem.xco** and **tx_dma_fifo.xco** files supplied on the distribution disk must be regenerated in the Xilinx project to use the local bus interface core.

7.1 Local Bus Interface FPGA Core - User Signals

The user signals can be broken into four logical groups; resets and clocks, local bus interface, interrupts, and DMA control and status. The following sections provide detailed signal descriptions for each group.

7.1.1 Resets and Clocks

The user interface includes one reset and one clock signal as described in Table 7-1. The signal named rst is equivalent to the PCI bus RST#, but is active high instead of active low. This signal is supplied to the FPGA from the PCI Controller chip and can be used as a master reset in user applications.

The PCI interface also provides a bus clock to synchronize internal logic to the remaining user signals of the core. The frequency is either 33 MHz or 66 MHz depending on the capability of the host PCI bus.

Table 7-1 Local Bus User Interface Reset and Clock Signals

Signal Name	Direction	Description
rst	Output	Active high system reset available to FPGA logic. The core will also enter reset when this signal is logic one.
srst	Input	Active high software reset. This can be implemented via a register in the FPGA logic to reset the core without resetting configuration registers.
PCIclk	Output	Either a 33 MHz or 66 MHz clock from the DMA Engine. All signals to and from the DMA Engine are synchronous to this clock.
PCIclk_2x	Output	2x PCIclk, used for input clock to IDELAYCTLR logic when in a 33 MHz system.

7.1.2 Local Bus Interface

The signals listed in Table 5-2 are used to connect internal logic to the PCI Controller chip. A PCI initiated transaction begins when Local_Data_ADS_N falls to a logic zero, signaling that there is a valid address on Local_Data_Addr [31:0]. The internal decoder must first determine whether the access is a read or write by evaluating the

state of the Local_Data_Rd_N and Local_Data_Wr_N bits. If the transaction is a read, data from the requested address should be loaded on the Local_Data_Out [31:0] bus. If the transaction is a write, data from the Local_Data_In [31:0] bus should be registered into the correct local address.

Table 7-2 DMA Engine Local Bus Interface

Signal Name	Direction	Description
Local_Data_Out [63:0]	Input	Bus used to transfer data out of the FPGA (read). For non-DMA transfers only the 32 LSBs are used.
Local_Data_Rdy_N	Input	During a read transaction, a logic zero indicates that the data on the Local_Data_Out bus is valid. During a write transaction, a logic zero indicates that the internal decoder is ready for the next sample to be presented on the Local_Data_In bus.
Local_Data_Addr [31:0]	Output	Local address where data should be read or written.
Local_Data_ADS_N	Output	Logic zero indicates the start of a bus transaction.
Local_Data_In [63:0]	Output	Bus used to transfer data into the FPGA (write). For non-DMA transfers only the 32 LSBs are used.
Local_Data_Rd_N	Output	Logic zero indicates the current transaction is a read operation to an internal FPGA register.
Last_Brst_Wd_N	Output	Logic zero indicates the last data transfer of a local bus transaction.
Local_Data_Wr_N	Output	Logic zero indicates the current transaction is a write operation to an internal FPGA register.

Local_Data_Rdy_N must be driven low for a single PCIclk clock period to indicate that the read data on Local_Data_Out is valid or that the write data on Local_Data_In has been captured. This action will increment the Local_Data_Address by four bytes, allowing the decoder to respond to the next read or write operation of a burst transaction. The final transfer will be flagged by a logic zero on Last_Brst_Wd_N.

7.1.3 Interrupts

The local bus interface core provides a simple mechanism for sourcing interrupts to the host processor using the signal listed in Table 7-3. Driving the PCI_int signal to a logic zero will cause the bridge chip to issue an interrupt if it has not been masked. The PCI_int signal must be driven low until the interrupt is serviced.

Table 7-3 Local Bus Interface Interrupt Signals

Signal Name	Direction	Description
PCI_int	Input	Logic zero sends an interrupt to the host.

7.1.4 DMA Control and Status

The DMA control and status signals, listed in 0, provide the user interface to the bus master DMA functions. The DMA engine occupies 383 contiguous addresses in the local FPGA memory map starting at the value specified by the Base_Addr [31:0] input to the core. The least significant 11 bits of this bus should be set to zero.

