



EZDMA2 IP for Xilinx Hard IP

Getting Started

Version 1.4.3 April 2010
Copyright © PLDA 1996-2010

EZDMA2 IP

Getting Started

Documentation Change History

Date	Version Number	Changes
April 2010	1.4.3	<ul style="list-style-type: none"> • No change
December 2009	1.4.2	<ul style="list-style-type: none"> • Updated specification versions. • Updated required versions of EDA tools.
August 2009	1.4.0	<ul style="list-style-type: none"> • Added support for Virtex 6 and Spartan 6.
June 2009	1.3.2	<ul style="list-style-type: none"> • Updated required versions of EDA tools and Endpoint Block Plus. • Added note about a known issue with Endpoint Block Plus.
March 2009	1.2.2	<ul style="list-style-type: none"> • Updated required version of ModelSim. • Updated Configuration Wizard and Reference Design.
October 2008	1.2.1	<ul style="list-style-type: none"> • Updated required version of NCSim to 6.1.1 S002. • Corrected minor bug in description of Simulating with ModelSim.
July 2008	1.2.0	<ul style="list-style-type: none"> • Change specification standard to <i>Virtex-5 LogiCORE Endpoint Block Plus v1.8 for PCI Express Designs User Guide UG341</i>. • Updated EZ DMA IP Wizard to reflect changes specified by Endpoint Block Plus. • Updated required versions of ISE.
May 2008	1.1.2	<ul style="list-style-type: none"> • Updated required versions of ISE and Impact
December 2007	1.1	<ul style="list-style-type: none"> • No change
November 2007	1.0.2	<ul style="list-style-type: none"> • Updated Wizard description and EDA Tools requirements
June 2007	1.0	<ul style="list-style-type: none"> • First release

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by PLDA SA. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by PLDA in good faith. This document is provided "as is" with no warranties whatsoever, including any warranty of merchantability, non infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample.

This document is intended only to assist the reader in the use of the product. PLDA shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product. Nor shall PLDA be liable for infringement of proprietary rights relating to use of information in this document. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Product Status

The information in this document is final content pertaining to the PLDA EZDMA2 IP Core.

Web Address

<http://www.plda.com>

Table of Contents

List of Tables	5
List of Figures	6
Preface	7
About this Document	7
Additional Reading	7
Feedback and Contact Information	8
Chapter 1 Before you Start...	9
1.1 System Requirements	9
1.2 EDA Tools Requirements	9
1.3 Package Features	9
1.3.1 Xilinx Source Package	10
1.3.2 Xilinx Full Package	10
1.3.3 Xilinx Board / Eval Package	11
1.4 Installing the Package	12
1.4.1 Windows 2000/XP/Vista	12
1.4.2 UNIX / Linux platform	12
1.5 Exploring the Installed Files	12
1.5.1 Xilinx Source Package	12
1.5.2 Xilinx Full and Board/Eval Packages	13
1.6 Creating a Parameterized Instance of the Core	14
1.6.1 Launching the EZDMA2 IP Wizard	14
1.6.2 Customizing the Core with the EZDMA2 IP Wizard	15
1.6.2.1 <i>Define Settings for <wrapper name> instance: Options tab</i>	16
1.6.3 Customizing Xilinx Hard IP Core for Use with EZDMA2 IP	17
1.7 Simulating your Design	18
1.7.1 Simulating with the PLDA PCIe BFM	18
1.7.1.1 <i>ModelSim</i>	18
1.7.1.2 <i>NCSim and VCS</i>	18
1.7.1.3 <i>Aldec Active-HDL and Riviera-Pro</i>	19
1.7.2 Simulating with ModelSim	19
1.7.3 Additional Xilinx Considerations	19

Chapter 2	Core Synthesis	20
2.1	Creating a New Project with ISE	20
2.2	Place and Route with ISE	20
2.3	Generating a Programming File with ISE	20
Chapter 3	Reference Design	21
3.1	Introduction	21
3.2	Architecture of the Reference Design	21
3.2.1	Top-Level Blocks	21
3.2.2	Reference Design Files and File Structure	22
3.2.3	Registers	24
3.3	Simulation of the Reference Design	25
3.3.1	Overview	25
3.3.2	Script Behavior and Arguments	26
3.3.3	Launching the Simulation	27
3.3.3.1	<i>1-Bit Simulation</i>	27
3.3.3.2	<i>ModelSim</i>	27
3.3.3.3	<i>NCSim</i>	27
3.3.4	Annotated Explanation of the Simulation	27
3.3.4.1	<i>Configuring the BFM</i>	27
3.3.4.2	<i>Training and Initializing the Link</i>	28
3.3.4.3	<i>Configuring the Reference Design</i>	28
3.3.4.4	<i>Testing Registers</i>	30
3.3.4.5	<i>Testing Internal SRAM</i>	30
3.3.4.6	<i>Programming DMA Transfers</i>	31
3.3.4.7	<i>Handling Interrupts</i>	31
3.3.4.8	<i>Performing a Read Transfer in DMA0 Scatter Gather Mode</i>	32
3.3.4.9	<i>Performing a Write Transfer in DMA1 Scatter Gather Mode</i>	33

List of Tables

Table 1: EDA Tools Requirements	9
Table 2: Source package features	10
Table 3: Full package features	10
Table 4: Board / Eval package features	11
Table 5: Directory structure for the Source package	12
Table 6: Directory structure for the Full and Board/Eval packages	13
Table 7: Description of Reference Design files	23
Table 8: Reference Design configuration of the Core	24
Table 9: Description of Simulation files	25
Table 10: Simulation Arguments	26

List of Figures

Figure 1: The choose wrapper window	14
Figure 2: The Define Settings for <wrapper name> instance Window: General tab	15
Figure 3: The Define Settings for <wrapper name> instance Window: Options tab	16
Figure 4: Launching Xilinx Coregen.....	17
Figure 5: Example of settings in a EZ DMA IP core instance.....	17
Figure 6: Simulation environment	18
Figure 7: Top-level blocks of the Reference Design.....	21
Figure 8: Architecture of the simulation environment	22
Figure 9: .Overview of the simulation environment	25

Preface

About this Document

Intended Audience

This document has been written for design managers, system engineers, and designers who are evaluating or using the PLDA EZDMA2 IP.

Scope

This document describes how to integrate the PLDA EZDMA2 IP into your design flow as quickly as possible (installing, customizing, integrating, and simulating the Core).

Typographical Conventions

<i>italic</i>	Highlights important notes or publications
bold	Highlights interface elements.
COURIER NEW	DENOTES TEXT USED IN A CODE EXAMPLE OR A SIGNAL.

Additional Reading

This section lists additional resources from PLDA and third-parties.

