FreeForm/PCI-104

Reference Design Guide

Connect Tech, Inc.
42 Arrow Road
Guelph, Ontario
Canada, N1K 1S6
Tel: 519-836-1291
800-426-8979
Fax: 519-836-4878
Email: sales@connecttech.com
support@connecttech.com
URL: http://www.connecttech.com

CTIM-00042 Revision 0.07 2013-01-21

Reference Design Release 5.0.0

Limited Lifetime Warranty

Connect Tech Inc. provides a Lifetime Warranty for all Connect Tech Inc. products. Should this product, in Connect Tech Inc.'s opinion, fail to be in good working order during the warranty period, Connect Tech Inc. will, at its option, repair or replace this product at no charge, provided that the product has not been subjected to abuse, misuse, accident, disaster or non Connect Tech Inc. authorized modification or repair.

You may obtain warranty service by delivering this product to an authorized Connect Tech Inc. business partner or to Connect Tech Inc. along with proof of purchase. Product returned to Connect Tech Inc. must be pre-authorized by Connect Tech Inc. with an RMA (Return Material Authorization) number marked on the outside of the package and sent prepaid, insured and packaged for safe shipment.

The Connect Tech Inc. Lifetime Warranty is defined as the serviceable life of the product. This is defined as the period during which all components are available. Should the product prove to be irreparable, Connect Tech Inc. reserves the right to substitute an equivalent product if available or to retract Lifetime Warranty if no replacement is available.

The above warranty is the only warranty authorized by Connect Tech Inc. Under no circumstances will Connect Tech Inc. be liable in any way for any damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use of, or inability to use such product.

Copyright Notice

Trademark Acknowledgment

Customer Support Overview

If you experience difficulties after reading the manual and/or using the product, contact the Connect Tech reseller from which you purchased the product. In most cases the reseller can help you with product installation and difficulties.

In the event that the reseller is unable to resolve your problem, our highly qualified support staff can assist you. Our support section is available 24 hours a day, seven days a week on our website at:

www.connecttech.com/support/support.asp. See the contact information section below for more information on how to contact us directly. Our technical support is always free.

Not listing all possible trademarks or copyright acknowledgments does not constitute a lack of acknowledgment to the rightful owners of the trademarks and copyrights mentioned in this document.

Contact Information

We offer three ways for you to contact us:

Telephone/Facsimile
Technical Support representatives are ready to answer your call Monday through Friday, from 8:30 a.m. to 5:00 p.m. Eastern Standard Time. Our numbers for calls are:

Telephone: 800-426-8979 (North America only)
Telephone: 519-836-1291 (Live assistance available 8:30 a.m. to 5:00 p.m. EST, Monday to Friday)
Facsimile: 519-836-4878 (on-line 24 hours)

Email/Internet
You may contact us through the Internet. Our email and URL addresses are:

sales@connecttech.com
support@connecttech.com
www.connecttech.com

Mail/Courier
You may contact us by letter and our mailing address for correspondence is:
Connect Tech, Inc.
42 Arrow Road
Guelph, Ontario
Canada N1K 1S6

Table of Contents

Revision History

Document Revision	Reference Design Version	Description
0.01	1.0.0	Initial release
0.02	2.0.0	Added support for TEMACs and Rocket I/O
0.03	2.1.0	Added support for Hardware revision C
0.04	3.0.0	Added support for hardware revision D Added support for V5 FXT GTP/GTX control now accessible through host interface TEMAC control now accessible through host interface
0.05	4.0.0	Minor updates
0.06	4.0.3	Updated to ISE 13.4 Add notes about Ethernet MAC Licensing
0.07	5.0.0	Updated Xilinx Projects to ISE 14.3 Updated Visual Studio Projects to VS2012 Updated designs to Support the latest PLX SDK (6.50)

Introduction

The FreeForm/PCI-104 reference design includes FPGA modules and software applications that demonstrate how a host system can communicate with the FreeForm/PCI-104 over the PCI bus, as well as interface with the peripheral hardware (i.e. memory, flash, Ethernet, and serial).

The provided FPGA modules are written in VHDL, and the software applications are written in C. The reader is expected to have a basic understanding of VHDL, FPGA design in general, and software development in C, as well as experience in using the Xilinx ISE development tools.

This guide is primarily focused on the implementation of the FPGA modules, with references to associated software applications and their register interface. While this guide contains important design details, the complete details of the FPGA implementation are beyond the scope of this document. Refer to the VHDL source and C source for complete details.

The reference design contains examples demonstrating:

- Local Bus Slave Transfers
- Local Bus Master Transfers
- GPIO Control
- Programming the SPI Flash
- Interfacing to the Built-in TEMACs
- RS-485 Serial Data Transfers
- Reading/Writing to the Serial EEPROM
- Reading/Writing to DDR2 Memory
- Interfacing to the Rocket I/O

The documentation included in this guide will aid the FPGA designer and/or software developer in creating custom hardware and software applications. In most applications, the host system will not be directly communicating with the FreeForm/PCI-104 hardware peripherals, as is the case with this reference design. It is expected that the user will modify the FPGA code to suit their individual application. The modular format of the reference design will ensure that modifications will be relatively straightforward.

Reference Design Overview

The FPGA reference design logic demonstrates how to interface the FreeForm/PCI-104 (Virtex-5 FPGA) with the PLX PCI 9056 PCI to Local Bus Bridge, as well as the various peripherals.

About the PLX 9CI 9056 Bridge and Local Bus

The PLX 9056 handles all PCI transactions, converting them to simple address/data cycles on a generic local bus. The PLX 9056 hides all PCI handshaking and error conditions from the FPGA, allowing the backend (user) logic to focus on the application.

The generic local bus connected between the FPGA has the following characteristics:

- Separate 30-bit address and 32-bit data buses (referred to as the C-Mode Operation)
- Operation frequency of 50MHz. The FPGA design forwards a 50MHz clock to the bridge
- 8/16/32 bit operations
- Single cycle read/writes (one byte, word, or dword), as well as multi-cycle burst read and writes (either four words at a time, or continuous)
- FPGA can be accessed as a local bus slave - referred to as direct slave (or ds)
- FPGA can be a local bus master, which causes the PLX 9056 to become a PCI bus master - referred to as direct master (or dm)

The format of the local bus, its characteristics, and address space size are controlled through configuration register settings. The configuration registers are loaded after a PCI reset via the FPGA after a PCI, or alternatively loaded via EEPROM.

For details on the configuration settings, refer to the PLX Configuration Settings

FPGA Modules

The follow sections briefly describe the FPGA sub-modules, as well as list applicable source files, reference documentation, registers accessible by host system software and software application examples.

Local Bus Slave Transfers

A PCI slave access will cause the PLX 9056 to request the local bus from the arbitration unit located in the FPGA (plxArb.vhd). Once granted the bus, the PLX 9056 generates single cycle or burst access to FPGA registers and/or memory handled by the slave controller (plx32bitslave.vhd). The registers are implemented in a configurable register bank (regBank_32.vhd). Most registers already have assigned functionality interfacing with other modules; see FPGA Register and Memory Map for more details.

More information:

Source Files	**PLX Local bus interface modules**
Reference Documentation	PLX 9056 Databook
Implementation Details	PLX Local Bus Slave
Registers	N/A
Software Example	DSTest_app

Local Bus Master Transfers

After a PCI reset (which causes a local bus reset), the FPGA becomes the local bus master. The direct master control unit (plx32bitmaster.vhd) arbitrates for the local bus, then loads the PLX 9056 configuration registers, from the FPGA memory (plxcfgrom.vhd). For details on the configuration settings, refer to the PLX Configuration Settings, and the PLX 9056 data book.

To perform PCI bus mastering, the host system application software must set up several registers used by the direct master control unit (plx32bitmaster.vhd). These registers include: the destination local address, the number of 32-bit words to transfer, and the operation. The local bus to PCI bus address association must be set up in the PLX configuration registers. Once the operation is complete, the control unit issues a local bus interrupt, which in turn generates a PCI interrupt. Interrupts can be masked using a combination of FPGA registers or PLX configuration registers.

In this reference design, the bus mastering unit transfers data to/from the FPGA memory located in BAR 2.

More information:

Source Files	**PLX Local bus interface modules**
Reference Documentation	PLX 9056 Databook
Implementation Details	PLX Local BusMaster
Registers	FPGA_DM_CTRL FPGA_DM_ADDR FPGA_DM_CNT FPGA_INTERRUPT_MASK FPGA_INTERRUPT_SOURCE
Software Example	DMTest_app

GPIO Control

Single ended GPIO can be controlled via registers (from the register bank) for simple input/output operations. Each of the 64 I/O pins can be set individually as input or output and can be driven independently.

More information:

Source Files	**N/A**
Registers	FPGA_GPIO_P_OUT FPGA_GPIO_P_TRI FPGA_GPIO_P_IN FPGA_GPIO_N_OUT FPGA_GPIO_N_TRI FPGA_GPIO_N_IN
Software Examples	GPIOScan GPIOTest_app

Programming the SPI Flash

The SPI flash programming module (ctiSPI.vhd) is controlled through registers and memory located in BAR 3. The SPI bus interface is implemented in a PicoBlaze Microcontroller (kcpsm3.vhd). Microcontroller firmware/code (ctiProg.psm) is pre-compiled into a block ram, in read only format (ctiProg.vhd). The registers and block memory are used to send data to/from the micro-controller; which in turn reads/writes to the SPI flash.

More information:

Source Files	**SPI Flash Programming Modules**
Reference Documentation	UG129 PicoBlaze 8-bit Embedded Microcontroller User Guide
Registers	FPGA_SPI_CMD FPGA_SPI_PARAM FPGA_SPI_STATUS FPGA_SPI_RESULT FPGA_INTERRUPT_MASK FPGA_INTERRUPT_SOURCE
Software Example	SPITest_app

Interfacing to the Built-in TEMACs

The Tri-mode Ethernet MAC (TEMAC) interface example demonstrates how to transmit and receive a small data buffer. Each TEMAC port has a separate transmit engine (emacTx.vhd) and receive engine (emacRx.vhd).The MAC address can be set via the EMAC initialization module (emac_init.vhd).

The FIFO based wrapper for the hard TEMAC's (v5_emac_v1_3_example_design.vhd) is based on the COREGEN Virtex-5 Embedded Tri-mode Ethernet MAC Wrapper v1.3. The COREGEN output has been modified for use with this reference design, where the address swap modules have been replaced by transmit and receive engines. Initialization of the TEMAC is performed over the Host Bus Interface, as described in UG194 (Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide).

Access to the host bus interface is also provided via local bus registers, see example software for details.

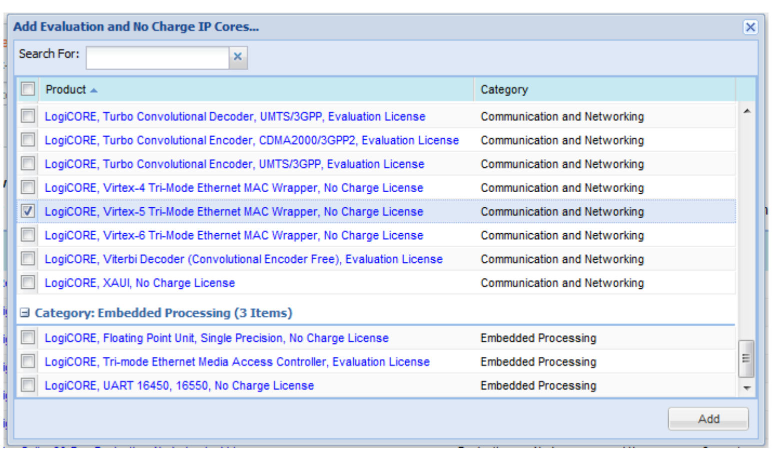
More information:

Source Files	**TEMAC Modules** **TEMAC wrapper modules**
Reference Documentation	DS550 Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC Wrapper UG194 Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide
Registers	FPGA_EMAC_CTRL FPGA_EMAC_STA FPGA_EMAC0TX_BUF FPGA_EMAC0RX_BUF FPGA_EMAC1TX_BUF FPGA_EMAC1RX_BUF FPGA_EMAC_HOST_CTRL FPGA_EMAC_HOST_WDATA FPGA_EMAC_HOST_STATUS FPGA_EMAC_HOST_RDATA
Software Example	miiTest_app

Note about Licensing:

While the TEMAC is hardened IP, ISE still requires the user to acquire a license and generate the wrapper. The license is free and can be obtained from Xilinx's website

Xilinx Part Number: 0451283
Product Name/Description: LogiCORE, Virtex-5 Tri-Mode Ethernet MAC Wrapper, No Charge License



RS-485 Serial Data Transfers

Serial data transmission is controlled by a small state machine (serialSimple.vhd) and a simple set of registers, interfacing to a basic UART (uart_rx.vhd, uart_tx_plus.vhd). Single words can pushed/popped from UART FIFOs; with flags indicating UART status.