Configuration registers within the local bus interface core are preloaded by the host with the block size, group size, and memory addresses for DMA operations. There are

actually two sets of registers to support independent transfers on the DMA channels. The DMA_Ch_BlkSize busses convey the current values stored in the block size registers. These are needed by the FPGA application code to set the number of words transferred in each burst. The group size and memory addresses are only needed by the DMA engine and are not supplied to the user interface.

DMA transactions are initiated by the FPGA application code when a block of data is ready to be transferred to the host. The most efficient use of PCI bandwidth is achieved with scatter-gather operations. The DMA engine to uses a different host memory address for each DMA group. There are 2048 addresses assigned to the receiver which are divided evenly across the total number of channels in the Local Bus Interface core.

The DMA_Channel signal indicates which of the receiver channels is requesting the next data transfer. This information is used by the DMA engine to select the appropriate set of registers to configure the operation.

The DMA_Direction and DMA_SG_En signals are not used in the Model 320 and should be tied to logic zero.

The DMA_Go input signal is used to start a DMA operation and the DMA_Busy status bit indicates that a DMA operation is in progress. Driving the DMA_Go signal to a logic one for a single 33 MHz clock period will initiate a new transfer as long as there is no current operation in progress. The DMA_Channel must be held constant throughout the entire operation, from initiation with DMA_Go to completion as indicated by DMA_Busy transitioning to logic zero.

Table 7-4 Local Bus Interface DMA Control and Status Signals

Signal Name	Direction	Description
Base_Addr [31:0]	Input	The base address of the DMA Engine in the FPGA memory map. Bits [8:0] should be tied to logic zero. The DMA engine will occupy the addresses from Base_Addr to Base_Addr + 0x14C.
DMA_Channel	Input	Logic zero indicates that the DMA should use Channel A control information (address, block size, group size, etc.) Logic one indicates that the DMA should use Channel B control information.
DMA_Go	Input	Logic one applied for one 33 MHz clock period while DMA_Busy is logic zero will start DMA processing.
DMA_SG_En	Input	Tie to logic zero.
DMA_Busy	Output	Logic one indicates that a DMA transaction is in process. All DMA_Go signals received while DMA_Busy is logic one are ignored.
DMA_Direction	Input	Tie to logic zero.
DMA_Ch*X*_BlkSize [15:0]	Output	Size of a Channel DMA block transfer in 64-bit words (8 bytes). *X* indicates the channel associated with the block size.
DMA_Interrupt Status	Output	Indicates that an interrupt has been issued by the DMA core.
DMA_Ch_Last_Blk	Output	Logic one on any bit indicates that the current DMA block will complete a group transfer for the Channel associated with that bit. Bits 0 to 31 correspond to

		DMA channels 0 to 31. For lower channel count cores only the LSBs are utilized.

The DMA Interrupt Status flag indicates that a bit in the *PCI Interface Interrupt Status* register has been set. This should be tied through any application specific interrupt logic to the PCI_int pin of the core.

The DMA_Ch_Last_Blk status bits provide a convenient mechanism for inserting header data at the start of each new group. This can be used to insert time stamps or other periodic status information. There is no requirement to use these signals in an application.

7.2 Local Bus Interface FPGA Core - Hardware Signals

All of the signals between the FPGA and the PCI controller are listed in Table 7-5. This list is provided for reference only, the core manages this interface independently.

Table 7-5 PCI Bridge Signals

Signal Name	Direction	Description
LBClk	Input	Local bus clock
LBRstn	Input	Logic zero indicates a reset to the FPGA
LB_user_req	Input	Logic one indicates that the PCI Interface is requesting a transaction
LB_user_mult	Input	Logic one signals a multiple 32-bit word transfer
LB_user_rdwr	Input	Logic zero signals write cycle, logic one signal read cycle.
LB_user_addr_vld	Input	Address/data start.
LB_user_stop	Output	Burst terminate.
bus_select	Output	Selects data source from the PCI interface.
LB_bus_cs	Output	Logic one indicates valid read data or increments the address/data on a write transaction.
fifo_flag	Input	Status of PCI interface FIFOs
LBAD	Inout	Address
D	I/O	Data bus
Int_to_FPGA	Input	Interrupt from PCI interface to indicate completion of a DMA.
Int_to_Bridge	Output	Interrupt to be passed to the PCI bus.