PLDA periodically updates its documentation. Please contact PLDA Technical Support or check the Web site at <http://www.plda.com> for current versions.

PLDA Publications

Please refer to the following documents for further information:

- *EZDMA2 IP Core Reference Manual*: The *Reference Manual* provides the complete functional description of the PLDA EZDMA2 IP Core.
- *Bus Functional Model Reference Manual*: The *BFM Reference Manual* provides the complete functional description of the PLDA PCI Express Testbench.
- *Software Tools for PCI/PCI-X and PCI Express IP Cores*: The *Software Reference Manual* describes PLDA's Software Development Kit (SDK).
- *Build History*: The *Build History* lists changes made to the packaging of each build.
- *Revision History*: The *Revision History* lists changes made to the RTL of the Core.

Other Publications

Please refer to the following documents for information on specification standards:

- *PCI Express™ Base Specification Revision 2.0*
- *Virtex-6 LogiCORE Integrated Block v1.4 for PCI Express Designs User Guide UG517*
- *Spartan-6 LogiCORE Integrated Block v1.2 for PCI Express Designs User Guide UG654*
- *Virtex-5 LogiCORE Endpoint Block Plus v1.13 for PCI Express Designs User Guide UG341*

Feedback and Contact Information

Feedback about this Document

PLDA welcomes comments and suggestions about its documentation. Please contact PLDA Technical Support and provide the following information:

- the title of the document
- the page number to which your comments refer
- a description of your comments

Contact information

Corporate Headquarters

PLDA
Parc club du golf - Bât. 11a
Rue Guilibert
13856 Aix-en-Provence Cedex 3 - France

Tel: USA +1 408 273 4528 - International +33 442 393 600

Fax: +33 442 394 902

Sales

For sales questions, please contact sales@plda.com.

Technical Support

For technical support questions, please contact PLDA Support at http://www.plda.com/plda_login.php using the Support Center if you have a PLDA online account.

If you don't have a PLDA account, contact http://www.plda.com/support_enquiry.php.

Chapter 1 Before you Start...

1.1 System Requirements

To install the EZDMA2 IPCore package, you need:

- **Memory:** 1 GB of RAM or greater
- **Operating System:** Windows 2000/XP/Vista or any UNIX/Linux platform supporting Java
- **Hard Disk:** 1 GB for Core installation and component design

1.2 EDA Tools Requirements

The following table describes EDA Tools Requirements unique to Windows, unique to Unix / Linux, and common to both operating systems:

Table 1: EDA Tools Requirements

Windows 2000 / XP / Vista	Unix / Linux
<ul style="list-style-type: none"> • ModelSim PE or SE 6.5 or above 	<ul style="list-style-type: none"> • ModelSim SE 6.5 or above • NCSim version 6.1.1 s002
<ul style="list-style-type: none"> • ISE® 11.4 • A valid license for the Xilinx Virtex-5 LogiCORE Endpoint Block Plus for PCI Express, or the Virtex 6 or Spartan 6 Integrated Blocks for PCI Express • Xilinx Platform USB cableQuartus II v9.1 • Altera USB Blaster Cable 	

1.3 Package Features

The EZDMA2 IP Core is available for Windows or Unix/Linux in any of the following packages:

- Xilinx Source
 - **Windows:** pcie-ezdma_vXXXbuildYYY_fpga_source.zip
 - **Unix / Linux:** pcie-ezdma_vXXXbuildYYY_fpga_source.tar.gz
- Xilinx Full
 - **Windows:** pcie-ezdma_vXXXbuildYYY_fpga_full.zip
 - **Unix / Linux:** pcie-ezdma_vXXXbuildYYY_fpga_full.tar.gz
- Xilinx Board / Eval
 - **Windows:** pcie-ezdma_vXXXbYYY_fpga_board_eval.zip
 - **Unix / Linux:** pcie-ezdma_vXXXbYYY_fpga_board_eval.tar.gz

1.3.1 Xilinx Source Package

The Source package contains full source code for the Core.

Table 2: Source package features

Core	Source <ul style="list-style-type: none"> • VHDL: Original clear-text source code • Verilog: Original clear-text source code
	Synthesis <ul style="list-style-type: none"> • VHDL: Single Xilinx clear-text file • Verilog: Single Xilinx clear-text file
PLDA testbench	<p>The PLDA testbench, exclusively compatible with the Core, includes the following:</p> <ul style="list-style-type: none"> • Unlimited duration for purchased products • BFM Instance: Root Port, Endpoint • Number of Transactions: 1,000 • Monitor <p>Refer to the PLDA BFM Reference Manual for more information.</p> <ul style="list-style-type: none"> • Modelsim <ul style="list-style-type: none"> • VHDL: VHDL pre-compiled library, multi-OS • Verilog: Verilog pre-compiled library, multi-OS • Cadence NCSim <ul style="list-style-type: none"> • VHDL: VHDL-protected, UNIX/Linux • Verilog: Verilog-protected, UNIX/Linux • Synopsys VCS <ul style="list-style-type: none"> • Verilog: Verilog-protected, Linux

1.3.2 Xilinx Full Package

The Full package is not time-limited and allows you to generate programming files for any supported device.

Table 3: Full package features

Core	Simulation <ul style="list-style-type: none"> • Modelsim <ul style="list-style-type: none"> • VHDL: pre-compiled VHDL library • Verilog: pre-compiled Verilog library • Cadence NCSim <ul style="list-style-type: none"> • VHDL: VHDL-protected, UNIX/Linux • Verilog: Verilog-protected, UNIX/Linux
	Synthesis <ul style="list-style-type: none"> • VHDL: Single Xilinx encrypted file • Verilog: Single Xilinx encrypted file

Table 3: Full package features

PLDA testbench	<p>The PLDA testbench, exclusively compatible with the Core, includes the following:</p> <ul style="list-style-type: none"> • Unlimited duration for purchased products • BFM Instance: Root Port, Endpoint • Number of Transactions: 1,000 • Monitor <p>Refer to the PLDA BFM Reference Manual for more information.</p> <ul style="list-style-type: none"> • Modelsim <ul style="list-style-type: none"> • VHDL: VHDL pre-compiled library, multi-OS • Verilog: Verilog pre-compiled library, multi-OS • Cadence NCSim <ul style="list-style-type: none"> • VHDL: VHDL-protected, UNIX/Linux • Verilog: Verilog-protected, UNIX/Linux • Synopsys VCS <ul style="list-style-type: none"> • Verilog: Verilog-protected, Linux
-----------------------	--

1.3.3 Xilinx Board / Eval Package

When testing hardware with the Eval package, the design will only work for a limited time. For the Board package, there is no time limit when connecting protocoresh signals with PLDA boards.