More information:

Source Files	**Serial Modules**
Reference Documentation	N/A
Registers	FPGA_RS0_CTRL_TX FPGA_RS0_STATUS_RX FPGA_RS1_CTRL_TX FPGA_RS1_STATUS_RX
Software Example	serialTest_app

Reading/Writing to the Serial EEPROM

The serial EEPROM can be accessed through a set of registers, allowing the host system to perform read/write operations. The EEPROM master logic (eepromMaster.vhd) can easily be modified for use with an embedded processor.

More information:

Source Files	**EEPROM Modules**
Reference Documentation	N/A
Registers	FPGA_EEPROM_CMD_WDATA FPGA_EEPROM_STA_RDATA
Software Example	eeTest_app

Reading/Writing to DDR2 Memory

DDR2 memory block transfers can be initiated via the register set as described in FPGA Register and Memory Map. A state machine and FIFOs (ddr2_interface.vhd) handles the interface to the memory controller (mig20_app.vhd). For a write operation, data is written to the block memory (COREGEN output dp_32_64.vhd), and a transfer is initiated. For a read operation, the transfer is initiated and data can be read from block memory when it is available.

The DDR2 memory controller was generated using the Memory Interface Generator (MIG 2.0) in COREGEN. MIG 2.0 generates a configurable VHDL controller, along with a self testing module. A FIFO based user application interface allows modules, like the self testing module or ddr2_interface.vhd, to initiate read/write memory operations.

Caution: Modifications to the MIG generated VHDL is not recommend. This controller has been carefully designed by Xilinx to work with the Virtex-5 architecture.

More information:

Source Files	**DDR2 Memory Controller Modules**
Reference Documentation	U086 Xilinx Memory Interface Generator (MIG) User Guide XAPP858 High-Performance DDR2 SDRAM Interface in Virtex-5 Devices
Registers	FPGA_DDR2_CTRL FPGA_DDR2_CMD_SZ FPGA_DDR2_ADDR FPGA_DDR2_STATUS
Software Example	DDR2Test_app

Interfacing to the Rocket I/O

The Rocket I/O interface example (mgt_tester.vhd) provides a simple method to transmit and receive a buffer of data, controlled by a register set. For this design to function in hardware, an external loopback must be connected between each transceiver's TX and RX. A transmitter engine (gtp_frame_tx.vhd) loads the Rocket I/O TX FIFO from dual port block memory, while the receive engine (gtp_frame_rx.vhd) empties the Rocket I/O RX FIFO to dual port block memory. Both block memories are accessible from the local bus. There are separate controllers and FIFOs for each TX and RX pair.

The interface example was generated using the Virtex-5 GTP Transceiver Wizard v1.7 in Coregen. PCIe was used as the base protocol to generate the logic. Using the wizard is essential to configure the many attributes/generics.

More information:

Source Files	**Rocket I/O (GTP) modules**
Reference Documentation	UG188 Virtex-5 GTP Transceiver Wizard v1.7 Getting Started Guide UG196 Virtex-5 FPGA RocketIO GTP Transceiver, User Guide
Registers	FPGA_GTP_CTRL FPGA_GTP_STA FPGA_GTP_TXSZ FPGA_GTP_TX0_BUF FPGA_GTP_TX1_BUF FPGA_GTP_TX2_BUF FPGA_GTP_TX2_BUF FPGA_GTP_TX3_BUF FPGA_GTP_RX0_BUF FPGA_GTP_RX1_BUF FPGA_GTP_RX2_BUF FPGA_GTP_RX3_BUF
Software Example	gtpTest_app

Note: Rocket I/O is also referred to as GTP / GTX (gigabit transceiver) or MGT (multi-gigabit transceiver)

Design Organization

The following source and project files (located in the \fpga sub-directory) are used to generate the reference design.

Source Files

The following tables describe the source files contained in the \fpga\source sub-directory

Common Modules & Packages

Directory = \common

File	Description
ctiSim.vhd	Simulation utilities
ctiUtil.vhd	Type declarations
regBank_32	Configurable 32bit register bank
shiftRegXX.vhd	Configurable shift register
Txt_util.vhd	Simulation text printing
Xto1mux_32.vhd	Configurable X down to 1 mux, 32 bits wide

Constraint Files

Directory = \constraints

File	Description
ffpci104_lxt_LVCMOS25_revb.ucf	Constraint file, for 2.5V CMOS GPIO, Hardware Revision B
ffpci104_lxt_LVCMOS25_revb.ucf	Constraint file, for 3.3V CMOS GPIO, Hardware Revision B
ffpci104_lxt_LVCMOS25_revc.ucf	Constraint file, for 2.5V CMOS GPIO, Hardware Revision C
ffpci104_lxt_LVCMOS25_revC.ucf	Constraint file, for 3.3V CMOS GPIO, Hardware Revision C

Top Modules

Directory = \toprefdesign

File	Description
ref_design.vhd	Top file for reference design, links all sub modules
ref_design_lxt_revb_pkg.vhd	Configuration package, sets various VHDL generics to configure the reference design for hardware revision B
ref_design_lxt_revc_pkg.vhd	Configuration package, sets various VHDL generics to configure the reference design for hardware revision C

Simple Testbench

Directory = \tb

File	Description
compile.do	Compilation script
init_tb.vd	Reference design test bench
sim.do	Simulation script
wave.do	Waveform setup script

PLX Local Bus Interface Modules

Directory = \plxControl

File	Description
plx32BitMaster.vhd	32 bit PLX bus master, includes local bus configuration on startup
plx32BitSlave.vhd	32 bit PLX slave, includes memory and register control for two separate BARs

plxArb.vhd	PLX bus arbitration, between PLX device and FPGA master
plxBusMonitor.vhd	Chipscope ILA PLX bus montior
plxCfgRom.vd	Configuration ROM, contains values for local bus registers. Used by Plx32BitMaster.vhd

SPI Flash Programming Modules

Directory = \spiFlash32

File	Description
ctiProg.psm	PicoBlaze firmware, small application that controls the SPI bus and programs the flash
ctiProg.vhd	Compiled from ctiProg.psm, implemented as a ROM
ctiSPI.vhd	Module that interfaces with the PLX 32 bit slave registers, and dual port RAM for flash page storage
kcpsm3.vhd	PicoBlaze microcontroller

Special Configuration Modules

Directory = \v5config

File	Description
V5InternalConfig	Access internal FPGA reconfiguration port to trigger a reset

TEMAC Modules

Directory = \emac

File	Description
emac_init.vhd	Configures the embedded Ethernet MACs via their control bus
emacTx.vhd	Transmit state machine
emacRx.vhd	Receive state machine

TEMAC Wrapper Modules

Directory = \v5coregen\v5_emac_v1_3\example_design

File	Description
v5_emac_v1_3.vhd	TEMAC wrapper, with generics pre-assigned
v5_emac_v1_3_block.vhd	wrapper of above, includes MII registers
v5_emac_v1_3_locallink.vhd	wrapper of above, includes local link fifos
v5_emac_v1_3_example_design.vhd	wrapper of above, includes address swap module OR ICMP ping module
\physical	
mii_if.vhd	MII I/O registers
\client	
address_swap_module_8.vhd	swaps MAC address
\client\fifo	
Eth_fifo_8.vhd	Ethernet local link fifo
rx_client_fifo_8.vhd	Ethernet local link fifo, recv fifo
tx_client_fifo_8.vhd	Ethernet local link fifo, transmit fifo

Serial Modules

Directory = \serial

File	Description
bbfifo_16x8.vhd	fifo 8 bits wide, 16 deep
kcuart_rx.vhd	Receiver uart
kcuart_tx.vhd	Transmit uart
serialSimple.vhd	register controlled serial interface
uart_rx.vhd	Receiver uart
uart_tx_plus.vhd	Transmit uart

EEPROM Modules

Directory = \eeprom

File	Description
eepromMaster.vhd	Microwire eeprom master, control initiated through register interface

DDR2 Memory Controller Modules

Directory = \v5coregen\mig20\user_design\rtl

File	Description
Mig20*.vhd	VHDL files generated by coregen – do not modify
Ddr2_interface.vhd	Example module showing how to use the ddr2 memory controller's application interface.

Rocket I/O Modules

Directory = \rocketio

File	Description
mgt_tester.vhd	Rocket I/O top level, contains 4 rocket I/O
gtp_frame_rx.vhd	Frame receiver state machine and buffer
gtp_frame_tx.vhd	Frame transmitter state machine and buffer

Directory = \v5coregen\pciegtp_wrapper\src

File	Description
pciegtp_wrapper.vhd	wrapper for of two GTP tiles
pciegtp_wrapper_tile.vhd	tile wrapper, contains generic/parameter settings for GTP

Directory = \v5coregen\pciegtx_wrapper\src

File	Description
mgt_usrclk_source_pll.vhd	MGT PLL
pciegtp_wrapper.vhd	wrapper for of two GTX tiles
pciegtp_wrapper_tile.vhd	tile wrapper, contains generic/parameter settings for GTX

Chipscope Modules

Directory = \v5chipscope

File	Description
_chipscope_gen.bat	Batch file to regenerate chipscope cores
*.arg	Arguments to created specified chipscope netlist
icon01.edn	Chipscope ICON, 1 control port
icon4.edn	Chipscope ICON, 4 control ports
ila160_8.edn	Chipscope ILA, 160 bits wide 1024 deep. 8 bit trigger port
ilaPlxBus.edn	Chipscope ILA for PLX bus
vio_async_in64.edn	Chipscope VIO 64 asynchronous inputs
vio_async_in256.edn	Chipscope VIO 256 asynchronous inputs
vio_sync_out32.edn	Chipscope VIO 32 synchronous outputs

Clocking Modules

Directory = \v5clock

File	Description
clkControl.vhd clkControlMem.vhd	Reference design clock control. Receives 100 MHz differential clock Generates 200 MHz, 100 MHz, 50 MHz, 33 MHz (synthesized). Also forwards 50 MHz clock to PLX device, with proper deskew
Clkfwd_mod.vhd	Modified version of clkfwd.vhd, generated by coregen (see \coregen). Deskews and forwards clock using ODDR register.
Plldiv_mod.vhd	Modified version of plldiv.vhd.

Coregen Modules

Directory = \v5coregen

File	Description
blockRAM.xco, .vhd, .ngc	A 32 bit wide x 16 deep single port RAM
Clkfwd.xaw, .vhd	Original file used to create clkfwd_mod.vd
dpRam32_8.xco, .v hd, .ngc	Dual port RAM, Port A = 32 bit wide x 64 deep, Port B = 8 bit wide x 256 deep
dp_a256x8_b256x8.xco, .vhd, .ngc	Dual port RAM, Port A = 8 bit wide x 256 deep, Port B = 8 bit wide x 256 deep
dp_32_64.xco, .vhd, .ngc	Dual port Ram, Port A = 32 bit wide x 64 deep, Port B = 64 bit wide x 32 deep
fifo_42x16.xco	Asynchronous fifo, 42 bit wide x 16 deep
mig20.xco	Original settings used to generate DDR2 memory controller
v5_emac_v1_3.xco	Original settings used to generate Ethernet MAC
gen200Mhz.xaw, .vhd	A DCM with an external differential input 100 MHz clock, then generates 200 MHz (x2), 100 MHz, 50 MHz (/2), and 33 MHz (synthesized).
lckDeskew.xaw, .vhd	Accepts 50 Mhz input clock and deskews it to clock logic. * Applicable to Hardware Revision C only

Project Files

The ISE Foundation project files are available under \projects\ref_design*. Reports from the last successful build have been included for future reference.

