Lattice
Semiconductor Corporation

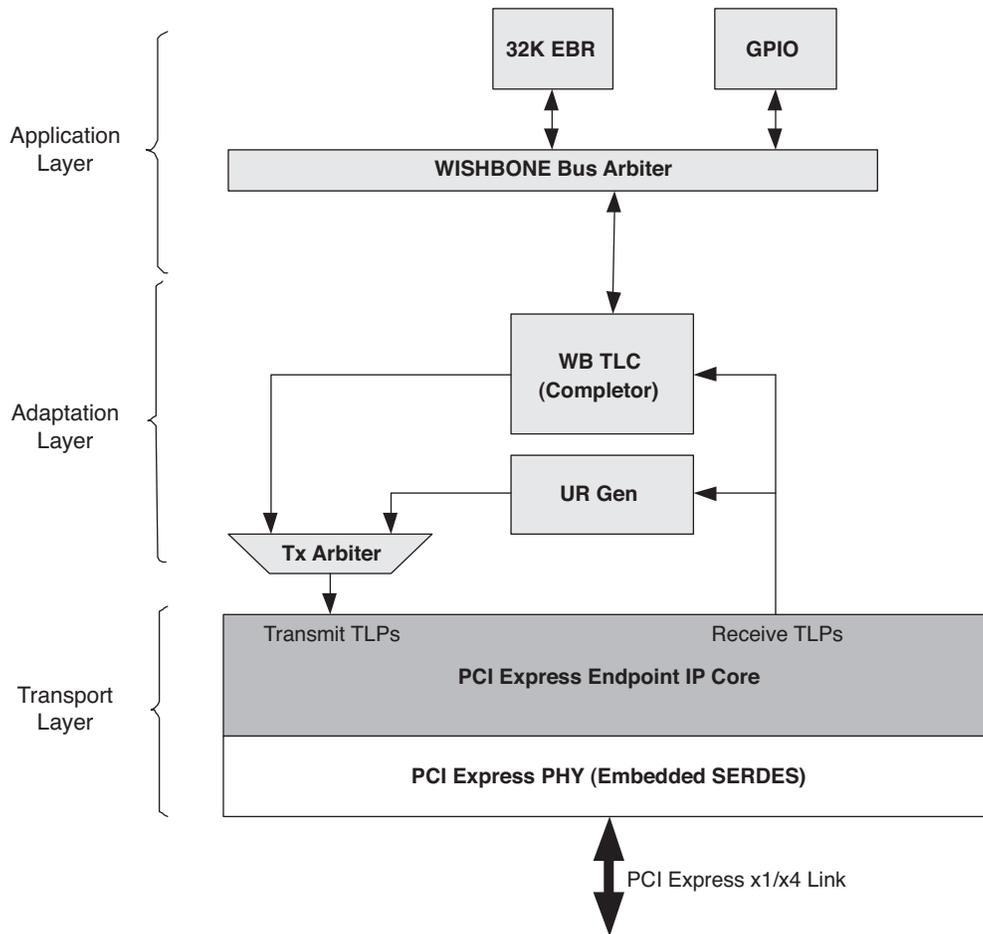# PCI Express Basic Demo Verilog Source Code

User's Guide

Introduction

This user's guide provides details of the Verilog code used for the Lattice PCI Express Basic Demo. A block diagram of the entire design is provided followed by a description for each module in the design. Instructions for building the demo design using Lattice Diamond™ design software are provided as well as a review of the preference file used for the demo.

Top Level

Figure 1 provides a top-level diagram of the demo Verilog design.

Figure 1. Top-Level Block Diagram

The block diagram is separated into three distinct functions. The Transport Layer is used to move TLPs to and from the PCI Express link. This set of features is supported by the Lattice PCI Express Endpoint IP core and embedded SERDES. Moving up the design stack, the next layer is the Adaptation Layer. This layer is responsible for converting PCI Express TLPs into useful data for the Application Layer. The Adaptation Layer uses several soft-IP Verilog modules to extract received TLP contents and repackage transmit data into TLPs for transmission. The final level is the Application Layer. The Application Layer provides the demo capability that is utilized by the demo software.