Table 4: Board / Eval package features

Core	<p>Simulation</p> <ul style="list-style-type: none"> • Modelsim <ul style="list-style-type: none"> • VHDL: pre-compiled VHDL library • Verilog: pre-compiled Verilog library • Cadence NCSim <ul style="list-style-type: none"> • VHDL: VHDL-protected, UNIX/Linux • Verilog: Verilog-protected, UNIX/Linux
	<p>Synthesis</p> <ul style="list-style-type: none"> • VHDL: Single Xilinx encrypted file • Verilog: Single Xilinx encrypted file
PLDA testbench	<p>The PLDA testbench, exclusively compatible with the Core, includes the following:</p> <ul style="list-style-type: none"> • Unlimited duration for purchased products • BFM Instance: Root Port, Endpoint • Number of Transactions: 1,000 • Monitor <p>Refer to the PLDA BFM Reference Manual for more information.</p> <ul style="list-style-type: none"> • Modelsim <ul style="list-style-type: none"> • VHDL: VHDL pre-compiled library, multi-OS • Verilog: Verilog pre-compiled library, multi-OS • Cadence NCSim <ul style="list-style-type: none"> • VHDL: VHDL-protected, UNIX/Linux • Verilog: Verilog-protected, UNIX/Linux • Synopsys VCS <ul style="list-style-type: none"> • Verilog: Verilog-protected, Linux

1.4 Installing the Package

1.4.1 Windows 2000/XP/Vista

Unzip the package obtained from PLDA web site. Files are extracted to your hard drive in a directory named **pcie-ezdma_vXXX_bYYY_ZZZ**, where “XXX” is the Core version number, “YYY” is the build number, and “ZZZ” is the supported technology.

1.4.2 UNIX / Linux platform

1. Unzip the tar.gz file using unzip software such as WinRAR or 7-Zip.
2. Open a shell and set the working directory to the directory where the Core package has been downloaded from the PLDA web site.
3. Create an empty directory.
4. At the prompt, type:

```
tar -xzf pcie-ezdma_vXXX_bYYY_ZZZ.tar.gz -C <your directory path>
```

Where “XXX” is the Core version number, “YYY” is the build number, and “ZZZ” is the supported technology.

1.5 Exploring the Installed Files

1.5.1 Xilinx Source Package

Table 5: Directory structure for the Source package

<ul style="list-style-type: none"> • core <ul style="list-style-type: none"> • simulation • Modelsim <ul style="list-style-type: none"> • vhdl: compiled core library • vlog: compiled core library • ncsim <ul style="list-style-type: none"> • vhdl: encrypted core file • vlog: encrypted core file • source <ul style="list-style-type: none"> • vhdl: core clear-text source code • vlog: core clear-text source code • synthesis <ul style="list-style-type: none"> • xilinx <ul style="list-style-type: none"> • vhdl: core clear-text source file • vlog: core clear-text source file • documentation <ul style="list-style-type: none"> build_history.pdf revision_history.pdf getting_started.pdf reference_manual.pdf pciebfm_reference_manual.pdf

- **ref_design**
 - **simulation**
 - **Endpoint**
 - **source**
 - **Endpoint**
 - **synthesis**
 - **Endpoint**
- **software**
 - **windows:** PLDA software Tools for Windows OS
 - **linux:** PLDA software Tools for Linux OS
- **testbench:** PLDA BFM
- **wizard:** EZDMA2 IP wizard provided for creating a top-level instance of the Core.

1.5.2 Xilinx Full and Board/Eval Packages

Table 6: Directory structure for the Full and Board/Eval packages

- **core**
 - **simulation**
 - **Modelsim**
 - **vhdl:** compiled core library
 - **vlog:** compiled core library
 - **ncsim**
 - **vhdl:** encrypted core file
 - **vlog:** encrypted core file
 - **synthesis**
 - **vhdl:** encrypted core file
 - **vlog:** encrypted core file
- **documentation**
 - build_history.pdf
 - revision_history.pdf
 - getting_started.pdf
 - reference_manual.pdf
 - pciebfm_reference_manual.pdf
- **ref_design**
 - **simulation**
 - **Endpoint**
 - **source**
 - **Endpoint**
 - **synthesis**
 - **Endpoint**
- **software**
 - **windows:** PLDA software Tools for Windows OS
 - **linux:** PLDA software Tools for Linux OS
- **testbench:** PLDA BFM
- **wizard:** EZDMA2 IP wizard provided for creating a top-level instance of the Core

1.6 Creating a Parameterized Instance of the Core

The EZDMA2 IP Wizard generates a VHDL or Verilog wrapper that instantiates the Core with custom values, input ports, and output ports.

WARNING:

- The file created by the Wizard *should not* be modified with a text editor. Open the Wizard and select an existing .vhd/.v file in order to modify an existing instance of the Core.
- If you have already created an instance of the Core and you create a second instance of the Core with the same name, the Wizard will overwrite your original file without warning.

Follow the directions below to launch the Wizard and create an instance of the Core:

1.6.1 Launching the EZDMA2 IP Wizard

1. Launch the Wizard:

- Windows: browse to the /wizard directory and run the run_wizard_ezdma.bat batch file to launch the Wizard GUI.
- Linux: open a terminal window and type: /bin/sh run_wizard_ezdma.sh (located in the ../wizard installation directory).

The following window appears:

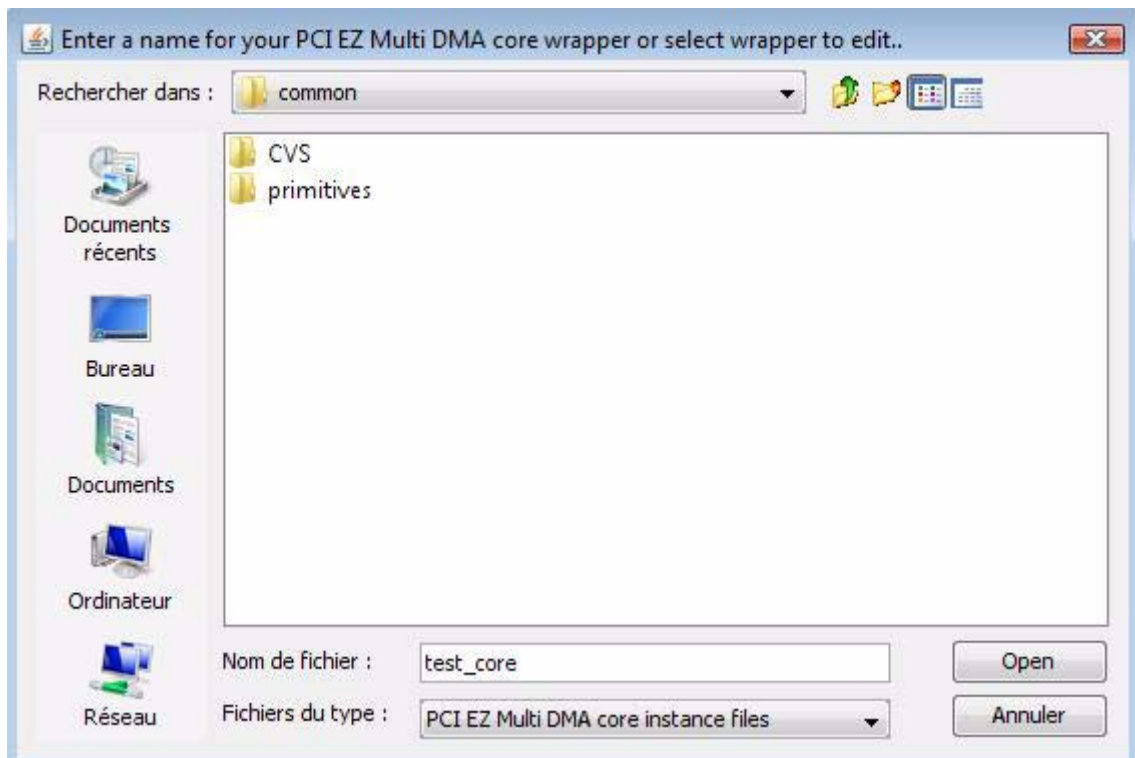


Figure 1: The choose wrapper window

2. To create a new wrapper, type a name in the **Name** text box and click **Open**. To modify an existing wrapper, browse your Hard Drive, select the desired wrapper, and click **Open**. The first page of the Wizard appears.

1.6.2 Customizing the Core with the EZDMA2 IP Wizard

Define Settings for <wrapper name> instance: General tab

The screenshot shows the 'PLDA PCI Express Core' configuration window. The title bar reads 'PLDA PCI Express Core'. Below the title bar, it says 'Define settings for 'test_core' instance :'. There are two tabs: 'General' (selected) and 'Options'. The 'General' tab contains the following settings:

- Device type: Virtex-5 (dropdown)
- Interface type: (dropdown)
- Protocol: (dropdown)
- Language: VHDL (dropdown)
- Maximum payload size: 256 bytes (slider)
- Maximum read request size: 512 bytes (slider)
- Number of outstanding requests: 1 (slider)
- Receive buffer size: n/a (text field)
- Number of built-in DMA channels: 1 (slider)
- User clock frequency: 125 MHz (dropdown)

At the bottom right of the window is a 'Recommended settings' button, and at the bottom center is a 'Generate file' button.

Figure 2: The Define Settings for <wrapper name> instance Window: General tab

The **Skip Customization** button is used to upgrade an existing instance of the EZ Module with a new version of the Module. In other words, if you receive a new version of the Module from PLDA, open your existing instance file with the Wizard and click **Skip Customization** in order to create a new instance compatible with the new version of the Module that uses your old parameters.

To create or edit an existing instance, modify the following parameters:

- **Device Type:** Select either Virtex-5, Virtex-6, or Spartan-6.
- **Interface Type:** This is set by default and cannot be changed.
- **Protocol:** This is set by default and cannot be changed.
- **Language:** VHDL, Verilog-95, or Verilog-2001
- **Maximum Payload Size:** .256 bytes to 512 bytes for Virtex-5; 256 bytes only for Virtex-6 and Spartan-6.
- **Maximum read request size:** This is set by default and cannot be changed.
- **Number of outstanding requests:** 1 - 8
- **Receive buffer size:** not applicable
- **Number of built-in DMA channels:** 1 - 8
- **User clock frequency:** The following values can be with Xilinx Coregen:
 - 250 MHz(32-bit/64-bit interface)
 - 125 MHz
 - 62.5 MHz (Virtex-5/Virtex-6 only)
 - 31.25 MHz (Virtex-6 only)
 - 250 MHz (128-bit interface - available for Virtex-6 and 5.0 Gbps packages only)

1.6.2.1 Define Settings for <wrapper name> instance: Options tab

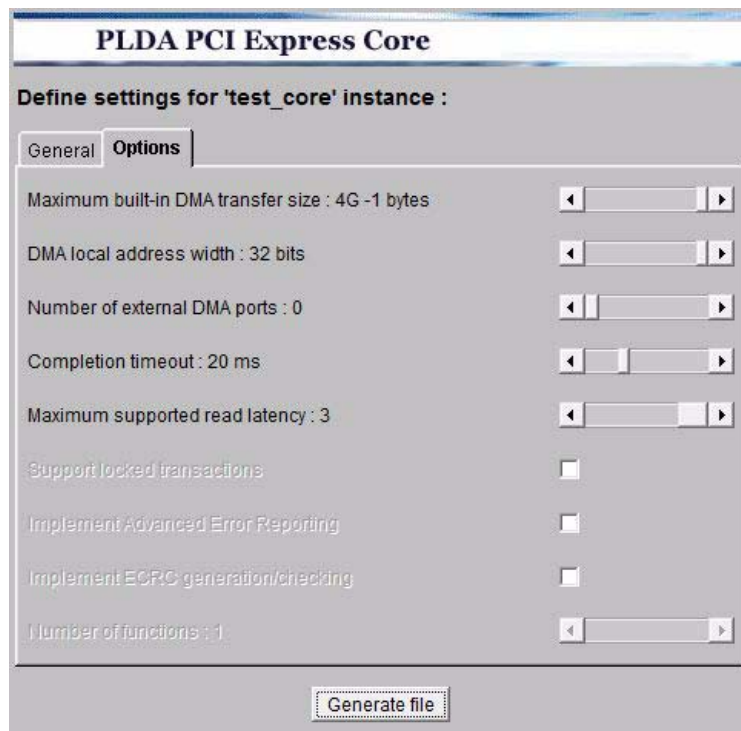


Figure 3: The Define Settings for <wrapper name> instance Window: Options tab

To customize DMA options, modify the following parameters:

- **Maximum DMA transfer size:** Maximum values are between 8K - 4G.
- **DMA local address width:** Values are between 13 - 32 bits.
- **Number of external DMA ports:** 0 - 8
- **Completion timeout:** Values are between 3 and 60 ms.
- **Maximum Supported Read Latency:** Values are between 0 - 3.
- **Support Locked Transactions:** Not supported.
- **Implement Advanced Error Reporting:** Not supported.
- **Implement ECRC generation/checking:** Not supported.
- **Number of functions:** Not supported.