File	Description
*.ise	ISE project file
*.syr	Synthesis report
*.bld	Translate report
*.mrp	Map report
*.par	Place and Route report
*.pad	Pad/Pin usage report
*.bgn	Bitgen report
*.bit	Configuration bitstream
*.prm	Promgen report
*.filter	Message filter
*.tcl	TCL script, with functions used to - recreate the project – add files to project and sets process properties - implement the project
*.bat	Various batch files, that execute the TCL scripts

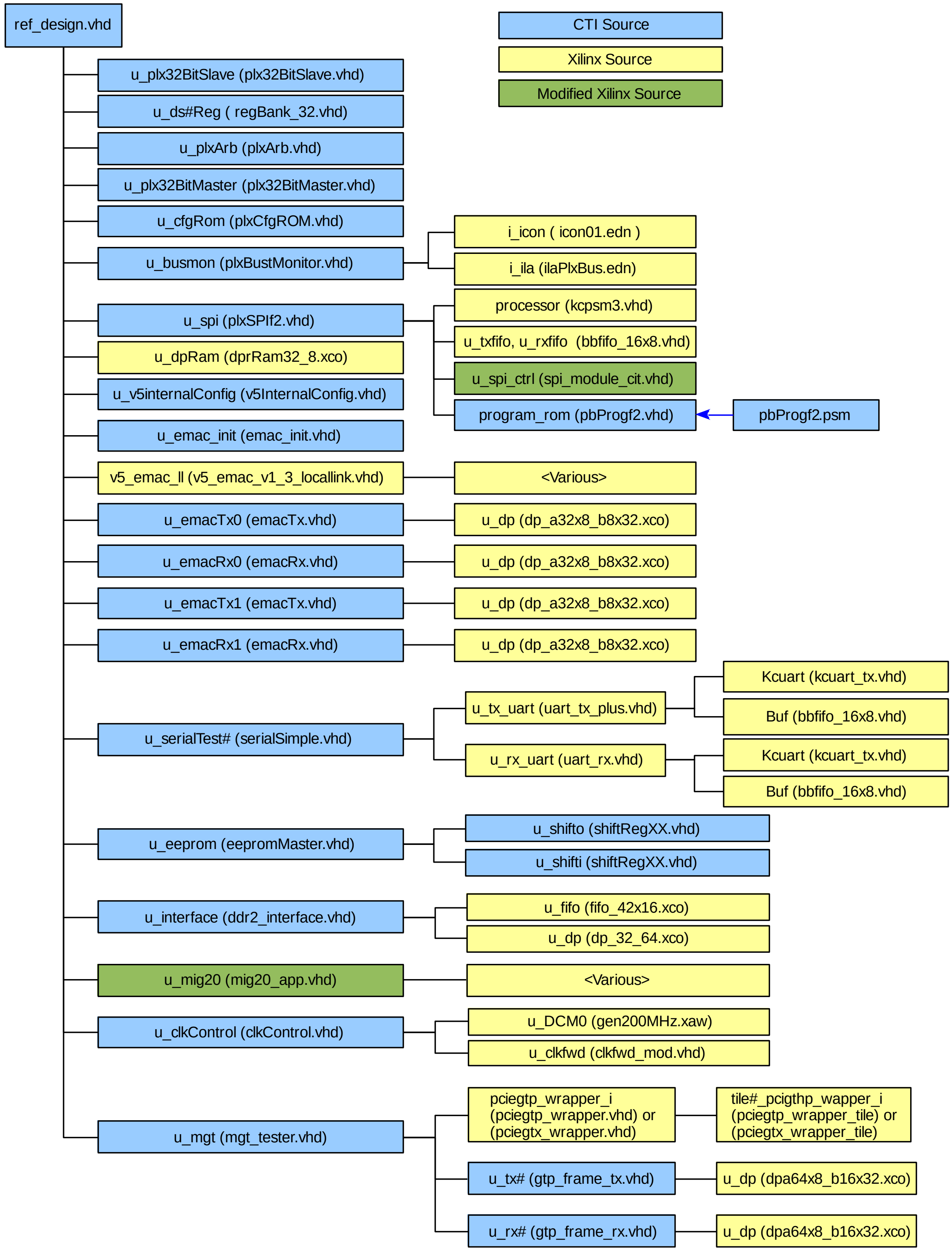
Hardware Revision Management

The reference design supports both hardware revision B and C/D. Separate reference design archives and ISE projects have been developed for each product configuration

Hardware revisions are managed by including the applicable VHDL package and constraint file.

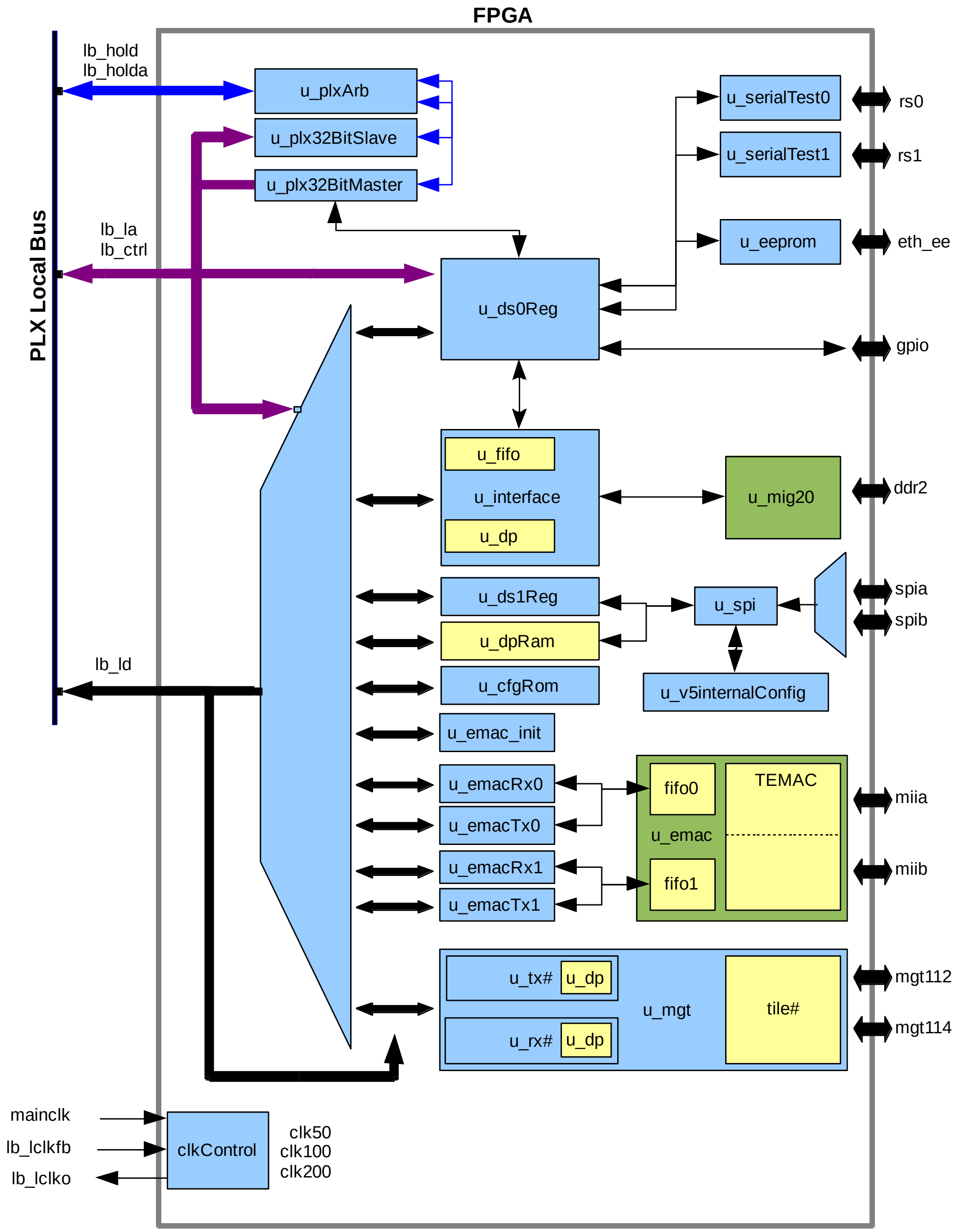
Project	HW Rev	GPIO	VHDL Package	Constraints
ref_design_fcg001rb	B	2.5V	ref_design_fcg001rb_pkg	ffpci104_fcg001rb.ucf
ref_design_fcg002rb	B	3.3V	ref_design_fcg002rb_pkg.vhd	ffpci104_fcg002rb.ucf
ref_design_fcg001rd	C & D	2.5V	ref_design_fcg001rd_pkg.vhd	ffpci104_fcg001rd.ucf
ref_design_fcg002rd	C & D	3.3V	ref_design_fcg002rd_pkg.vhd	ffpci104_fcg002rd.ucf
ref_design_fcg003rd	D	2.5V	ref_design_fcg003rd_pkg.vhd	ffpci104_fcg003rd.ucf
ref_design_fcg004rd	D	3.3V	ref_design_fcg004rd_pkg.vhd	ffpci104_fcg004rd.ucf

Module Hierarchy



FPGA Block Diagram

The following FPGA block diagram summarizes the organization and connectivity of the VHDL modules in the reference design. Each block is labeled with its instance name, as listed in the module hierarchy. For information on each module/design file refer to the previous sections.



Module Implementation Details

This section contains implementation details of several critical modules. These details include:

- VHDL ports
- State diagrams
- Timing diagram excerpts, typically simulation captures

PLX Local Bus Slave

As previously mentioned this module controls single cycle or burst (4 words) access to FPGA registers and memory.

Module Ports (plx32BitSlave.vhd)

The key local bus signals for controlling data transfer are adsn (address strobe), readyn (data ready), blastn (last burst). These form the basic handshake between the FPGA and PLX.

The application data control basically acts as write enables for registers and memory, and also controls the number of wait states the state machine inserts.