Basic Project Directory Structure

Figure 2 provides the directory structure for the Diamond project. The LatticeECP3™, LatticeECP2M™ and LatticeSC™ implementations use most of the same files. The only differences are the top level modules and the IP cores. Specific files for each architecture family are located in their respective directories.

Figure 2. Directory Structure

```
<device>_Basic_<link width>/
        Source/
                wb_tlc/
                gpio/
                ur_gen/
                32kebr/
                wb_arb/
                <architecture>/
        ipExpressGenCore/
                <architecture>/
                        pcie[x4,x4d1]/
        Implementation/
                <device>_PCIeBasic_<link width>/
        Testbench/
        Simulation/
                <architecture>/
                        PCIeBasic.<simulator>
```

Modules

This section will discuss the details of each of the modules that make up the demo design. Each module listed will be followed by the filename of the Verilog file which includes this module. Verilog files can be found in the "Source" directory of the demo package.

Top Level – top_basic.v

The top-level module contains all of the blocks found in Figure 1. There is a different top-level file for the LatticeECP3, LatticeECP2M and LatticeSC devices. The LatticeECP3 and LatticeECP2M use a PIPE interface to the embedded SERDES. The LatticeSC encapsulates the SERDES inside of the PCI Express Endpoint IP core. The LatticeSC also uses the system bus to control the multi-channel alignment registers inside of the embedded PCS/SERDES. Other than these two differences, the top level files are the same.

PCI Express Endpoint IP Core – pcie_bb.v

This module is the Lattice PCI Express Endpoint IP core. This module is an encrypted IP core which uses a Verilog black box model for synthesis and an obfuscated Verilog simulation model. IPexpress™ is used to create this module. In the ipExpressGenCore/<arch>/pcie[x4,x4d1] directory the file pcie.lpc is located. This file can be used to load the settings into IPexpress to recreate or modify the PCI Express module.

For the LatticeECP3 and LatticeECP2M an additional file, pcie_top.v, is provided in the Source/[ecp2m, ecp3] directory. This file provides a Verilog wrapper for the LatticeECP3 and LatticeECP2M PCI Express cores and PIPE interfaces.

x4 and x4d1

In the ipExpressGenCore/<arch>/ directory there is either a pciex4 or pciex4d1 directory. The pciex4 directory is used if the demo supports a native x4 PCI Express link. The pciex4d1 is used if the demo supports a x1 link. The pciex4d1 directory contains the Lattice x4 PCI Express Endpoint IP core downgraded to a x1. This core is used instead of the native Lattice x1 PCI Express Endpoint IP core to maintain the 64-bit interface required by the demo code. The native Lattice x1 PCI Express Endpoint IP core uses a 16-bit interface, which can not be used with the demo code.

Embedded SERDES – pcs_pipe_bb.v (LatticeECP3 and LatticeECP2M Only)

The LatticeECP3 and LatticeECP2M use a PIPE interface to the embedded SERDES. This module provides the PIPE interface and the SERDES interface. This file is created when creating the PCI Express Endpoint IP core for the LatticeECP3 and LatticeECP2M.

Tx Arbiter – ip_tx_arbiter.v

This module allows several clients to send TLPs to the PCI Express Endpoint IP core. Using a round-robin scheduler, the tx arbiter waits for the current client to finish sending TLPs before switching to the next client.

Rx Credit Processing – ip_rx_crpr.v

This module monitors receive TLPs coming from the PCI Express Endpoint IP core and terminates credits for Posted and Non-Posted TLPs which are not handled by the wb_tlc module. The number of credits used by each TLP is calculated and using the PCI Express Endpoint IP core ports these credits are terminated. Terminated credits are stored until an UpdateFC DLLP can be sent by the PCI Express Endpoint IP core informing the far end that the credits have been freed.