1.6.3 Customizing Xilinx Hard IP Core for Use with EZDMA2 IP

When the EZDMA2 IP is customized, you must create a Xilinx Hard IP core instance as explained in this section.

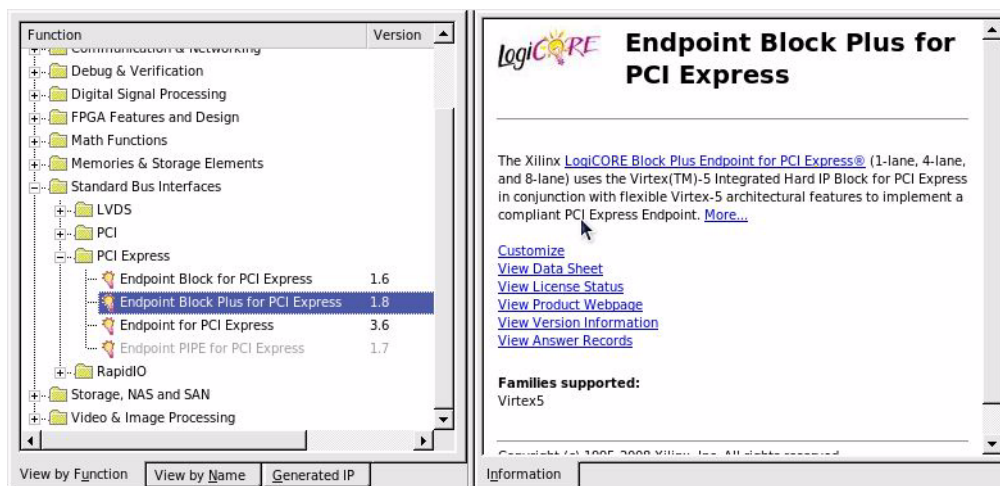


Figure 4: Launching Xilinx Coregen

1. Start Xilinx Coregen and create a new projects for your target Virtex-5 device.
2. Select **Endpoint Block for PCI Express** function and click **Customize**.
3. Open the EZDMA2 IP core instance created previously with a text editor.
4. The first few lines indicate mandatory values for some settings in Coregen, all other settings can be set to any value you choose.

```
-- megafunction wizard: %PLDA PCI Express EZ Core%
-- PLDA HDL Writer v10.1
-- This instance is for PLDA PCIEZ Multi DMA for Xilinx Hard IP build xxx x8@250Mhz
--
library ieee;
use ieee.std_logic_1164.all;

--
-- Xilinx Endpoint Block Plus for Express (v1.10 or later) must be configured as
-- indicated below for proper core behaviour :
--
-- - Maximum payload size supported : 256
-- - Multiple message capable : 1
--
-- all other settings are left to user's choice
```

Figure 5: Example of settings in a EZ DMA IP core instance

1.7 Simulating your Design

This section provides background information for running a functional simulation of the Core. The simulation environment includes the Core, the Core wrapper, the PCI Express BFM, and all or part of your design (application layer logic).

The figure below illustrates a typical simulation environment using a serial or parallel interface.

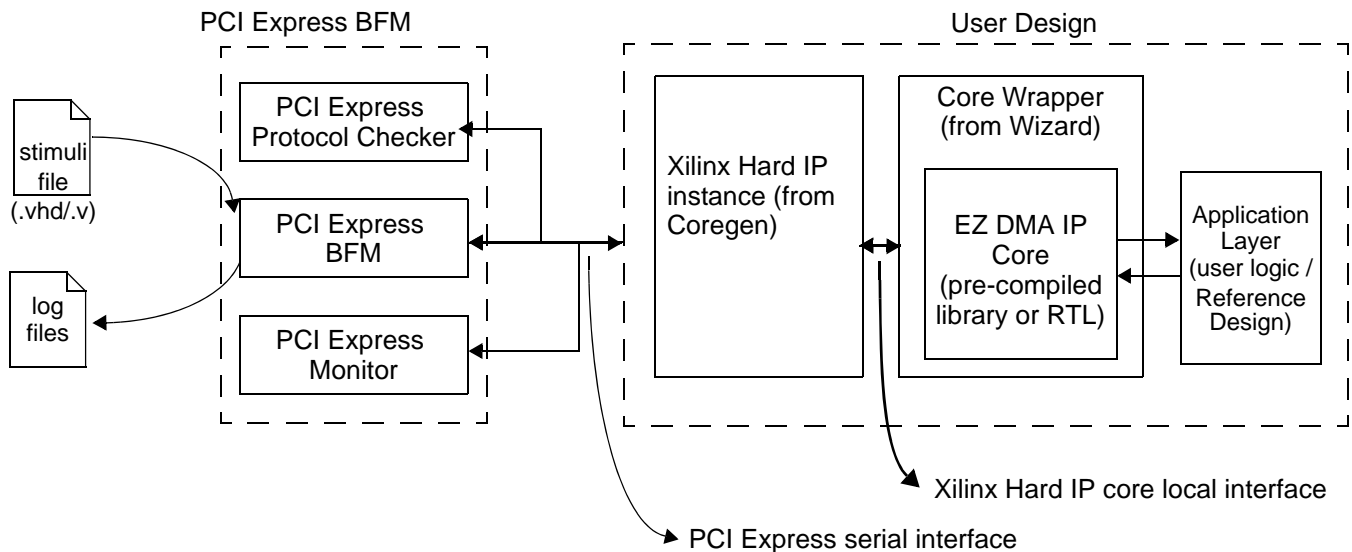


Figure 6: Simulation environment

The simulation architecture includes the following elements:

- **PCI Express BFM:** The Core package includes evaluation versions of the PLDA PCI Express BFM. The BFM is provided as compiled libraries or as protected code for various simulators. It includes a link monitor, and can be instantiated at the serial, parallel, and PIPE interface levels. Refer to the *BFM Reference Manual* in the documentation directory for additional information.
- **Xilinx Hard IP:** This is a customized instance of Xilinx LogiCore Endpoint Block Plus for PCI Express created with Xilinx Coregen wizard.
- **EZDMA2 IP Core wrapper:** This is the customized wrapper you create with the Core Wizard, which instantiates the Core with custom parameters, features, and ports. See [Section 1.6](#) for detailed information about the Wizard and available configuration options).
- **EZDMA2 IP Core:** Provided as clear-text RTL or obfuscated/scrambled RTL depending on your particular license (for synthesis) and as a compiled library (for simulation).
- **User Design:** Application layer logic implemented by the designer that connects to the Core's local Application Layer interface. For a tryout simulation, it is not required to have a fully functional or complete design, however, all of the Core's Application Layer inputs should be set to appropriate values. The interface between the Core and the Application Layer is described in the Reference Manual.