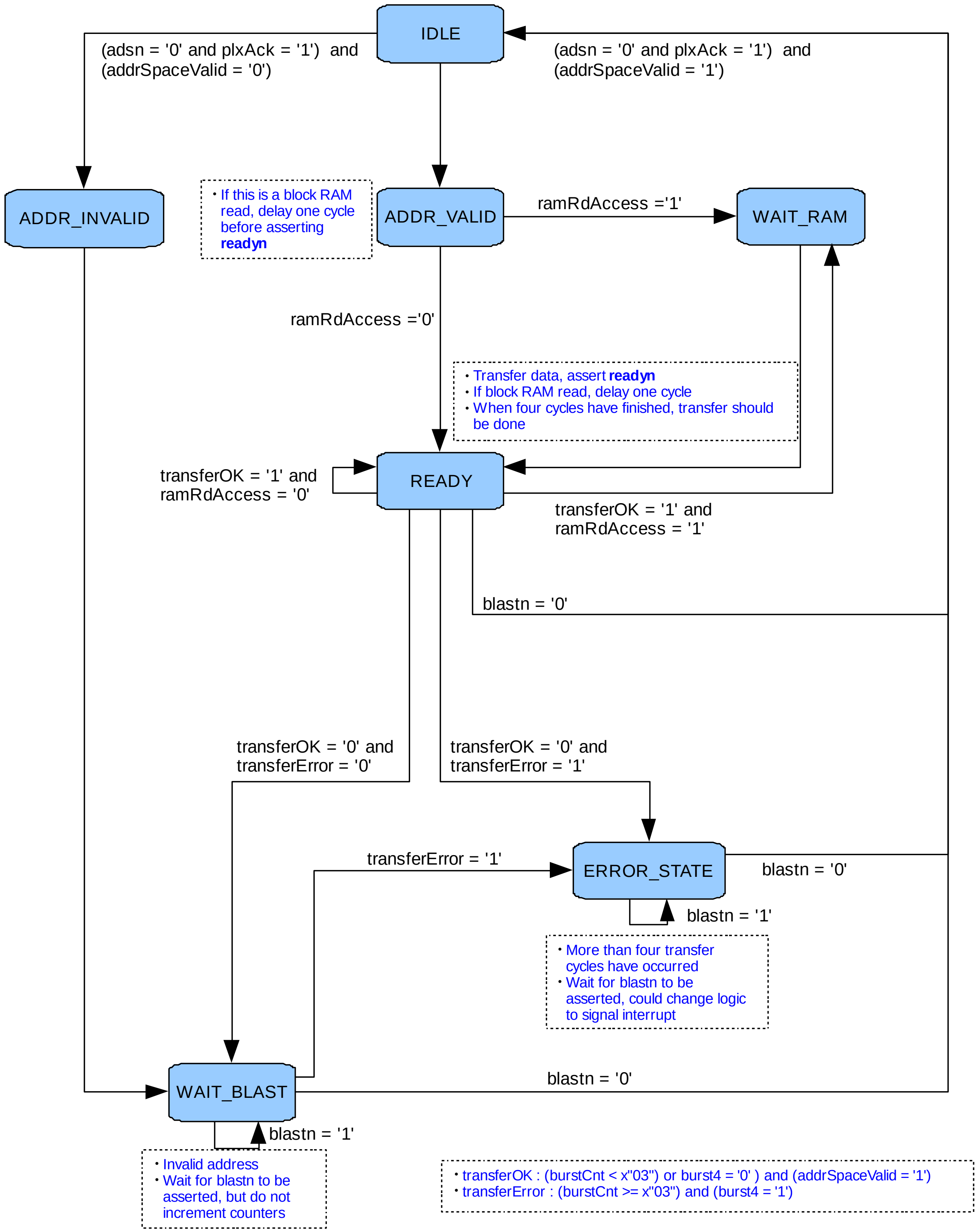
Category	VHDL Port

Local Bus:	`lresetn : in std_logic;`
	`lclk : in std_logic;`
	`ld_dir : out std_logic;`
	`lben : in std_logic_vector(3 downto 0);`
	`adsn : in std_logic;`
	`blastn : in std_logic;`
	`READYn : out std_logic;`
	`lw_rn : in std_logic;`

Arbitration	`plxAck : in std_logic;`

Application Data Control	`addrValid0 : in std_logic;`
	`addrValid1 : in std_logic;`
	`wrByte0 : out std_logic_vector( 3 downto 0);`
	`wrByte1 : out std_logic_vector( 3 downto 0);`
	`ramAccess0 : in std_logic;`
	`ramAccess1 : in std_logic;`
	`burst4 : in std_logic`

State Diagram



PLX Local Bus Master

Module Ports (plx32BitMaster.vhd)

The application interface and data control can be easily connected to another sub-module. In the reference design the interface control is mapped to registers manipulated by the host system application while data is a block ram.

Category	VHDL Port

Local Bus:	`lclk : in std_logic;`
	`la : out std_logic_vector(31 downto 2);`
	`ld_dir : out std_logic;`
	`lben : out std_logic_vector(3 downto 0);`
	`adsn : out std_logic;`
	`blastn : out std_logic;`
	`readyn : in std_logic;`
	`lw_rn : out std_logic;`
	`lresetn : in std_logic;`
	`ccsn : out std_logic;`
	`dmpaf : in std_logic;`

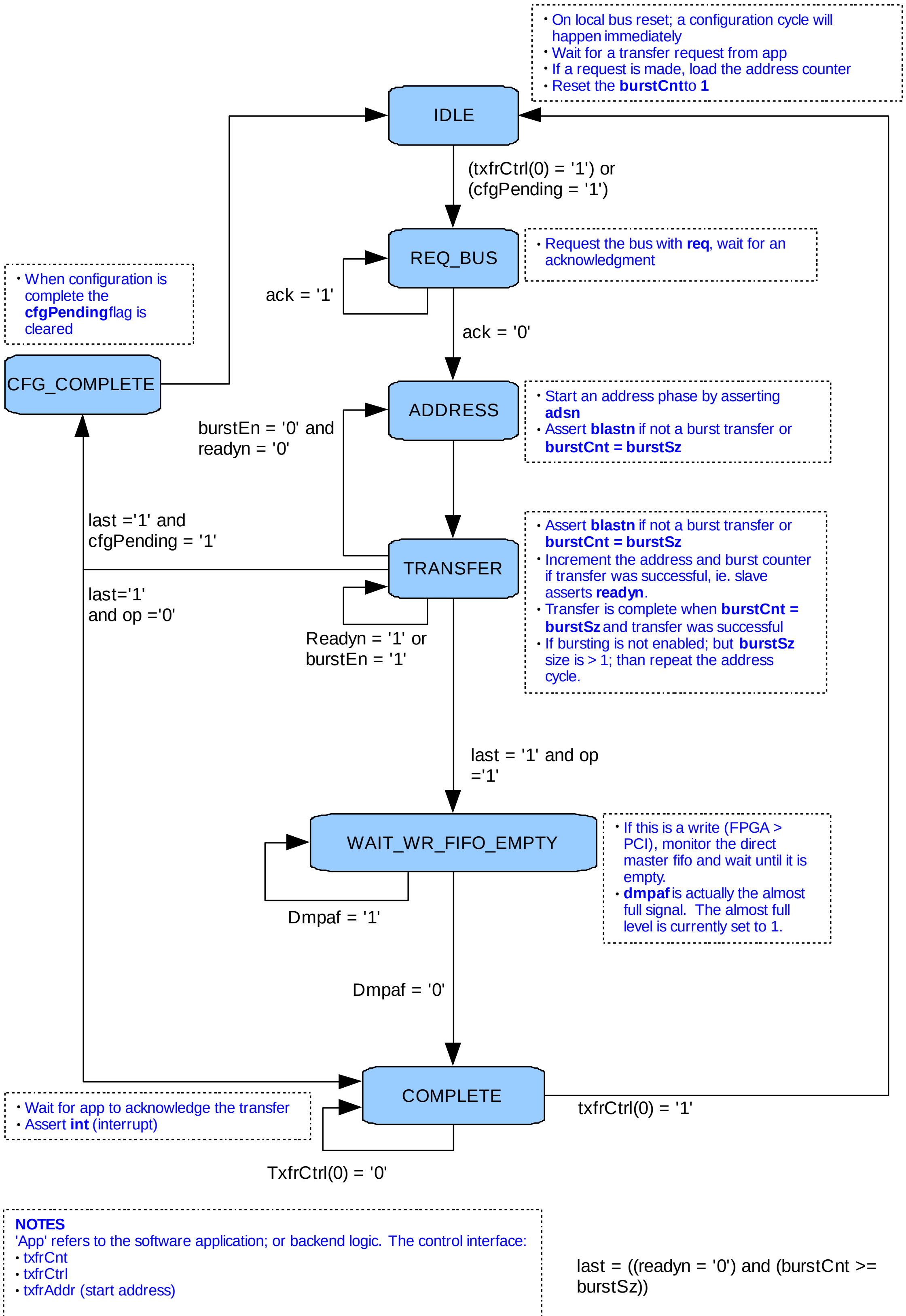
Arbitration	`req : out std_logic;`
	`ack : in std_logic;`
	`backoff : in std_logic;`

Application Interface Registers (or back end logic)	`txfrCtrl : in std_logic_vector(31 downto 0);`
	`txfrAddr : in std_logic_vector(31 downto 0);`
	`txfrCnt : in std_logic_vector(31 downto 0);`
	`int : out std_logic;`

Application Data Control	`ramAddr : out std_logic_vector(3 downto 0);`
	`ramWr : out std_logic_vector( 3 downto 0);`
	`ramEn : out std_logic;`

Configuration Memory Control	`cfgRomPtr : out unsigned(4 downto 0);` `cfgRomDout : in std_logic_vector(47 downto 0);`

State Diagram



Timing Diagrams

Start of direct master transaction

Previously, data had been written to the block memory

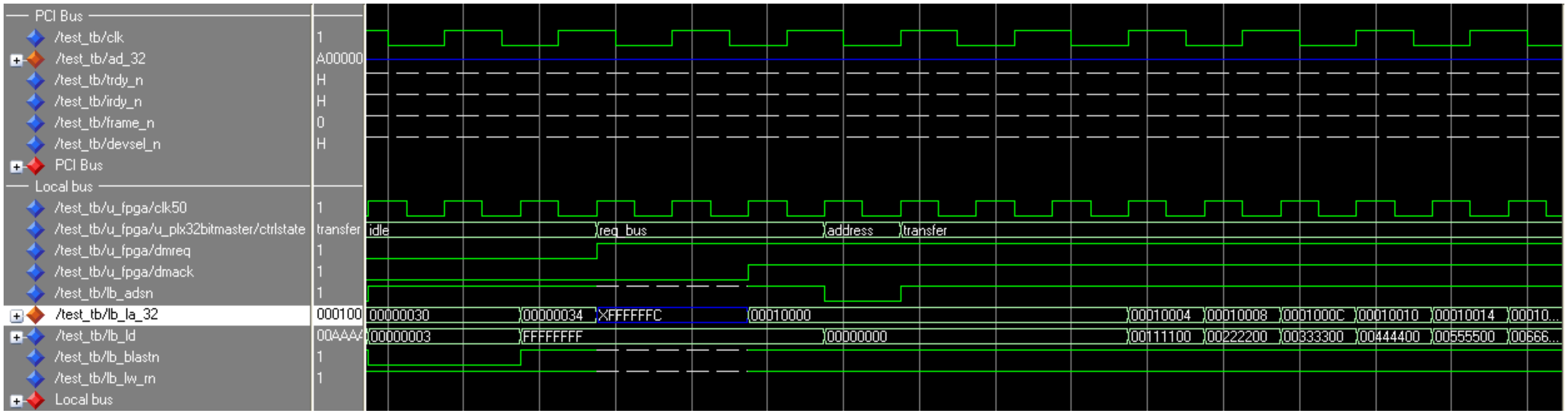
```
00000000, 00111100, 00222200, 00333300, 00444400, 00555500, 00666600, 00777700
00888800, 00999900, 00AAAA00, 00BBBB00, 00CCCC00, 00DDDD00, 00EEEE00, 00FFFF00
```

The FPGA's interface registers have been setup with:

```
txCnt = 0x10
txAddr = 0x00010000
```