7.3 Local Bus Interface Simulation Model

The Channel Adapter Local Bus Interface Simulation Model provides an environment to verify logic function in the user programmable portion of the Channel Adapter products.

The two VHDL files contained in the model, **LBI_access.vhd** and **Local_Bus_Interface.vhd,** are described in the following sections along with a description of how the model operates.

7.3.1 LBI_access.vhd

The **LBI_access.vhd** file is a VHDL package that contains procedure calls and data types used in the local bus interface simulation model. This package will need to be included in any files that use local bus interface data types or that call the local bus interface procedures. A list of the data types is found in Table 7-6. A list of the procedures is found in Table 7-7.

Table 7-6 LBI_access Data Types

Data Type	Description
DATA_ARRAY_TYPE	An array of 64-bit logic vectors 512 vectors deep.
REG_ARRAY_10x32	An array of 10-bit logic vectors 32 vectors deep.
REG_ARRAY_16x32	An array of 16-bit logic vectors 32 vectors deep.
REG_ARRAY_19x32	An array of 19-bit logic vectors 32 vectors deep.

Table 7-7 LBI_access Procedures

Procedure	Description
reset	Reset the simulation model and send a reset to the user logic.
read32	Read from a 32-bit register from the user logic.
write32	Write to a 32-bit register in the user logic.

7.3.2 Local_Bus_Interface.vhd

The **Local_Bus_Interface.vhdl** file contains procedures and logic that mimics the local bus interface core. Replacing the local bus interface core instantiation with the simulation model will provide an environment to test the functionality of user logic.

7.3.2.1 Local_Bus_Interface.vhd Signals

The local bus interface simulation model contains some interconnects not found in the local bus interface core. The additional I/O is described in Table 7-8. These signals will have to be tied to ports that map through the hierarchy to the level where the LBI_access procedures are called.

Table 7-8 Local Bus Interface Simulation Model Additional I/O

Signal Name	Direction	Description
start_bit	input	Indicates the start of a transaction in the local bus interface simulation model. This signal is set in the provided procedures and should not be set by the user other than initializing it to logic zero in the testbench.
done_bit	output	Indicates the end of a transaction in the local bus interface simulation model. This signal is set in the provided procedures and should not be set by the user other than initializing it to logic one in the testbench.
command[3:0]	input	A 4-bit signal that is set in the provided procedures and should not be set by the user other than initializing it to logic zero in the testbench.
address [31:0]	input	A 32-bit signal that sets the address of the register in the user defined logic that is going to be accessed.
data [3:0]	input	A 32-bit signal that sets the value to be written to a register or the expected return value of a register read.
data_array [63:0]	input	A DATA_ARRAY_TYPE signal that indicates expected DMA data.

7.3.2.2 Local_Bus_Interface.vhd Generic Signal Definitions

The local bus interface simulation model defines generic inputs to set core parameters such as DMA block and group sizes and the number of groups between interrupt signals. The generic inputs can be defined through a generic map in the **Local_Bus_Interface.vhd** instantiation or they can be modified in the **Local_Bus_Interface.vhd** file. The signals are defined in Table 7-9.

Table 7-9 Local Bus Interface Simulation Model Generic Inputs

Signal Name	Description
BlockSize	A REG_ARRAY_19x32 signal that defines the DMA block size in bytes for all 32 channels.
GroupSize	A REG_ARRAY_16x32 signal that defines the DMA group size in number of blocks for all 32 channels.
GroupsPerInt	A REG_ARRAY_10x32 signal that defines the number of DMA groups that will complete before an interrupt is generated. The value is zero indexed so zero indicates one group per interrupt, one indicates two groups per interrupt, and so on.

7.3.3 Local Bus Interface Simulation Model Operation

The local bus interface core is capable of five different transaction types: resets, target read operations, target write operations, DMA transfers, and interrupts. The simulation model is capable of mimicking all types of transactions through procedures and internal logic.