1.7.1 Simulating with the PLDA PCIe BFM

PLDA PCI Express BFM is available for the following simulation tools:

1.7.1.1 ModelSim

The BFM is delivered as a simulation library. To use the BFM, map the library **pciebfm_lib**. When compiling with Verilog, you must include the **.h** file located in the **include** directory. When launching vsim, add the BFM library with the following command:

```
vsim -L pciebfm_lib
```

1.7.1.2 NCSim and VCS

The BFM is delivered in the form of encrypted files. To use the BFM, compile the encrypted library **pciebfm_lib**.

When compiling with Verilog, you must include the **.h** file located in the **include** directory.

1.7.1.3 Aldec Active-HDL and Riviera-Pro

Aldec's Active-HDL and Riviera-Pro simulators are also supported by the EZDMA2 IP Core. To request the necessary simulation libraries and models, contact PLDA Support at http://www.plda.com/plda_login.php using the Support Center (if you have a PLDA online account) or http://www.plda.com/support_enquiry.php (if you don't have a PLDA account).

1.7.2 Simulating with ModelSim

The following directions describe how to create a project and compile the necessary libraries in order to run a simulation with the Core. Note that this section is offered as background information only. The `plda_simulate.tcl` script provided with the Core package automatically performs these tasks and starts a simulation for you.

1. Launch ModelSim.
2. Select the working directory:
 - Choose Change Directory from the File menu. The **Choose a Directory** window appears.
 - Browse your hard drive and select or create a directory in which to compile your project.
3. Create a work library:
4. Map the required libraries. There are three libraries necessary to run a simulation of the Core.
 1. Map the PLDA BFM (PCI-Express Testbench) library. At the ModelSim prompt, type:


```
vmap pciebfm_lib <absolute path>/testbench/plda/modelsim/<XXX>/pciebfm_lib
```

 where "<absolute path>" is replaced by the directory in which you installed your project and "<XXX>" is replaced by either "vhdl" or "vlog", depending on your programming language.
 2. Map the EZDMA2 IP library. At the ModelSim prompt, type:


```
vmap ez_lib <absolute path>/core/simulation/modelsim/<XXX>/ez_lib
```

 where "<absolute path>" is replaced by the directory in which you installed your project and "<XXX>" is replaced by either "vhdl" or "vlog", depending on your programming language.

Warning: If you have recently upgraded your version of ModelSim, we recommend refreshing previously-compiled libraries. At the ModelSim prompt, type:

```
vcom -refresh -work <name_of_library>
```

5. Compile your core instance file(s) previously generated with the EZDMA2 IP Wizard.
 - If you are using VHDL, type:


```
vcom -work work -93 <absolute path>/<filename>.vhd
```
 - If you are using Verilog, type:


```
vlog -work work <absolute path>/<filename>.v
```
6. Compile all your design files.
7. Compile the stimuli file used to simulate your design by generating traffic with the Core. An example file, `ref_design_stimuli`, is provided with the Reference Design.
8. Compile your top level that connects your design with the Core, the BFM, and the stimuli instance. Example files (`ref_design_testb`) are provided with the Reference Design.
9. Simulate the project, by typing:


```
vsim -t ps -L <bfm_library> -L ez_lib work.<design_top_level>
```

1.7.3 Additional Xilinx Considerations

In order to simulate your design, Xilinx smart models must be installed and configured and Xilinx simulation libraries must be compiled. Please refer to the ISE installation guide for details.

Use the memory model furnished with the Xilinx Library of Parameterized Modules (UNISIM). Note that the DCRAM located in the Xilinx primitive directory uses this Xilinx library for compilation.

Chapter 2 Core Synthesis

2.1 Creating a New Project with ISE

To create a new project with ISE:

1. Launch ISE and select **New Project...** from the **File** menu. The New Project window appears.
2. Select a project name, project location, and keep HDL Top-Level module Type.
3. Click **Next** and choose your device.
4. Click **Next**. Do not create new files.
5. Click **Next**. Add the following files:
 - Click **Add** and browse for `core/synthesis/xilinx/vhdl/ezdma_xilinx_<package type>.vhd` where `<package type>` is replaced by "source" or "full".
 - Click **Add** and browse for your core instance created by the wizard.
 - Click **Add** and browse for Xilinx Hard IP core design files generated by Coregen
 - Click **Add** and browse for any additional design files.
6. Click **Next**. Review your settings and click **Finish**.

Once the project is created, change Synthesize properties (In Process View window, right click on Synthesize-XST):

- Select **Optimization Goal Speed** from the **Synthesis Options** menu.
- Select **Optimization Effort High** from the **Synthesis Options** menu.
- Select **Hierarchy No** from the **Synthesis Options** menu.
- Uncheck **Rom extraction**.
- Select **Register Balancing No** from the **Xilinx Specific Options** menu.

You can leave other Synthesize options with default values.

2.2 Place and Route with ISE

Note: There is a known issue with Endpoint Block Plus Wrapper v1.10 and v1.10.1 that can cause a map to fail. See Answer Record #32727 - 'Endpoint Block Plus Wrapper v1.10 and v1.10.1 for PCI Express - MAP failing to complete due to predictable IP placement constraints' on the Xilinx web site for a full description and workaround for this issue.

1. Change Implement properties (In Process View window, right click on Implement Design):
 - Check **Perform Timing-Driven Packing and Placement** from the **Map Properties** menu.
 - Select **Map Effort Level High** or **Medium** from the **Map Properties** menu.
 - Check **Global Optimization, Equivalent Register Removal**, and **Allow Logic Optimization Across Hierarchy** from the **Map Properties** menu for better performance.
 - Select **Place & Route Effort Level High** or **Medium** from the **Place & Route Properties** menu.
2. Add your UCF file, which must contain at least a:
 - clock pin assignment
 - clock timing constraint
 - MGT and MGTCLK area constraint
3. Start **Place and Route**.

2.3 Generating a Programming File with ISE

Because the PCI Express specifications stipulate that the core must be ready 100 ms after power-up, you must choose the appropriate configuration rate to configure your FPGA in less than 100 ms depending on the bitstream size (for example more than 20Mbits for a V4FX60) and the PROM you use. For example, you might configure your FPGA at 30MHz in parallel mode 8 bits for a V4FX60 ($20\text{Mbits}/(8\text{bits}\cdot 30\text{MHz}) = 83\text{ ms} < 100\text{ ms}$).

Chapter 3 Reference Design

3.1 Introduction

The Reference Design demonstrates an implementation example of the EZDMA2 IP Core. The complete Register Transfer Level (RTL) source code of the design (VHDL or Verilog) is provided so that you can:

- Perform hardware testing using a prototyping board.
- Modify the design in order to integrate the Core with your own design.