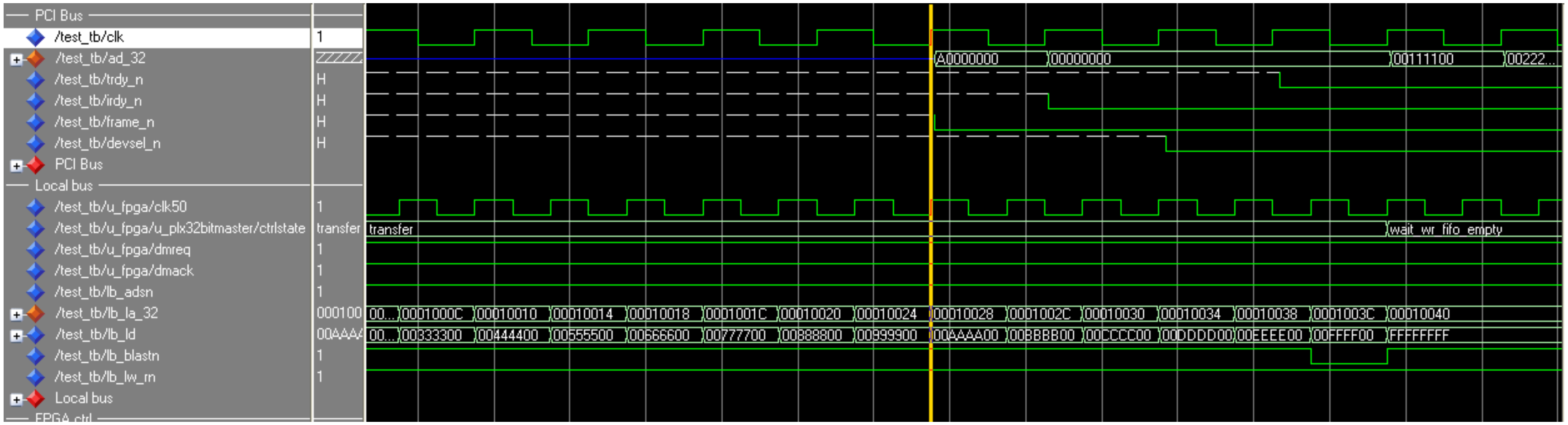
The direct master configuration registers must be setup on the bridge.

```
DMPBAM[31:16] (0x28) = 0xA000
```



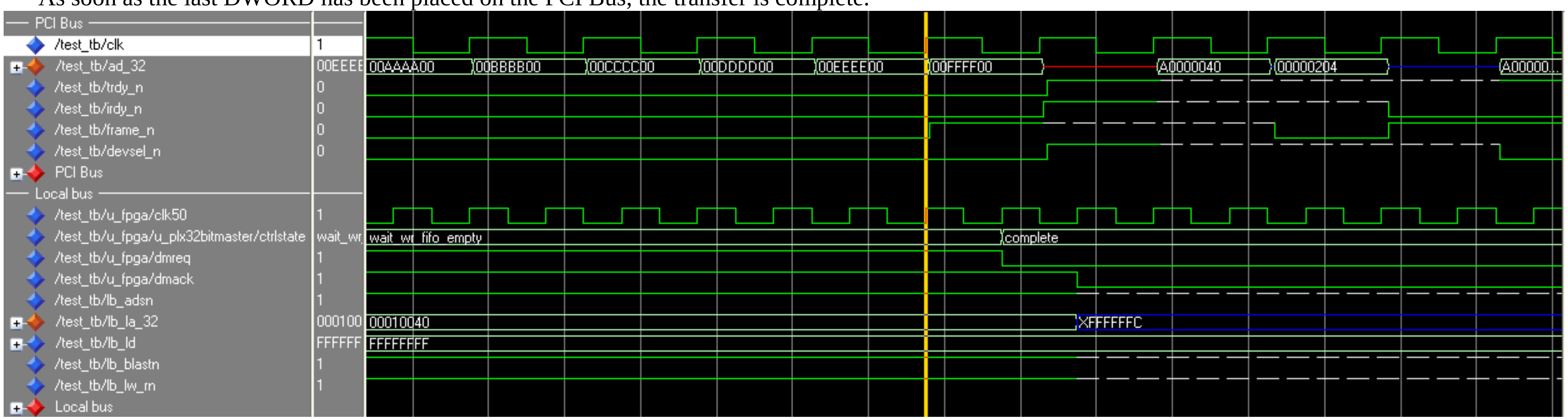
Start of PCI Transfer

Some time later the PCI transfer will begin with the address 0xA0000000 as previously setup. The direct master phase also completes with blastn being asserted during the last data phase.



End of PCI Transfer

As soon as the last DWORD has been placed on the PCI Bus, the transfer is complete.



Local Address Space Decoding

The local address space decoding is setup by the PLX configuration registers and handled by logic in the FPGA. The reference design comes pre-configured with:

Local address space 0 (BAR2) = 512 Bytes, offset 0x10000000
Local address space 1 (BAR3) = 1024 Bytes, offset 0x00000000

Modifying PLX Configuration Registers

Local Address Space 0 is characterized by:

- LAS0RR (PCI:00h, LOC:80h) Direct Slave Local Address Space 0 Range
- LAS0BA (PCI:04h, LOC:84h) Direct Slave Local Address Space 0 Local Base Address (Remap)
- LBRD0 (PCI:18h, LOC:98h) Local Address Space 0/Expansion ROM Bus Region Descriptor

LAS0RR is used to define the size (range) of the address space.

PLX default LAS0RR[31:4] = x"FFF0000" = 1048576 byte
FreeForm/PCI-104 default LAS0RR[31:4] = x"FFFFFF8" = 128 Byte

Example LAS0RR[31:4] = x"FFFFFE0" = 512 Byte

x"FFFFFE00" (adding a 0) is the two's complement of the size

LAS0RR[3:0] must be set to "0000" for memory access

LAS0BA is used to specify the PCI to Local Bus address space translation. Currently:

LAS0BA[31:4] = x"0000000"

These values do not need to change, unless a different local address offset is required.

LBRD0 is used to configure the width and bursting capabilities of address space 0.

LBRD0[1:0] = "11" = 32 Bit access
LBRD0[6] = '1' = Enables the READY# signal, used by the slave state machine
LBRD0[7] = '0' = Sets burst mode to burst size 4
LBRD0[24] = '1' = enables bursting

These values do not need to change.

To change the size and base address, locate the following constants in refdesign.vhd:

c_ds#NumAddrBit	natural	Total number of address bits required to decode space
c_ds#ByteSz	natural	Size of address space in bytes
c_ds#BaseAddr	std_logic_vector(31 downto c_ds0NumAddrBit+2)	Base Address, for purposes of decoding at FPGA level
c_ds#0BaseAddr28	std_logic_vector(31 downto 4)	Base Address, for PLX registers

These constants are used to set the configuration ROM generics (u_cfgRom: plxCfgRom)

Decoding the address

The local address is valid if it matches the set local base address. For example, a 1024byte address space is matched as follows:

```
ds0AddrValid <= '1' when lb_la(31 downto 10) = ds0BaseAddr(31 downto 10) else '0';
```

To decode write enables, compare the lower address bits with the desired offset. The following 'table' decodes the write enables for all modules in address space 0:

```
gtp_ram_en    <= '1' when lb_la(9)          = '1';       else '0';-- 8 to 2, 2^7 =128
ds0RamEn      <= '1' when lb_la(9 downto 8) = "01"       else '0';-- 7 to 2, 2^6 = 64
emac_ram_en   <= '1' when lb_la(9 downto 7) = "001"      else '0';-- 6 to 2, 2^5 = 32
emac0Tx_lb_en <= '1' when lb_la(9 downto 5) = "00100"    else '0';-- 4 to 2, 2^3 =  8
emac0Rx_lb_en <= '1' when lb_la(9 downto 5) = "00101"    else '0';-- 4 to 2, 2^3 =  8
emac1Tx_lb_en <= '1' when lb_la(9 downto 5) = "00110"    else '0';-- 4 to 2, 2^3 =  8
emac1Rx_lb_en <= '1' when lb_la(9 downto 5) = "00111"    else '0';-- 4 to 2, 2^3 =  8
ds0RegEn      <= '1' when lb_la(9 downto 7) = "000"      else '0';-- 6 to 2, 2^5 = 32
```

The read data path is implemented with an output multiplexer, based on the address valid signals and the lower address bits:

```
p_ld_out : process()
begin
    if dmAck='1' then
        if cfgComplete = '1' then
                ld_out <=     dpDoutA;
        else
                ld_out <=     cfgRomDout(31 downto 0);
        end if;
    else
        if  ds1AddrValid = '1' then   --   = '1' lb_la(10)
            if ds1RamEn = '1' then
                ld_out <=     ds1RamDout;
            else
                ld_out <=     ds1RegDout;
            end if;
        else
            if lb_la(9) = '1' then
                ld_out <=     gtp_do;
            else
                if lb_la(8) = '1' then
                    ld_out <=     dpDoutA;
                else
                    if lb_la(7) = '1' then
                        case lb_la(6 downto 5) is
                            when "00" => ld_out <= emac0Tx_lb_do;
                            when "01" => ld_out <= emac0Rx_lb_do;
                            when "10" => ld_out <= emac1Tx_lb_do;
                            when "11" => ld_out <= emac1Rx_lb_do;
                            when others => ld_out <= (others=>'0');
                        end case;
                    else
                        ld_out <=     ds0RegDout;
                    end if;
                end if;
            end if;
        end if;
    end if;
end process p_ld_out;
```

Design Constraints

FPGA design constraints are stored in UCF file, which is used by the ISE software to assign pin locations, define timing, and force logic assignment. Care must be taken when modifying the supplied constraint files. The following sections describe these critical constraints.

GPIO I/O Standard

FCG001 – built with **L12** populated, which connects 2.5V to the GPIO, in banks 11 and 13. This configuration will also support LVDS – constraints must be modified.
>> Use *_LVCMOS25.ucf

FCG002 – built with **L13** populated, which connects 3.3V to the GPIO, in banks 11 and 13.
>> Use *_LVCMOS33.ucf

Local Bus Timing

Timing requirements can be modified as necessary, however the PLX setup and hold times must be met.

Inputs (C-mode)
Tplx_setup = 4.0 ns (max)
Tplx_hold = 1 ns (max)

Outputs (C-mode)
Tplx_co = 7.5 ns

With a 50 MHz local bus clock (20 ns period)

Tfpga_setup = Tperiod – Tplx_co
= 20 ns – 7.5 ns
= 12.5 ns

Tfpga_tco = Tperiod – Tplx_setup
= 20 ns – 4.0 ns
= 16 ns

Currently the following timing is set on the local bus pads:

```
TIMEGRP "LB_IOPAD" OFFSET = IN 8.5 ns BEFORE "mainclkp";
TIMEGRP "LB_IOPAD" OFFSET = OUT 10 ns AFTER "mainclkp";
```

The output datapath is affected by the number of multiplexed registers and memory; care must be taken when adding more of either.

DDR2, Ethernet/MII Interface, Rocket I/O

These I/O standards, placement, and timing constraints have been carefully defined by Xilinx – they should not be modified.

FPGA Pin Assignments

The following table shows the VHDL signal to signal end point mapping. The signal to FPGA pin mapping is listed in the appropriate constraints file (UCF).