7.3.3.1 Resets

Resets are handled by the LBI_access reset procedure outlined below.

Prototype	reset(Clk, start_bit, command, done_bit);
Arguments	Clk – Local bus clock signal. start_bit – Signal that is tied to the Local_Bus_Interface.vhd start_bit input. command – Signal that is tied to the Local_Bus_Interface.vhd command input. done_bit – Signal that is tied to the Local_Bus_Interface.vhd done_bit input.
Description	Drives a reset signal on the Local_Bus_Interface.vhd Rst output.

7.3.3.2 Target Read Operations

Target read operations are always broken up into single 32-bit register accesses by the local bus interface core. This makes them indistinguishable from single 32-bit read operations. The LBI_access read32 procedure provides a mechanism to access registers in the user defined logic.

Prototype	read32(Clk, address, data, start_bit, command, done_bit);
Arguments	Clk – Local Bus clock signal. address – Signal that is tied to the Local_Bus_Interface.vhd address input indicating the address of the register to be read. data – Signal that is tied to the Local_Bus_Interface.vhd data input indicating the expected return data from the register read. start_bit – Signal that is tied to the Local_Bus_Interface.vhd start_bit input. command – Signal that is tied to the Local_Bus_Interface.vhd command input. done_bit – Signal that is tied to the Local_Bus_Interface.vhd done_bit input.
Description	Reads data from the register that is located at the address specified and compares it to the data provided. If the data does not match, an error message is displayed and the simulation is halted. The behavior of a mismatch can be modified by changing lines 216 and 217 of the Local_Bus_Interface.vhd file.

7.3.3.3 Target Write Operations

Target write operations are always broken up into single 32-bit register accesses by the local bus interface core. This makes them indistinguishable from single 32-bit write operations. The LBI_access write32 procedure provides a mechanism to access registers in the user defined logic.

Prototype	writ32(Clk, address, data, start_bit, command, done_bit);
Arguments	Clk – Local Bus clock signal. address – Signal that is tied to the Local_Bus_Interface.vhd address input indicating the address of the register to be written. data – Signal that is tied to the Local_Bus_Interface.vhd data input indicating the data to be written to the register start_bit – Signal that is tied to the Local_Bus_Interface.vhd start_bit input. command – Signal that is tied to the Local_Bus_Interface.vhd command input. done_bit – Signal that is tied to the Local_Bus_Interface.vhd done_bit input.
Description	Writes the supplied data to the register that is located at the address specified.

7.3.3.4 DMA Transfers

DMA transfers are initiated by the FPGA, so the DMA function of the local bus simulation model is handled by logic within the **Local_Bus_Interface.vhd** file. A DMA is initiated in the simulation model identically to DMA initiation on the local bus interface core.

The DMA provides two methods for checking the data. DMA data is automatically checked against the data_array input and an error is indicated if the data does not match, but the simulation will not halt. Each DMA will restart the comparison with the 64-bit sample located in the zero index of the data_array input. Since DMA transfers are initiated by the user logic, it is up to the user to update the data_array

values at the appropriate times if this error checking is desired. The data_array depth can be adjusted in the **LBI_access.vhd** file at line 20.

DMA data is also appended to a text file called **channelXdata.txt** where "X" is the channel number 0 to 31. The files need to be deleted or renamed between simulation runs otherwise the new data will be added to the end of the existing file. The default file format has one 64-bit sample per file line displayed as a bit vector. This format can be modified by changing the line written at line 336 in the **Local_Bus_Interface.vhd** file.

7.3.3.5 Interrupts

Interrupts are generated by the Local Bus Interface whenever a specified number of groups has been transferred. The number of groups between interrupts is specified by the *GroupsPerInt* array with an array index for each channel (0 to 31). When the specified number of groups has been transferred for any of the channels, the DMA_Interrupt_Status output will transition to a logic one for use in any user defined interrupt registers. This status can be cleared by doing a *read32* access to address 0x10 with any data value.

8.0 FPGA and Configuration PROM Programming

The FPGA and Configuration PROM are user programmable devices that can be loaded with application logic directly from the host through the PCI Controller or through the JTAG pins on the back of the card. As shown in Figure 7-1, the entire JTAG chain is composed of the Configuration PROM followed by the FPGA.