Unsupported Request Generation – ur_gen.v

All Non-Posted TLPs and memory accesses to unsupported memory must provide a completion. This module will generate unsupported request completions informing the far-end device of these types of memory transactions.

This module receives input from both the PCI Express Endpoint IP core rx_us_req signal as well as TLPs from the receiver. Whenever the rx_us_req signal indicates an unsupported request this module will send an unsupported request completion to the PCI Express Endpoint IP core.

This demo does not support I/O requests. Whenever an I/O request is made this module will send an unsupported request completion to the PCI Express Endpoint IP core.

This demo implements two BARs (BAR0 and BAR1). If a memory request is made to an address other than that serviced by BAR0 or BAR1 an unsupported request completion will be sent to the PCI Express Endpoint IP core.

WISHBONE Transaction Layer Completer (WB_TLC) – wb_tlc.v

The WISHBONE Transaction Layer Completer (WB_TLC) is used as a control plane interface for the endpoint. This module accepts received TLPs from the PCI Express Endpoint IP core. If they are memory transactions to either BAR0 or BAR1, the memory transaction is used by the completer. Otherwise, the TLP is dropped and is handled by the unsupported request module.

The WB_TLC is responsible for adapting 1DW TLP memory requests into WISHBONE transactions. 1 DW TLPs are only supported since this is a low throughput control plane interface which reduces logic. There are several modules underneath the WB_TLC top level.

TLP Decoder – wb_tlc_dec.v
The WB_TLC TLP decoder is responsible for decoding the type of TLP that enters the WB_TLC. The WB_TLC is only capable of supporting MRd and MWr TLPs that are accessing BAR0 or BAR1. All other TLPs are dropped and presumably handled by other modules in the design. After leaving the decoder, MRd and MWr TLPs write into a FIFO.

Memory Request TLP FIFO – wb_tlc_req_fifo.v
The request FIFO is used to store accepted TLPs until they can be converted into WISHBONE transactions on the WISHBONE bus. The request FIFO provides two clock domains (write and read), however this demo design connects both ports to the single 125MHz clock domain.

TLPs are read from the FIFO when the FIFO is no longer empty under control of the WISHBONE interface module. This module terminates write credits when the TLPs are pulled from this FIFO. Read credits are terminated when the completion is sent.

Credit Processor - wb_tlc_cr.v
This module converts credits terminated in the WISHBONE clock domain to the PCI Express domain.

WB_TLC WISHBONE Interface – wb_intf.v
The WISHBONE interface of the WB_TLC is responsible for reading TLPs from the request FIFO and creating WISHBONE transactions. When the request FIFO is not empty, the WISHBONE interface will read the TLP from the FIFO until the end of the TLP. As the TLP is read, the address and data will be converted into a WISHBONE transaction. The address is adjusted to use the least significant 18 bits.

For read transactions the transaction ID is passed out of the module to be used by the completion generation module.

WB_TLC Completion Generation – wb_tlc_cpld.v
The WB_TLC completion generation module is responsible for accepting data from a read request on the WISH-BONE bus and creating a CplD TLP. As data is returned from a read on the WISHBONE bus the CplD TLP is filled with this data. The CplD also uses the transaction ID and length from the wb_intf module to fill the remaining fields in the CplD. Once the CplD TLP is created it is stored in the CplD FIFO.

WB_TLC Completion FIFO – wb_tlc_cpld_fifo.v
The completion FIFO stores the CplD TLPs from the completion generation module until the TLP can be sent to the PCI Express Endpoint IP core.

WISHBONE Arbiter - wb_arb.v

The WISHBONE arbiter is responsible for arbitrating between the WB_TLC and the Unsupported Request module for access to the WISHBONE bus. It is also responsible for the slave select based on an address decode from the master selected. To account for the demo applications features, the arbiter does not support all masters transacting with all slaves. The WB_TLC, however, can request data from all slaves on the WISHBONE bus.