Signal Name	Direction	IO Standard	Bank	Endpoint
ddr2_a<0>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<1>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<2>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<3>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<4>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<5>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<6>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<7>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<8>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<9>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<10>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<11>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<12>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_a<13>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_ba<0>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_ba<1>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_ba<2>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_cas_n	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_ck<0>	OUTPUT	DIFF_SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_ck_n<0>	OUTPUT	DIFF_SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_cke<0>	OUTPUT	SSTL18_I_DCI	17	Memory (U12/U13)
ddr2_cs_n<0>	OUTPUT	SSTL18_I_DCI	17	Memory (U12/U13)
ddr2_dm<0>	OUTPUT	SSTL18_I_DCI	17	Memory (U12/U13)
ddr2_dm<1>	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_dm<2>	OUTPUT	SSTL18_I_DCI	17	Memory (U12/U13)
ddr2_dm<3>	OUTPUT	SSTL18_I_DCI	17	Memory (U12/U13)
ddr2_dq<0>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<1>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<2>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<3>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<4>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<5>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<6>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<7>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)

Signal Name	Direction	IO Standard	Bank	Endpoint
ddr2_dq<8>	BIDIR	SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dq<9>	BIDIR	SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dq<10>	BIDIR	SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dq<11>	BIDIR	SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dq<12>	BIDIR	SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dq<13>	BIDIR	SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dq<14>	BIDIR	SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dq<15>	BIDIR	SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dq<16>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<17>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<18>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<19>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<20>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<21>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<22>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<23>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<24>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<25>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<26>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<27>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<28>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<29>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<30>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dq<31>	BIDIR	SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dqs<0>	BIDIR	DIFF_SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dqs<1>	BIDIR	DIFF_SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dqs<2>	BIDIR	DIFF_SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dqs<3>	BIDIR	DIFF_SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dqs_n<0>	BIDIR	DIFF_SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dqs_n<1>	BIDIR	DIFF_SSTL18_II_DCI	18	Memory (U12/U13)
ddr2_dqs_n<2>	BIDIR	DIFF_SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_dqs_n<3>	BIDIR	DIFF_SSTL18_II_DCI	17	Memory (U12/U13)
ddr2_odt<0>	OUTPUT	SSTL18_I_DCI	17	Memory (U12/U13)
ddr2_ras_n	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
ddr2_we_n	OUTPUT	SSTL18_I_DCI	18	Memory (U12/U13)
eth_ee_clk	OUTPUT	LVCMOS33	2	EEPROM (U14)
eth_ee_cs	OUTPUT	LVCMOS33	2	EEPROM (U14)
eth_ee_dido	BIDIR	LVCMOS33	2	EEPROM (U14)
gpio_n<0>	BIDIR	LVCMOS25	11	GPIO Connector (P7)

Signal Name	Direction	IO Standard	Bank	Endpoint
gpio_n<1>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<2>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<3>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<4>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<5>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<6>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<7>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<8>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<9>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<10>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<11>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<12>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<13>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<14>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<15>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<16>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<17>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_n<18>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<19>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<20>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<21>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<22>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<23>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<24>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<25>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<26>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<27>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<28>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<29>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<30>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_n<31>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<0>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<1>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<2>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<3>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<4>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<5>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<6>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<7>	BIDIR	LVCMOS25	11	GPIO Connector (P7)

Signal Name	Direction	IO Standard	Bank	Endpoint
gpio_p<8>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<9>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<10>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<11>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<12>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<13>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<14>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<15>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<16>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<17>	BIDIR	LVCMOS25	11	GPIO Connector (P7)
gpio_p<18>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<19>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<20>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<21>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<22>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<23>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<24>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<25>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<26>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<27>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<28>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<29>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<30>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
gpio_p<31>	BIDIR	LVCMOS25	13	GPIO Connector (P7)
hss_user_io<0>	OUTPUT	LVCMOS33	12	High Speed IO (P4)
hss_user_io<1>	OUTPUT	LVCMOS33	12	High Speed IO (P4)
hss_user_io<2>	OUTPUT	LVCMOS33	12	High Speed IO (P4)
hss_user_io<3>	OUTPUT	LVCMOS33	12	High Speed IO (P4)
lb_adsn	BIDIR	LVCMOS33	15	PLX (U4)
lb_bigendn	OUTPUT	LVCMOS33	15	PLX (U4)
lb_blastn	BIDIR	LVCMOS33	15	PLX (U4)
lb_breqo	INPUT	LVCMOS33	12	PLX (U4)
lb_breqi	UNUSED		15	PLX (U4)
lb_btermn	UNUSED		12	PLX (U4)
lb_ccsn	OUTPUT	LVCMOS33	15	PLX (U4)
lb_dackn<0>	INPUT	LVCMOS33	12	PLX (U4)
lb_dackn<1>	INPUT	LVCMOS33	15	PLX (U4)
lb_dp<0>	UNUSED		15	PLX (U4)
lb_dp<1>	UNUSED		15	PLX (U4)

Signal Name	Direction	IO Standard	Bank	Endpoint
lb_dp<2>	UNUSED		15	PLX (U4)
lb_dp<3>	UNUSED		12	PLX (U4)
lb_dreqn<0>	UNUSED		15	PLX (U4)
lb_dreqn<1>	UNUSED		15	PLX (U4)
lb_eotn	INPUT	LVCMOS33	15	PLX (U4)
lb_la<2>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<3>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<4>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<5>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<6>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<7>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<8>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<9>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<10>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<11>	BIDIR	LVCMOS33	12	PLX (U4)
lb_la<12>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<13>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<14>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<15>	BIDIR	LVCMOS33	12	PLX (U4)
lb_la<16>	BIDIR	LVCMOS33	12	PLX (U4)
lb_la<17>	BIDIR	LVCMOS33	12	PLX (U4)
lb_la<18>	BIDIR	LVCMOS33	12	PLX (U4)
lb_la<19>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<20>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<21>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<22>	BIDIR	LVCMOS33	12	PLX (U4)
lb_la<23>	BIDIR	LVCMOS33	12	PLX (U4)
lb_la<24>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<25>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<26>	BIDIR	LVCMOS33	12	PLX (U4)
lb_la<27>	BIDIR	LVCMOS33	12	PLX (U4)
lb_la<28>	BIDIR	LVCMOS33	15	PLX (U4)
lb_la<29>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<30>	BIDIR	LVCMOS33	16	PLX (U4)
lb_la<31>	BIDIR	LVCMOS33	16	PLX (U4)
lb_lben<0>	BIDIR	LVCMOS33	16	PLX (U4)
lb_lben<1>	BIDIR	LVCMOS33	16	PLX (U4)
lb_lben<2>	BIDIR	LVCMOS33	16	PLX (U4)
lb_lben<3>	BIDIR	LVCMOS33	16	PLX (U4)

Signal Name	Direction	IO Standard	Bank	Endpoint
lb_lclkfb	INPUT	LVDCI_33	4	FPGA (U5)
lb_lclko_loop	OUTPUT	LVCMOS33	15	FPGA (U5)
lb_lclko_plx	OUTPUT	LVCMOS33	15	PLX (U4)
lb_ld<0>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<1>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<2>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<3>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<4>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<5>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<6>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<7>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<8>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<9>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<10>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<11>	BIDIR	LVCMOS33	12	PLX (U4)
lb_ld<12>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<13>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<14>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<15>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<16>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<17>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<18>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<19>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<20>	BIDIR	LVCMOS33	15	PLX (U4)
lb_ld<21>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<22>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<23>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<24>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<25>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<26>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<27>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<28>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<29>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<30>	BIDIR	LVCMOS33	16	PLX (U4)
lb_ld<31>	BIDIR	LVCMOS33	16	PLX (U4)
lb_lhold	INPUT	LVCMOS33	15	PLX (U4)
lb_lholda	OUTPUT	LVCMOS33	15	PLX (U4)
lb_lintin	OUTPUT	LVCMOS33	15	PLX (U4)
lb_linton	INPUT	LVCMOS33	15	PLX (U4)

Signal Name	Direction	IO Standard	Bank	Endpoint
lb_lresetn	INPUT	LVCMOS33	12	PLX (U4)
lb_lserrn	INPUT	LVCMOS33	12	PLX (U4)
lb_lw_rn	BIDIR	LVCMOS33	16	PLX (U4)
lb_pmereon	UNUSED	LVCMOS34	15	PLX (U4)
lb_readyn	BIDIR	LVCMOS33	15	PLX (U4)
lb_useri	UNUSED		15	PLX (U4)
lb_usero	INPUT	LVCMOS33	15	PLX (U4)
lb_waitn	INPUT	LVCMOS33	15	PLX (U4)
mainclkn	INPUT	LVDS_25	3	100 MHz Osc (O1)
mainclkp	INPUT	LVDS_25	3	100 MHz Osc (O1)
mgt112_rx0n	INPUT			High Speed IO (P4)
mgt112_rx0p	INPUT			High Speed IO (P4)
mgt112_rx1n	INPUT			High Speed IO (P4)
mgt112_rx1p	INPUT			High Speed IO (P4)
mgt112_tx0n	OUTPUT			High Speed IO (P4)
mgt112_tx0p	OUTPUT			High Speed IO (P4)
mgt112_tx1n	OUTPUT			High Speed IO (P4)
mgt112_tx1p	OUTPUT			High Speed IO (P4)
mgt114_refclkn	INPUT			100 MHz Osc (O2)
mgt114_refclkp	INPUT			100 MHz Osc (O2)
mgt114_rx0n	INPUT			High Speed IO (P4)
mgt114_rx0p	INPUT			High Speed IO (P4)
mgt114_rx1n	INPUT			High Speed IO (P4)
mgt114_rx1p	INPUT			High Speed IO (P4)
mgt114_tx0n	OUTPUT			High Speed IO (P4)
mgt114_tx0p	OUTPUT			High Speed IO (P4)
mgt114_tx1n	OUTPUT			High Speed IO (P4)
mgt114_tx1p	OUTPUT			High Speed IO (P4)
mii_clk	INPUT	LVDCI_33	4	Ethernet PHY (U17)
mii_mdc	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
mii_mdio	BIDIR	LVDCI_33	4	Ethernet PHY (U17)
mii_resetn	OUTPUT	LVCMOS33	1	Ethernet PHY (U17)
miia_col	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miia_crs	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miia_intn	OUTPUT	LVCMOS33	1	Ethernet PHY (U17)
miia_rxclk	INPUT	LVDCI_33	4	Ethernet PHY (U17)
miia_rxd<0>	BIDIR	LVCMOS33	1	Ethernet PHY (U17)
miia_rxd<1>	BIDIR	LVCMOS33	1	Ethernet PHY (U17)
miia_rxd<2>	INPUT	LVCMOS33	1	Ethernet PHY (U17)

Signal Name	Direction	IO Standard	Bank	Endpoint
miia_rxd<3>	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miia_rxdv	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miia_rxer	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miia_txclk	INPUT	LVDCI_33	4	Ethernet PHY (U17)
miia_txd<0>	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
miia_txd<1>	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
miia_txd<2>	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
miia_txd<3>	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
miia_txen	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
miib_col	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miib_crs	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miib_intn	OUTPUT	LVCMOS33	1	Ethernet PHY (U17)
miib_rxclk	INPUT	LVDCI_33	4	Ethernet PHY (U17)
miib_rxd<0>	BIDIR	LVCMOS33	1	Ethernet PHY (U17)
miib_rxd<1>	BIDIR	LVCMOS33	1	Ethernet PHY (U17)
miib_rxd<2>	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miib_rxd<3>	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miib_rxdv	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miib_rxer	INPUT	LVCMOS33	1	Ethernet PHY (U17)
miib_txclk	INPUT	LVDCI_33	4	Ethernet PHY (U17)
miib_txd<0>	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
miib_txd<1>	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
miib_txd<2>	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
miib_txd<3>	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
miib_txen	OUTPUT	LVDCI_33	4	Ethernet PHY (U17)
plx_hostenn	INPUT	LVCMOS33	12	PLX (U4)
rs0_renn	OUTPUT	LVCMOS33	2	Transceiver (U15)
rs0_rx	INPUT	LVCMOS33	2	Transceiver (U15)
rs0_ten	OUTPUT	LVCMOS33	2	Transceiver (U15)
rs0_tx	OUTPUT	LVCMOS33	2	Transceiver (U15)
rs1_renn	OUTPUT	LVCMOS33	2	Transceiver (U16)
rs1_rx	INPUT	LVCMOS33	2	Transceiver (U16)
rs1_ten	OUTPUT	LVCMOS33	2	Transceiver (U16)
rs1_tx	OUTPUT	LVCMOS33	2	Transceiver (U16)
spia_csn	OUTPUT	LVCMOS33	2	SPI Flash (U10)
spia_mosi	OUTPUT	LVCMOS33	2	SPI Flash (U10)
spib_clk	OUTPUT	LVCMOS33	2	SPI Flash (U11)
spib_csn	OUTPUT	LVCMOS33	2	SPI Flash (U11)
spib_miso	INPUT	LVCMOS33	2	SPI Flash (U11)

Signal Name	Direction	IO Standard	Bank	Endpoint
spib_mosi	OUTPUT	LVCMOS33	2	SPI Flash (U11)
user_led<0>	OUTPUT	LVCMOS33	12	LED (D1)
user_led<1>	OUTPUT	LVCMOS33	12	LED (D2)
user_led<2>	OUTPUT	LVCMOS33	12	LED (D3)
user_led<3>	OUTPUT	LVCMOS33	12	LED (D4)

Build Instructions

This reference design has been developed and tested in hardware with ISE 10.01.03

While migrating to the newest release of ISE has its benefits, successful implementation is not guaranteed.