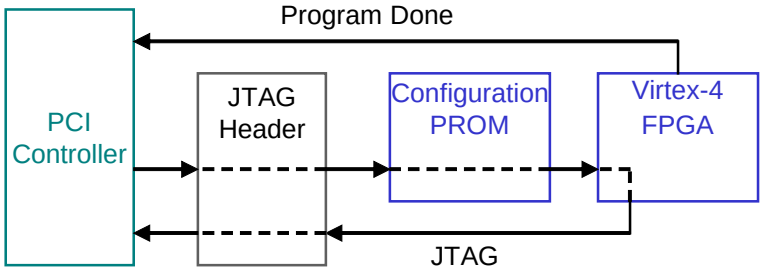

Figure 7-1 JTAG Chain

The Channel Adapter product includes Xilinx Serial Vector Format (XSVF) programming files for the default FPGA configuration (diagnostic) preloaded in the Configuration PROM as well as source and XSVF files for the Local Bus example. These files are meant to aid in the development of application specific FPGA designs. The FPGA source code package includes VHDL source, Xilinx CORE Generator™ files, the DMA Engine core, and a User Constraint File (.ucf).

Sections 0 and 0 summarize the process of creating and compiling an FPGA using the Xilinx ISE software with the FPGA source code provided. Section 8.3 discusses how to load the XSVF file to the Channel Adapter. Section **Error! Reference source not found.** has a brief overview for creating custom FPGA code to work with the provided DMA Engine.

8.1 Setting up a Xilinx ISE Project for the Local Bus Example

This section details the steps needed to recompile the Local Bus example code. All paths are relative to the FPGA Source Disk, DSK-320-003-RXX

1. Launch ISE and select File / New Project. Enter CA_LBExample for the project name and specify a location for the project. Leave the source type set to HDL. Click next and select the correct FPGA type, package and speed grade based on your hardware.
2. Click Next to dismiss the Add New Source, Add Existing Source and Summary windows.
3. Select “Add Copy of Source” or “Add Source” and add ucf/CA_LBExampleTop.vhd and the appropriate LBExample UCF for your hardware.
4. Select “Add Copy of Source” and add all of the Xilinx CORE Generator™ (.xco) files in src/LocalBusCore/CoreGen and Local_Bus_Interface_Wrapper.vhd from src/LocalBusCore/4-channel.

5. Copy the appropriate Local Bus Engine (.ngc) from src/LocalBusCore/4-channel to the top level directory of your design. M314 and M320 built on R00 of the PWB requires the “5ns_ps” version of the engine, all others require the non phase shifted version.

> ☞ The Xilinx CORE Generator™ (.xco) and Local Bus Engine core files (.ngc) need to be located in the top level of the ISE project. Other source files can be copied to the project directory or referenced externally

6. Select **addr_mem.xco** in the source tree and select Regenerate Core. Repeat for **tx_dma_fifo.xco**.
7. Select “Generate Programming File” and click “Run” to compile the project and generate CA_LBExampleTop.bit.

8.2 Creating XSVF Programming Files

Creating the XSVF programming files is a two-step process. First a Configuration PROM file needs to be generated from a BIT file and then the XSVF file can be created. Each XSVF file can either update the contents of the PROM, which will then be used to load the bitstream to the FPGA on the next power cycle, or to leave the contents of the PROM unchanged and program the FPGA directly, The Channel Adapter FPGA development code distribution disk contains BIT files and the compilation reports for the Local Bus example for all of the supported FPGA targets in the `lbexample_firmware` directory.

8.2.1 Creating PROM File for M314 & M320

The Model 314 and Model 320 products use a single XCF PROM device to store the FPGA bitstream. For units with SX25, SX35 and LX40 FPGAs this is an XCF16P (16 Mbit) PROM. For units with LX60s the PROM is an XCF32P (32 Mbit PROM.)

To create the MCS files which will be used to program the PROM, launch the Xilinx iMPACT™ tool. This can be done by double clicking on the Generate PROM, ACE, or JTAG File in the Process View window or by opening the standalone iMPACT™ application.