GPIO – wbs_gpio.v

The General Purpose Input Output (GPIO) module is responsible for several housekeeping functions. It provides an ID register used by the software to identify the feature set and version of the design. It also provides access to control the 16-segment LED on the board. There is a section dedicated to interrupt control logic as well as other maintenance type functions. Table 1 is a memory map for the GPIO module.

Table 1. GPIO Module Memory Map

Address	Bits	Description
0x0	[0:31]	ID register
0x4	[0:31]	Scratch pad
0x8	[0:15]	DIP switch value
	[16:31]	16 segment LED
0xc		Generic down counter control
	[0]	Counter run
	[1]	Counter reload
0x10	[0:31]	Counter value
0x14	[0:31]	Counter reload value
0x18		SGDMA Control and Status
	[0:4]	DMA Request (per channel)
	[5:9]	DMA Ack (per channel)
0x1c	[0:31]	DMA Write Counter - The number of clock cycles from the DMA request to the DMA ack of channel 0
0x20	[0:31]	DMA Read Counter - The number of clock cycles from the DMA request to the DMA ack of channel 1

Table 1. GPIO Module Memory Map (Continued)

Address	Bits	Description
0x24	[0:15]	Root complex Non-Posted Buffer size
	[16:31]	Root complex Posted Buffer size
0x28	[0:31]	EBR Filter value used for triangle manipulation
0x2c		Not used
0x30	[0]	ColorBar reset
0x100	[0:31]	Interrupt Controller ID
0x104	[0]	Current status of interrupt
	[1]	Test mode
	[2]	Interrupt enable to pass interrupt onto the PCI Express Endpoint IP core
	[3:7]	Not used
	[8:15]	Interrupt Test value 0
	[16:23]	Interrupt Test value 1
	[24:31]	Not used
0x108	[0:15]	If in normal mode: [0:4] dma_ack [5] down counter = 0 If in test mode: [0:7] Test value 0 [8:15] Test value 1
0x10c	[0:31]	Interrupt mask for all sources

32K EBR – wbs_32kebr.v

The 32K EBR is used to store data on the WISHBONE bus. The WISHBONE slave is 64 bits wide and supports burst operations on the bus.

System Bus – sysbus.v (LatticeSC Only)

The system bus is an embedded block of the LatticeSC that provides access to the memory map for the PCS/SERDES block. This is required in the LatticeSC implementation of the PCI Express for multi-lane links.

uML Controller – uml.v (LatticeSC Only)

The uML controls the PCS Multi Channel aligner registers of the LatticeSC PCS to account for the lane width determined during LTSSM training. This module receives information from the PCI Express Endpoint IP core and then writes registers in the PCS to control the multi channel aligner. More information on the uML can be found in the PCI Express core user's guide.

LED Status – led_status.v

This module provides the control of the LEDs on the demo board for the LTSSM states.

PLL – pll.v (LatticeSC Only)

A FPGA PLL is required to take the 100MHz PCI Express clock and create a 250MHz clock for the SERDES reference clock. The SERDES then uses a 10x multiplier to create the 2.5GHz clock for the PCI Express link.

Building the Design in Diamond

This section describes how to open an existing project or create a new project and build a bitstream for the demo design.

Opening an Existing Project

You can open an existing Diamond project .ldf file in Diamond. To open an existing project:

1. In Diamond, choose **File > Open > Project**.

2. In the Open Project dialog box, choose the project .ldf file.

3. Click **Open**.

To build the project, double-click **Bitstream File** in the Process Pane.

Note: This project enables the use of the IP Hardware Timer. If the user does not have a license for any of the IP used in this design, the IP Hardware Timer will hold the FPGA in global reset after approximately four hours of operation. Once a valid license is installed and the project rebuilt the Hardware Timer will no longer be used.