3.2 Architecture of the Reference Design

3.2.1 Top-Level Blocks

The Reference Design includes several blocks, illustrated below:

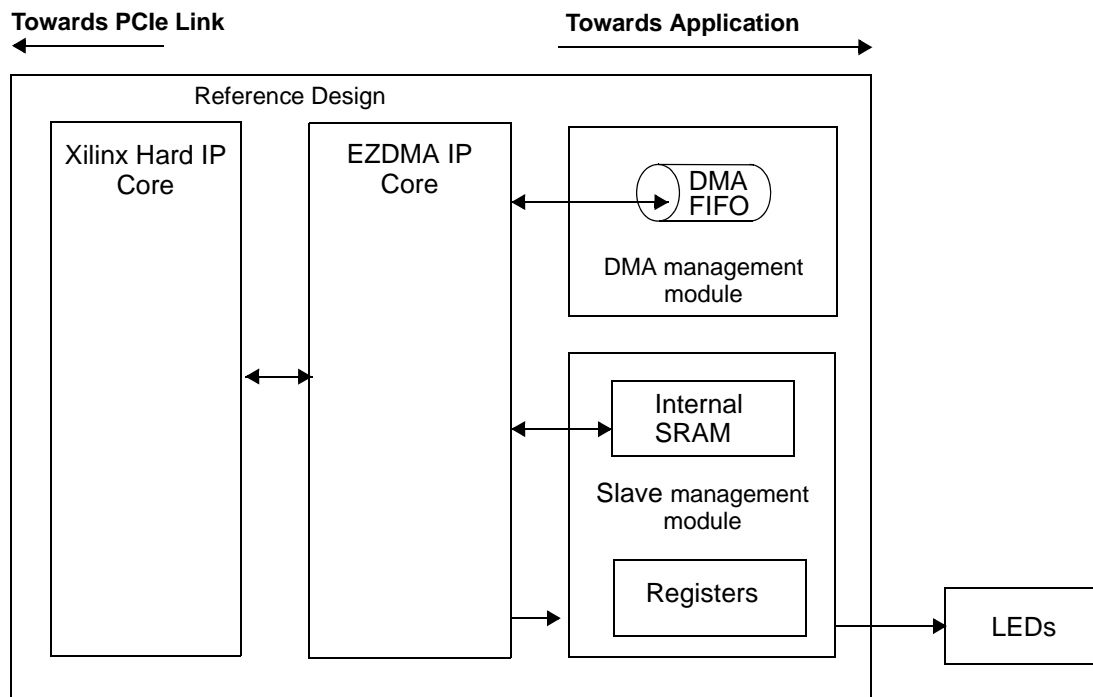


Figure 7: Top-level blocks of the Reference Design

- **Xilinx Hard IP core:** Custom instance of the core created by Xilinx Coregen.
- **EZDMA2 IP Core:** Custom instance of the core created by the EZDMA2 IP Core Wizard.
- **DMA management:** Handles two independent DMA channels, one that reads data from host memory, the other that writes the data back to host memory.
- **Slave management:** Handles target-mode read and write access to a memory-mapped on-chip SRAM and registers.

3.2.2 Reference Design Files and File Structure

The following figure provides a schema of the files implicated in the Reference Design. Each file is described in [Table 7](#)

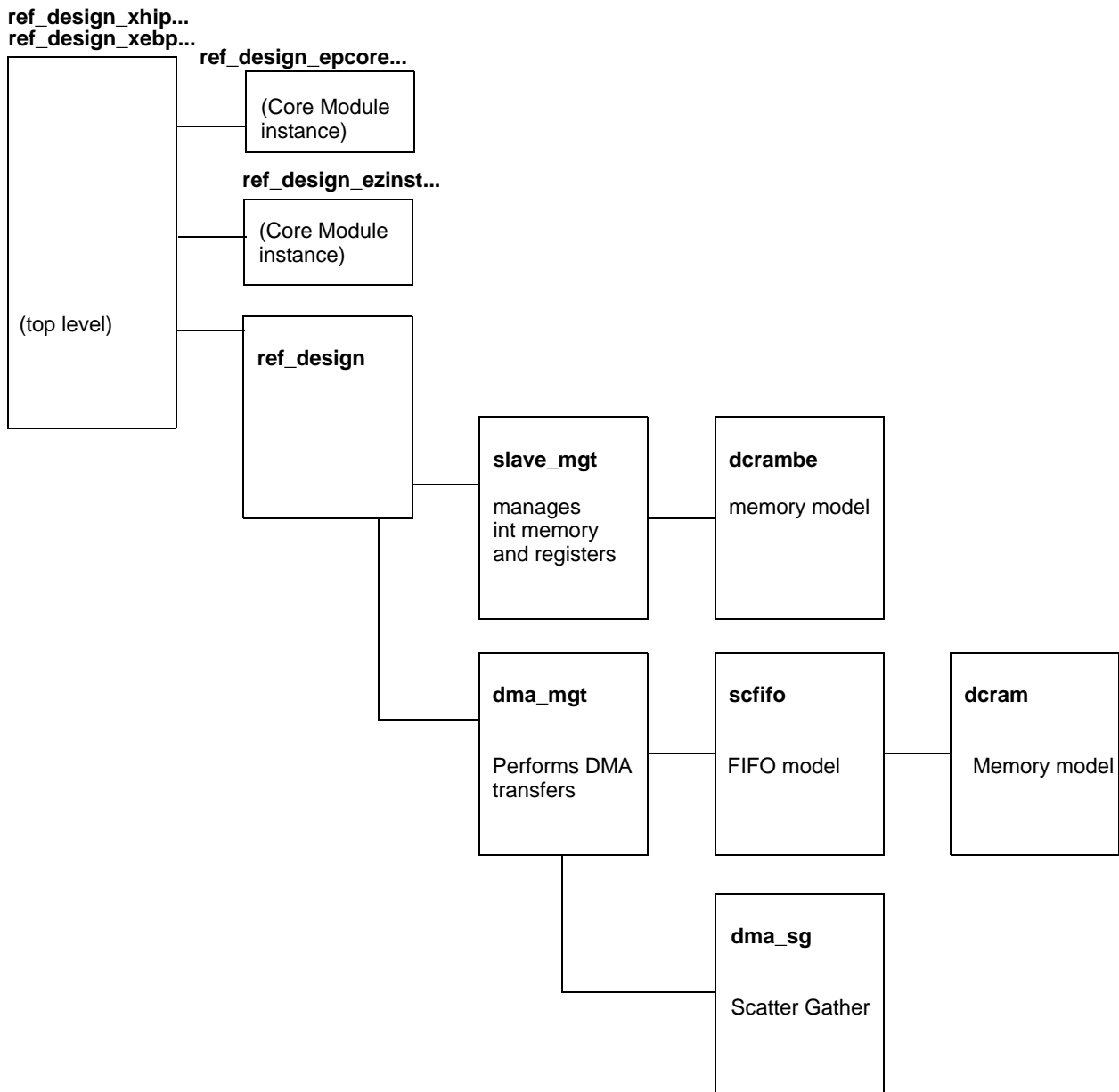


Figure 8: Architecture of the simulation environment

The following table describes each file used by the Reference Design.