If iMPACT™ is launched from within ISE, the wizard will open a window that asks which type of file to create. Click the button next to PROM File and click Next.

If iMPACT™ is launched as a separate application, set the mode to File Mode by clicking Mode->File Mode in the menus. Left click on the PROM Formatter tab and then right click in the window to launch the wizard.

In the Prepare Prom Files window, select the Xilinx PROM as a target with the MCS file format. Fill in the desired Checksum Fill Value, PROM File Name, and location where you want the file to be created and click the Next button at the bottom of the window.

In the Specify Xilinx Prom Device window, set the Select a PROM options by choosing xcf in the first pull-down field and the appropriately sized part in the second. Click the Add button and then click the Next button at the bottom of the window.

Verify that your settings are correct in the File Generation Summary window. Click the Next button at the bottom of the window.

In the Add Device File window, click the Add File button. Navigate to the project directory and select the **ChannelAdapterTop.bit** file. Click No in the window that asks, "Would you like to add another design file to Data Stream: 0?" Click the Finish button at the bottom of the window and click Yes when asked, "Do you want to generate file now?"

8.2.2 Creating XSVF File for M314 & M320

Launch the Xilinx iMPACT™ tool. This can be done by double clicking on the Generate PROM, ACE, or JTAG File in the Process View window or by opening the standalone iMPACT™ application.

If iMPACT™ is launched from within ISE, the wizard will open a window that asks which type of file to create. Click the Cancel button at the bottom of the window.

Set the iMPACT™ mode to File Mode by clicking Mode->File Mode in the menus. Click on the SVF-STAPL-XSVF tab. Right click in the window to start adding devices. A window will appear asking which type of Boundary Scan file to create click the buttons for Create and XSVF and click OK. Specify the name for the XSVF file to create and click Save. Click OK in File Generation Mode window confirming that all device operations will be directed to the XSVF file specified and then dismiss the first "Add a New Device" window.

Right click on the iMPACT™ workspace and select "Add New Xilinx Device." Navigate to and select the PROM file that was created in Section 8.2.1 and click open select the xcf16p or xcf32p option (as described in the MCS creation section) in the Part Name pull-down options and click OK.

Right click in the window and select Add Xilinx Device. Navigate to and select the **CA_LBExampleTop.bit** file created in Section 0. Click open and click OK when the Warning window appears indicating that the startup clock has been changed to 'JtagClk' for the bitstream.

Right click on the PROM device and select the operations that are desired for the XSVF file. A typical PROM programming XSVF can be created by right clicking the PROM icon and selecting Program and "Erase before programming" (the check box for "Verify" can also be checked if desired). To create an XSVF file for to program the FPGA right click on the FPGA icon and select Program. Click on OK to accept the defaults and generate the XSVF PROM programming file. It is not possible to create an XSVF file that loads both the PROM and the FPGA.

8.2.3 Creating PROM File for M316

The Model 316 have two XCP devices (an XCF32P and a XCF08P) to store the FPGA bitstream. For units with LX160 FPGAs both PROMs will be used, other devices will use only the XCF32P.

To create the MCS files which will be used to program the PROM, launch the Xilinx iMPACT™ tool. This can be done by double clicking on the Generate PROM, ACE, or JTAG File in the Process View window or by opening the standalone iMPACT™ application.

If iMPACT™ is launched from within ISE, the wizard will open a window that asks which type of file to create. Click the button next to PROM File and click Next.

If iMPACT™ is launched as a separate application, set the mode to File Mode by clicking Mode->File Mode in the menus. Left click on the PROM Formatter tab and then right click in the window to launch the wizard.

In the Prepare Prom Files window, select the Xilinx PROM as a target with the MCS file format. Fill in the desired Checksum Fill Value, PROM File Name, and location where you want the file to be created and click the Next button at the bottom of the window.

In the Specify Xilinx Prom Device window, set the Select a PROM options by choosing xcf in the first pull-down field and the XCF32P part in the second. Click the Add button and then add an XVF08P in the same manner. Now click the Next button at the bottom of the window.

Verify that your settings are correct in the File Generation Summary window. Click the Next button at the bottom of the window.