If design changes are made to this design, place and route options may need to be adjusted to run multiple placement iterations.

For instructions on how to import an ispLEVER project into Diamond, refer to the Lattice Diamond User Guide.

Creating a New Project for the LatticeECP3 and LatticeECP2M

To create a new project the user must select a device, import all of the HDL files, create the .lpf file, set the search paths, and copy all of the autoconfig files to the project directory. Below is a list of steps that need to be completed to create a new project.

1. From the Diamond main window, choose **File > New > Project**.

    The New Project dialog box of the Project Wizard opens. Click **Next**.

2. In the Project Name dialog box, do the following:

    Under Project, specify the name and directory location for the new project. The default implementation name is the same as the new project name. You can change it if you wish. Implementations are comprised of source and strategy files. Multiple implementations are possible for one project.

    Click **Next**.

3. In the Add Source dialog box, click **Add Source**. Add all source file described in the previous section.

    Additionally, add the **pmi_def.v** file which provides the module definition for PMI modules used by the design.

    Click **Next**.

4. On the Select Device page, select the family and device for the evaluation board you are using. For the LatticeECP2M PCI Express Evaluation Board, select the **LatticeECP2M** family, **LFE2M50E** device, **-6** speed grade, and a **FPBGA672** package. For the LatticeECP3 PCI Express Evaluation Board, select the **LatticeECP3** family, **LFE3-95EA** device, **-7** speed grade, and a **FPBGA672** package.

5. Choose **Project > Active Strategy > Translate Design Settings**. Verify that Macro Search Path is set to the directory path **..\..\ipExpressGenCore\[epc3,ecp2m]\[pciex4,pciex4d1]** for Windows. Your path would have forward slashes for Linux.

6. Next, the autoconfig text files need to be copied to the project implementation directory at **ecp3-95_PCIeBasic_SBx4\Implementation\ecp3-95_PCIeBasic_SBx4\impl1**. The autoconfig text files contain lines to program the hard macros in the design. The file **pcs_pipe_8b_X4.txt** file should be copied into the project directory. If the design is targeting a x1 link (using a PCIe x4 downgraded x1 core) then the unused channels will need to be disabled by editing the **pcs_pipe_8b_x4.txt** file. See the PCI Express User's Guide for more information on modifying the file.

7. The project requires a logical preference file (.lpf). Start with the file from the kit at **ecp3-95_PCIeBasic_SBx4\Implementation\ecp3-95_PCIeBasic_SBx4**. This file may need modification to match your design.

8. Choose **Project > Active Strategy > Place and Route Design Settings**.

9. In the Strategy dialog box, set the number of Placement Iterations to **10**. This will allow place and route to try 10 different placements while attempting to satisfy all of the timing constraints.

10. The design is now ready to be built. Double-click on the Bitstream File in the Process pane of Diamond to create the bitstream.

Note: This project enables the use of the IP Hardware Timer. If the user does not have a license for any of the IP used in this design, the IP Hardware Timer will hold the FPGA in global reset after approximately four hours of operation. Once a valid license is installed and the project rebuilt the Hardware Timer will no longer be used.

Simulation

Provided in the simulation directory is a sample simulation for the PCI Express Basic demo. This simple testbench uses a PCI Express driver to toggle the LEDs and write and read from the EBR of the PCI Express Basic design.

The simulation is supported for both the ModelSim® and Active-HDL® simulators. For each simulator, simply set the simulator's working directory to the location of the tb.do script and then execute the tb.do script.

Conclusion

This user's guide provides a description for the PCI Express Basic Demo design. With this guide, a user can rebuild the bitstream used for the demo and begin to modify the design to achieve their design goals.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
            +1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
August 2008	01.0	Initial release.
August 2008	01.1	Updated Directory Structure figure.
		Added Simulation text section.
December 2009	01.2	Added LatticeECP3 support.
December 2010	01.3	Updated for Lattice Diamond design software support.