Note that due to the configurable nature of Xilinx's COREGEN generated interfaces and several of Connect Tech's modules, many warnings will be generated in each of the processes. A filter file has been provided to help block the warnings that can be safely ignored.

Synthesis Properties

Defaults:

- Set Core Search Directories = ../source/v5chipscope | ../source/v5coregen
  - Instructs the synthesis tools to search for the icon01.edn and ilaPlxBus.edn in the v5chipscope directory
  - Instructs the synthesis tools to search for the *.ngc files in the v5coregen sub-directory

Alternatively, the coregen and chipscope files can be moved into the project directory.

Translate Properties

Defaults:

- Set Macro search path = ../source/v5chipscope | ../source/v5coregen
  - Instructs the translate process to search subdirectories for cores

Simulation

A simple test bench is provided (in .\source\tb), allowing simulation of direct slave and direct master bus cycles. This test bench will work in either ISE Simulation or ModelSim; it has been tested with ModelSim XE III 6.2g. A compilation script, simulation script, and waveform setup script are provided.

Full PLX Simulation

More complex simulations may be performed with an encrypted simulation model of the PLX 9056 bridge. The model provided is a completely functional module, emulating the 9056 including all register configurations. To use this simulation model, the following is required:

- Simulation models, acquired from PLX through their sales or technical support channels
- Simulator which supports SWIFT models, such as ModelSim PE or better. ModelSim XE <u>does not</u> currently support SWIFT models.

DDR2 Simulation

Simulation with a memory model supplied by Micron can only be performed in a mixed language simulation environment; such as ModelSim PE. ISE Foundation simulator will not work at this time.

TEMAC & Rocket I/O Simulation

Xilinx provides simulation ‘smart models’ for the Virtex-5 built-in TEMACs and GTP transceivers, however these smart models are also SWIFT models, and therefore have the same requirements as the PLX SWIFT model.

FPGA Memory Map

The FPGA's memory space is divided between BAR2 and BAR3. Each BAR is mapped to a local address space, which has a corresponding local address offset and shown in the table below. Each BAR/local address space may have separate properties and can respond to PCI transactions in different ways as set by the PLX Configuration Registers.

Local Address Space	Local Address Offset	Size	Bar
0	0x00000000	512 bytes	2
1	0x10000000	512 bytes	3

Address Space 0 (Bar 2)

For register usage, refer to sample applications.

Local Address Offset (Hex)	Register name	Access	Bit	Description
00000000	FPGA_INTERRUPT_MASK	R/W	FPGA Interrupt mask	
			0	Direct master state machine
			1	SPI programmer
0x004	FPGA_INTERRUPT_SOURCE	R	FPGA interrupt source	
			0	Direct master state machine
			1	SPI programmer
0x008	FPGA_EMAC_CTRL	R/W	31	EMAC1 RX Packet Processed
			23	EMAC1 Send Packet
			21:16	EMAC1 TX Packet Size
			15	EMAC0 RX Packet Processed
			7	EMAC0 Send Packet
			5:0	EMAC0 TX Packet Size
0x00C	FPGA_EMAC_STA	R	31	EMAC1 RX packet done
			29:24	EMAC1 RX Packet Size
			23	EMAC1 TX Packet done
			15	EMAC0 RX packet done
			13:8	EMAC0 RX Packet Size
			7	EMAC0 TX Packet done
0x010	FPGA_GPIO_P_OUT	W	GPIO registered output, for gpiop	
			0	GPIO_P(0) output value
			1	GPIO_P(1) output value
			…	
			31	GPIO_P(31) output value
0x014	FPGA_GPIO_P_TRI	W	GPIO registered direction, where 1=output, 0=input	
			0	GPIO_P(0) direction
			1	GPIO_P(1) direction
			…	

Local Address Offset (Hex)	Register name	Access	Bit	Description
			31	GPIO_P(31) direction
0x018	FPGA_GPIO_P_IN	R	GPIO registered input, for gpiop	
			0	GPIO_P(0) input value
			1	GPIO_P(1) input value
			…	
			31	GPIO_P(31) input value
0x01C	FPGA_GPIO_N_OUT	W	GPIO registered output, for gpion	
			0	GPIO_N(0) output value
			1	GPIO_N(1) output value
			…	
			31	GPIO_N(31) output value
0x020	FPGA_GPIO_N_TRI	W	GPIO registered direction, where 1=output, 0=input	
			0	GPIO_N(0) direction
			1	GPIO_N(1) direction
			…	
			31	GPIO_N(31) direction
0x024	FPGA_GPIO_N_IN	R	GPIO registered input, for gpion	
			0	GPIO_N(0) input value
			1	GPIO_N(1) input value
			…	
			31	GPIO_N(31) input value
0x028	FPGA_GTP_TXSZ	RW	31:24	GTP3 TX Size
			23:16	GTP2 TX Size
			15:8	GTP1 TX Size
			7:0	GTP0 TX Size
0x02C	FPGA_USER_LED	W	3:0	USER_LED(3:0)
0x030	FPGA_DM_CTRL	W	Direct master control	
			0	start operation, when complete must be cleared before another operation can begin
			1	Write = 1, Read = 0
0x034	FPGA_DM_ADDR	W	Direct master destination address	
			31:0	Local bus destination address. Must match what is programmed into PLX configuration register DMLBAM. See PLX Configuration Settings
0x038	FPGA_DM_CNT	W	31:0	Number of DWORDs to transfer
0x03C	FPGA_REV	R	FPGA reference design revision	
			7:0	Build
			15:8	Minor revision
			23:16	Major revision
0x040	FPGA_DDR2_CTRL	R/W	DDR2 transfer control	

Local Address Offset (Hex)	Register name	Access	Bit	Description
			0	Indicates new transfer when 1 *
			1	Acknowledge transfer complete *
0x044	FPGA_DDR2_CMD_SZ	R/W	DDR2 controller command and transfer size	
			2:0	Transfer cmd, see source for more details
			31:16	Transfer size
0x048	FPGA_DDR2_ADDR	R/W	31:0	DDR2 linear address
0x04C	FPGA_DDR2_STATUS	R	DDR2 controller status	
			0	Transfer complete
			1	Transfer engine busy
			2	Phy calibration has been completed
			3	transfer operation fifo is full
0x050	FPGA_EEPROM_CMD_WDATA	R/W	EEPROM command, count, address, and write data	
			0	Start operation *
			2:0	Command – see source code for more details
			7:4	Read Count
			15:8	Address
			31:16	Write data
0x054	FPGA_EEPROM_STA_RDATA	R	EEPROM status and read data	
			0	Operation Done
			31:16	Read data
0x058	FPGA_RS0_CTRL_TX	R/W	Serial port 0, control and transmit data byte	
			7:0	TX data
			16	Push TX data *
			17	Pop RX data *
			31:24	baud rate setting, see source for details
0x05C	FPGA_RS0_STATUS_RX	R	Serial port 0, status and receive data byte	
			7:0	RX data
			16	TX data present
			17	TX FIFO full
			18	TX FIFO half full
			20	RX data present
			21	RX FIFO full
			22	RX FIFO half full
0x060	FPGA_RS1_CTRL_TX	R/W	Serial port 1, control and transmit data byte	
			7:0	TX data
			16	Push TX data *
			17	Pop RX data *
			31:24	baud rate setting
0x064	FPGA_RS1_STATUS_RX	R/W	Serial port 1, status and receive data byte	

Local Address Offset (Hex)	Register name	Access	Bit	Description
			7:0	RX data
			16	TX data present
			17	TX FIFO full
			18	TX FIFO half full
			20	RX data present
			21	RX FIFO full
			22	RX FIFO half full
0x068	FPGA_EMAC_HOST_CTRL	R	TEMAC host bus control	
			9:0	Address
			17:16	OP Code
			18	Select EMAC 1, when 1
			19	Select MDIO interface
			24	Request
0x06C	FPGA_EMAC_HOST_WDATA	R/W	31:0	TEMAC host bus write data
0x070	FPGA_EMAC_HOST_STATUS	R	TEMAC host bus status	
			1	EMAC initialization done
			0	MDIO interface busy
0x074	FPGA_EMAC_HOST_RDATA	R	31:0	TEMAC host bus read data
0x078	FPGA_GTP_CTRL	R/W	31	GTP TX RST
			30	GTP RX RST
			28	GTP3 RX OK
			27	GTP3 TX start
			26:24	GTP3 Loopback
			20	GTP2 RX OK
			19	GTP2 TX start
			18:16	GTP2 Loopback
			12	GTP1 RX OK
			11	GTP1 TX start
			10:8	GTP1 Loopback
			4	GTP0 RX OK
			3	GTP0 TX start
			2:0	GTP0 Loopback
0x07C	FPGA_GTP_STA	R/W	31:27	GTP3 RX size
			26	GTP3 RX done
			25	GTP3 TX done
			24	GTP3 PLL ok
			23:19	GTP2 RX size
			18	GTP2 RX done
			17	GTP2 TX done
			16	GTP2 PLL ok
			15:11	GTP1 RX size
			10	GTP1 RX done
			9	GTP1 TX done
			8	GTP1 PLL ok
			7:3	GTP0 RX size
			2	GTP0 RX done

Local Address Offset (Hex)	Register name	Access	Bit	Description
			1	GTP0 TX done
			0	GTP0 PLL ok
0x080 – 0x9FF	FPGA_EMAC0TX_BUF	R/W		EMAC0 Transmit Buffer
0x0A0 – 0xBFF	FPGA_EMAC0RX_BUF	R/W		EMAC0 Receive Buffer
0x0C0 – 0xDFF	FPGA_EMAC1TX_BUF	R/W		EMAC1 Transmit Buffer
0x0E0 – 0x0FF	FPGA_EMAC1RX_BUF	R/W		EMAC1 Receive Buffer
0x100 – 0x1FC	FPGA_MEMORY0_LOC	R/W		Block memory, for DDR2 transfers
0x200 – 0x23F	FPGA_GTP_TX0_BUF	R/W		GTP0 Transmit buffer
0x240 – 0x27F	FPGA_GTP_TX1_BUF	R/W		GTP1 Transmit buffer
0x280 – 0x2BF	FPGA_GTP_TX2_BUF	R/W		GTP2 Transmit buffer
0x2C0 – 0x2FF	FPGA_GTP_TX3_BUF	R/W		GTP3 Transmit buffer
0x300 – 0x3CF	FPGA_GTP_RX0_BUF	R/W		GTP0 Receive buffer
0x340 – 0x37F	FPGA_GTP_RX1_BUF	R/W		GTP1 Receive buffer
0x380 – 0x3BF	FPGA_GTP_RX2_BUF	R/W		GTP2 Receive buffer
0x3C0 – 0x3FF	FPGA_GTP_RX3_BUF	R/W		GTP3 Receive buffer

* = must be cleared before set again

Local Address Space 1 (Bar 3)

For more details, refer to example application source code.