In the Add Device File window, click the Add File button. Navigate to the project directory and select the **CA_LBExampleTop.bit** file. Click No in the window that asks, "Would you like to add another design file to Data Stream: 0?" Click the Finish button at the bottom of the window and click Yes when asked, "Do you want to generate file now?" If you are no targeting an LX160 you will receive a warning that the XCF08P PROM is unused and no programming file will be created.

8.2.4 Creating XSVF File for M316

Launch the Xilinx iMPACT™ tool. This can be done by double clicking on the Generate PROM, ACE, or JTAG File in the Process View window or by opening the standalone iMPACT™ application.

If iMPACT™ is launched from within ISE, the wizard will open a window that asks which type of file to create. Click the Cancel button at the bottom of the window.

Set the iMPACT™ mode to File Mode by clicking Mode->File Mode in the menus. Click on the SVF-STAPL-XSVF tab. Right click in the window to start adding devices. A window will appear asking which type of Boundary Scan file to create click the buttons for Create and XSVF and click OK. Specify the name for the XSVF file to create and click Save. Click OK in File Generation Mode window confirming that all device operations will be directed to the XSVF file specified and then dismiss the first "Add a New Device" window.

Right click on the iMPACT™ workspace and select "Add New Xilinx Device." Navigate to and select the PROM file that was created in Section 8.2.3 and click open select the file for the 8 Mbit PROM (`filename_1.mcs`) if one was created. Otherwise insert an XCF08P as a placeholder. Right click in the window and select Add Xilinx device, and select the file for the 32 Mbit PROM (filename_0.mcs) and select XCF32 option (as described in the MCS creation section) in the Part Name pull-down options and click OK.

Right click in the window and select Add Xilinx Device. Navigate to and select the **CA_LBExampleTop.bit** file created in Section 8.1. Click open and click OK when the Warning window appears indicating that the startup clock has been changed to 'JtagClk' for the bitstream.

Right click on the devices and select the operations that are desired for the XSVF file. A typical XSVF to program the PROMs can be created by right clicking on each PROM icon and selecting Program, Erase before programming (Verify can also be selected if desired). To create an XSVF file for the FPGA right click on the FPGA icon and select Program. Click on OK to accept the defaults and generate the programming file. It is not possible to create an XSVF file that loads both the PROM and the FPGA.

8.3 Loading XSVF Files

XSVF files can be loaded by running the *Adapter_LoadXSVF* sample code described in the Channel Adapter 312/213 Operating Guide.

8.4 Programming via the JTAG connector

The FPGA and/or configuration PROM can be programmed via the JTAG interface using the Xilinx iMPACT™ tool. A Xilinx programming cable is needed to connect through the Digital I/O cable connector described in Section **Error! Reference source not found.**.

With the Xilinx cable connected, launch the Xilinx iMPACT™ tool. This is done by double clicking on the Configure Device icon in the Process View window or by opening the standalone iMPACT™ application.

If iMPACT™ is launched from within ISE, a wizard will open a window asking which configuration mode to use. Click the Cancel button in this window.

If iMPACT™ is launched as a standalone application, it may ask if you want to load the most recent project or if you would like to create a new project. Click the Cancel button in this window.

Click on the Boundary-Scan tab. Right click in the window that reads “Right click to Add Device or Initialize JTAG Chain.” Select the Initialize Chain option in the contextual menu that appears. This will cause the JTAG cable to scan the chain and identify the devices.

When the initialization completes, the window should display the JTAG chain with two devices and open a file browser for selecting the file to load to the configuration PROM. Navigate to the .mcs file created following the instructions in Section 8.2.1 and click the open button. Another file browser window will open for selecting the .bit file for programming the FPGA. Navigate to the .bit file created from the project created in Section 0. Programming can proceed by right clicking on the device icons and selecting the desired operations.

8.5 Custom FPGA Development Projects

Custom developments projects should be created as detailed in Section 8.1 with minor modifications. The full UCF (CA_MXXX.ucf or CA_MXXX_qdr for devices with SRAM) should be used in place of the Local Bus example UCF. CA_MXXTop.vhd can be used as a template for a top level file in custom designs.