Local Address (Hex)	Contents	Access	Description	
0x000	FPGA_SPI_CMD	RW	31:0	SPI controller command register, once command is written operation begins
0x004	FPGA_SPI_PARAM	RW	SPI command parameters, specific to command	
			7:0	Param0
			15:8	Param1
			23:16	Param2
			31:24	Param3
0x008	FPGA_SPI_STATUS	R	0	Operation complete
0x00C	FPGA_SPI_RESULT	R	SPI command results	
			7:0	Result0
			15:8	Result1
			23:16	Result 2
			31:24	Result 3
0x010	FPGA_SPI_SEL	RW	0	0 = FPGA configuration flash 1 = Embedded code storage flash
0x014	FPGA_BASE_ADDR_0	RW	31:0	Sets space 0 base address; must also change corresponding register in PLX brdige
0x018	FPGA_HSS	RW	11:8	HSS User IO tristate
			3:0	HSS User IO output
0x01C	FPGA_HSS_IN	R	3:0	HSS User IO input
0x020 – 0x0FC	<NON-ADDRESSABLE>	N/A	N/A	
0x100 – 0x1FC	FPGA_MEMORY1	RW	256 Bytes of for flash page storage	

About the PLX 9056 Local Bus Bridge

C Mode Operation

The PLX 9056 C-Mode operation, while full of features, only requires a sub-set of the signals to implement data transactions. A typical sequence:

- Master asserts adsn (for one clock cycle) and asserts lw_rn as necessary
- Slave asserts readyn when it has stored data or has data to transmit
- Master asserts blastn to indicate last data phase

Refer to the PLX 9056 data book for more information:
- Section 4 C and J modes operation
- Section 5 C and J modes functional description (includes timing diagrams)
- Section 12.5 C Bus Mode Pinout

The following table lists all signals in the C mode local bus and the associated VHDL port. It also identifies signals related to unused features:

(a) Big endian selection, data parity, and locked transaction
(b) Bus interruption
(c) Built-in DMA engine

Signal	Description	Bits	Dir	VHDL Port	Unused Features
adsn	Address Strobe	1	inout	lb_adsn : inout std_logic;	
bigendn	Big Endian Select	1	out	lb_bigendn : out std_logic;	(a)
blastn	Burst Last	1	inout	lb_blastn : inout std_logic;	
breqi	Bus Request In	1	out	lb_breqi : out std_logic;	(b)
breqo	Bus Request Out	1	inout	lb_breqo : inout std_logic;	(b)
btermn	Burst Terminate	1	inout	lb_btermn : inout std_logic;	(b)
ccsn	Configuration Register Select	1	out	lb_ccsn : out std_logic;	
dack0n	DMA Channel 0 Demand Mode Acknowledge	1	in	lb_dack0n : in std_logic;	(c)
dack1n	DMA Channel 1 Demand Mode Acknowledge	1	in	lb_dack1n : in std_logic;	(c)
eotn	DMPAF = Direct Master Programmable Almost Full EOTn = End of Transfer for Current DMA Channel	1	inout	lb_eotn : inout std_logic;	
dp	Data Parity	4	inout	lb_dp : inout std_logic_vector(3 downto 0);	(a)
dreq0n	DMA Channel 0 Demand Mode Request	1	out	lb_dreq0n : out std_logic;	(c)
dreq1n	DMA Channel 1 Demand Mode Request	1	out	lb_dreq1n : out std_logic;	(c)
dt_rn	Data Transmit/Receive				
la	Local Address Bus	30	inout	lb_la : inout std_logic_vector(31 downto 2);	
lben	Local Byte Enables	4	inout	lb_lben : inout std_logic_vector(3 downto 0);	
lclk	Local Processor Clock	1	out	lb_lclk_plx : out std_logic;	

Signal	Description	Bits	Dir	VHDL Port	Unused Features
ld	Local Data Bus	32	inout	lb_ld : inout std_logic_vector(31 downto 0);	
lhold	Local Hold Request	1	in	lb_lhold : in std_logic;	
lholda	Local Hold Acknowledge	1	out	lb_lholda : out std_logic;	
lintin	Local Interrupt Input	1	out	lb_lintin : out std_logic;	
linton	Local Interrupt Output	1	in	lb_linton : in std_logic;	(a)
lresetn	Local Bus Reset	1	inout	lb_lresetn : inout std_logic;	
lserrn	Local System Error Interrupt Output	1	in	lb_lserrn : in std_logic;	
lw_rn	Local Write/Read	1	inout	lb_lw_rn : inout std_logic;	
pmereqn	PME Request	1	out	lb_pmereqn : out std_logic;	
readyn	Ready I/O	1	inout	lb_readyn : inout std_logic;	
useri	User Input or LLOCKin Local Lock Input	1	out	lb_useri : out std_logic;	(a)
usero	User Output or LLOCKon Local Lock Output	1	in	lb_usero : in std_logic;	(a)
waitn	Wait I/O	1	inout	lb_waitn : inout std_logic;	(c)
	Pin Count	**95**			

.

Configuration Register Settings

The following PLX 9056 configuration register settings are stored in the FPGA ROM memory (plxcfgrom.vhd); and loaded into the PLX after a local bus reset. Note that the byte enables ensure that only certain bytes are written, default values are accepted otherwise.

For more details, see PLX 9056 data book, section 11.

Local Address	Byte Enables	Value	Register	Settings
0x098	0x0	0x41000043	LBRD0[31:0]	32 bit; enable readyn input; burst 4; burst Enable
0x004	0xD	0x00000106	PCISR / PCICR;	Serr enable
0x02C	0x0	0x999912C4	PCISID[15:0] / PCISVID[15:0]	Sub-vendor ID = 0x12C4 Sub-device ID = 0x9999
0x03C	0x0	0x00000100	PCIMLR[7:0] / PCIMGR[7:0] / PCIIPR[7:0] / PCIILR[7:0]	
0x080	0x0	0xFFFFFE00	LAS0RR[31:0]	Memory space Decode 512 byte
0x084	0x0	0x00000001	LAS0BA[31:0]	Enable address space 0
0x088	0x0	0x00240000	MARBR[31:0]	Breqi enabled. Release bus when direct slave fifos, full or empty
0x08C	0x2	0x00002000	LMISC2[5:0] / PROT_AREA[6:0] / LMISC1[7:0] / BIGEND[7:0]	All little endian Enable lserr interrupt
0x090	0x0	0x00000000	EROMRR[31:0]	Not used
0x094	0x0	0x00000000	EROMBA[31:0]	Not used
0x09C	0x3	0xFFFF0000	DMRR[31:0]	Decode a 64Kbyte space. Note that this is the minimum space.
0x0A0	0x3	0x00010000	DMLBAM[31:0]	Set local address for direct master transactions
0x12C	0x3	0x00000000	DMLBAI[31:0]	Not used
0x0A8	0x0	0x00000001	DMPBAM[31:0]	Memory access enable; no I/O access continuous prefetch on read; write fifo almost full flag set to 0.
0x0Ac	0x0	0x00000000	DMCFGA[31:0]	Not used
0x0E8	0x0	0x00000900	INTCSR	PCI interrupt enable, local int in enable, local int out enable
0x170	0x0	0xFFFFFE00	LAS1RR[31:0]	Memory space Decode 512 bytes
0x174	0x0	0x10000001	LAS1BA[31:0]	Enable address space Base address 0x10000000
0x178	0x0	0x00000143	LBRD1[31:0]	32 Bit; enable ready input; use Burst 4 (not continuous burst); enable bursting
0x08C	0xD	0x00000500	LMISC2[5:0] / PROT_AREA[6:0] / LMISC1[7:0] / BIGEND[7:0]	Configuration complete

Software Examples

Windows/Linux

PLX Software Development Kit (SDK)

PLX provides a software development kit (SDK) to aid in the creation of applications using the PLX 9056 bridge. The SDK provides a generic driver for Windows 2000/XP and Linux (2.4 / 2.6). A common API is also included; which encapsulates functions such as:

- Configuration register read/write
- Block read/block write to local address space (i.e. memory/registers in the FPGA)
- Physical memory allocation, for bus mastering or DMA purposes
- Interrupt handling
- EEPROM read/write by address

The SDK is available for download from:

http://www.plxtech.com/products/sdk/

In order to download the SDK, you will need to register with PLX.
All projects have been compiled with version 6.5, release 2011-09-30

Windows Development

Developing applications under Windows is relatively straightforward. The PLX SDK installer will install and register all the drivers, then copy the API to the install directory. An environment variable PLX_SDK should be created and point to the base SDK directory.

When building the application, simply include the headers files in:

$PLX_SDK\Include
$PLX_SDK\Windows\PlxAPi\Release\PlxApi.lib

For more details refer to the PLX SDK User Manual.

All application include a Visual Studio 2010 project *ApplicationName*_Cur.sln, .vcxproj

Linux Development

Several build scripts are provided, which will properly compile the drivers and the API. Follow the included directions specified for the target kernel. Make files are provided to build the sample applications.

Example Applications

The following table outlines the applications developed using the SDK, located in the \software\windows sub-directory. Each application has a Microsoft Visual Studio .Net solution file / project for build in Windows and a makefile for building the application under Linux.

Location	Application	Files	Description
\AllTest	AllTest		Executes all programs listed below: - DDR2Test - DMTest - DSTest - eeTest - GPIOTest - miiTest - serialTest
\DDR2Test	DDR2Test_app	DDR2Test_app.c DDR2Test.c DDR2Test.h	Writes to all various locations in all memory banks, the reads back the data to validate memory.
\DMTest	DMTest_app	DMTest_app.c DMTest.c DMTest.h	Sets up a direct master transfer, writing data from FPGA internal memory to host system memory
\DSTest	DSTest_app	DSTest_app.c DSTest.c DSTest.h	Performs reads/writes to internal FPGA memory and registers, then toggle LEDs
\eeTest	eeTest_app	eeTest_app.c eeTest.c eeTest.h	Writes, then reads to EEPROM attached to FPGA, while validating contents.
\GPIOScan	GPIOScan	Gpioscan.c	Scans the GPIO registers, displaying the state of the pins
\GPIOTest	GPIOTest_app	GPIOTest_app.c GPIOTest.c GPIOTest.h	Requires a loopback cable; writes to pins then reads state of connected pin. Direction is reversed, and tests executes again
\miiTest	miiTest_app	miiTest_app.c miiTest.c miiTest.h	Reads the internal registers of the FPGA’s TEMAC, reads the PHY registers using the MDIO interface, and then controls the PHYs LED pins, toggling them on and off. Transmits data out one port, and receives through the other port. Requires an Ethernet cable to be connected between ports.
\gtpTest	gtpTest_app	gtpTest_app.c gtpTest.c gtpTest.h	Sends data out each transmitter and verifies received data through loopback. Also toggles side band user IO.
\Peek	Peek	Peek.c	Reads data from the FPGA given the BAR, offset, data type, and count
\Poke	Poke	Poke.c	Writes data to the FPGA given the BAR, offset, data type, and value
\serialTest	serialTest_app	serialTest_app.c serialTest.c serialTest.h	Requires a loopback between ports. Sends data from one port to the other. Data is validated, then the bit rate is toggled and transmission is repeated.
\SPIProgram	SPITest_app	SPITest_app.c SPIProgram.c SPIProgram.h SPIInterface.c SPIInterface.h	Executes common SPI flash commands: read ID, read Signature, erase sector, read page, write page. Validates data is written.

In general, the applications call functions specifically created for the FreeForm/PCI-104. These functions are essentially wrappers for API function calls. See PlxInit.c / PlxInit.h for more details.

QNX

Included in the reference design distribution are several utilities for QNX6: peek6 and poke6. Source and makefile are located in the \software sub-directory.

peek6

Usage:

```
peek6 [-0 | -1 | -2 | -3] [-n count] [-A format] [-s size] [address]
defaults: -n 1 -A x -s 4 0
options:
-0 .. -3   Base Address Register Select\n"
-A format  Display data in following format:
           x  hex
           o octal
           d decimal
           a ascii
-n count   Number of values to read
-s size    Size of values (1, 2 or 4 bytes)
```

Example: to read from the LED register

```
peek6 -2 -n 1 -A x -s 4 2C
```

poke6

Usage:

```
poke6 [-0 | -1 | -2 | -3] [-n count] [address] [data]
defaults:   -0 -n 1 0 0
options:
-0 .. -3     Base Address Register Select
-n count     Number of bytes to write
```

Example**:** to write to the value of 0xA1B2C3B4 to the LED register

```
poke6 -2 -n 4 2C B4 C3 B2 A1
```

References

Device	Location	
PLX 9056	Data Book and SDK	PLX PCI 9056 Registration may be required.
Virtex-5 FPGA	Main Web Page	Virtex-5
	User Guides	UG190 Virtex-5 User Guide UG191 Virtex-5 Configuration User Guide
	FPGA Development Environment	ISE WebPACK