Table 7: Description of Reference Design files

Module / File	Description
dcram	Generic DCRAM model available in ASIC (RTL description) and Virtex (uses Virtex memory primitives) versions.
scfifo	Generic Single Clock FIFO model
dcrambe	Generic DPRAM with Byte Enable model available in ASIC (RTL description) and Virtex (uses Virtex memory primitives) versions.
dma_sg	The DMA Scatter Gather module provides scatter-gather functionality to DMA 0 and DMA 1 channels.
dma_mgt	The DMA Management module handles two DMA channels that connect to the Core Master interface through a 64-bit X 256 Words FIFO. The FIFO uses PLDA's scfifo general-purpose single-clock FIFO model. A set of memory-mapped registers (accessible from the PCI Express bus) is used to set up DMA registers and start transfers.
slave_mgt	The Slave Management module instantiates a 1-KB synchronous dual-port RAM using PLDA's plda_dpram general-purpose DPRAM model. the memory is mapped in BAR 2/3 address space and can be read to or written from the PCI Express bus. The registers are mapped in the BAR 0 / 1 address space.
ref_design	Reference design top-level that connects the peripheral modules.
ref_design_ezinst ref_design_ezint_32 ref_design_ezinst_128	EZDMA2 IP Core custom instance: <ul style="list-style-type: none"> • DMA0: Write to FIFO, read from PCI Express • DMA1: Read from FIFO, write to PCI Express • DMA2: Completions • 6 Completion resources • BAR 0/1: 4 KB memory space (registers) • BAR 2/3: 4 KB memory space (internal SRAM)
ref_design_epcore...	Xilinx Core custom instance <ul style="list-style-type: none"> • BAR 0/1: 4 KB memory space (registers) • BAR 2/3: 4 KB memory space (internal SRAM) • 256 bytes maximum payload • 100MHz reference clock
ref_design_xhip... ref_design_xebp...	Reference Design top level for Hard IP or Endpoint Block Plus.

3.2.3 Registers

The following table describes the registers used by the Reference Design:

Table 8: Reference Design configuration of the Core

Register	Description	Offset
DMA0_ADDR32	DMA0 address bits [31:0]	00h
DMA0_ADDR64	DMA0 address bits [63:32]	04h
DMA0_SIZE	DMA0 transfer size in bytes	08h
DMA0_CTRL	DMA Control <ul style="list-style-type: none"> • DMA0_CTRL[1]: Enables demo mode • DMA0_CTRL[2]: Starts DMA • DMA0_CTRL[3]: Abort transfer and clear FIFO • DMA0_CTRL[4]: Enable scatter-gather • DMA0_CTRL[7:5]: reserved • DMA0_CTRL[11:8]: Reports DMA0 channel state • DMA0_CTRL[9:15]: reserved • DMA0_CTRL[31:16]: Reports number of DMA finished per 50 ms period (demo mode) 	0Ch
DMA1	DMA1 control registers (same layout as DMA0)	10h..1Ch
STATUS_REG	Bits [7:0] report reference design version.	20h
MAILBOX_REG	General purpose 32-bit read / write register. Bits [2:0] are connected to onboard LEDs	24h
INT_REG	Bit 0 indicates an interrupt request; writing 1 to this bit clears interrupt	34h
reserved	--	other

3.3 Simulation of the Reference Design

3.3.1 Overview

The following figure offers an overview of the simulation environment:

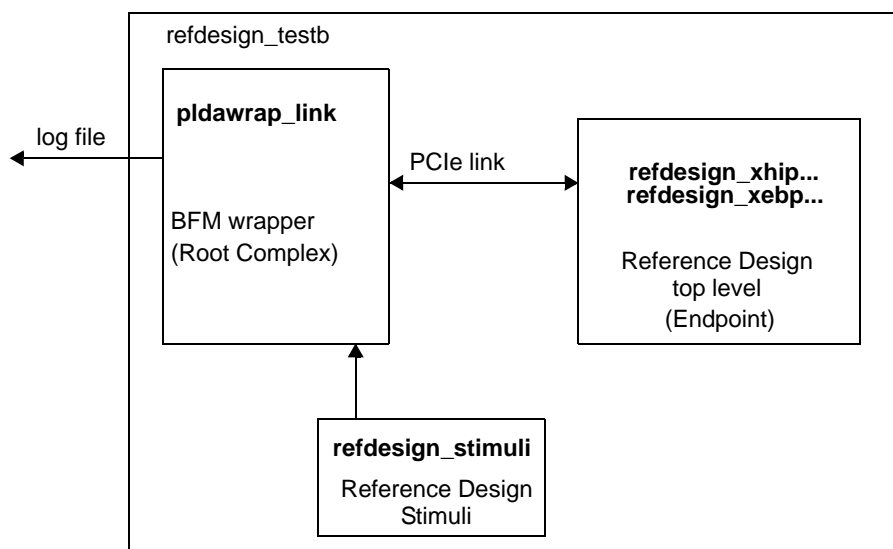


Figure 9: .Overview of the simulation environment

The Reference Design shown above (refdesign_testb) is illustrated in greater detail in [Figure 8](#) and described in [Table 7](#).

The following table describes each file used in the Simulation environment:

Table 9: Description of Simulation files

Module/File	Description
refdesign_stimuli	Sends PCI Express transactions and sets up the BFM.
pldawrap_link	Connects the Monitor, Checker, and BFM with the serial interface.
refdesign_testb	Testbench design that connects the BFM and the Reference Design. This file is the top-level file for simulation. It also contains a stimuli module that sends PCI Express transactions to configure and test the design.

3.3.2 Script Behavior and Arguments

The script copies and compiles the necessary files and launches a simulation based on the arguments passed. The script:

- Reads passed arguments
- Detects the operating system used
- Compiles the BFM, which is provided as a library for Modelsim or a protected file for NCsim and VCS
- Compiles the PLDA Core, which is provided as a library for Modelsim, an encrypted file for NCsim, or as source code for VCS
- Compiles reference source code and libraries depending on the programming language
- Simulates the design using the selected arguments

The following command line includes all potential arguments:

```
plda_simulate.tcl $language $simulator $lane $coretype $datapath$linkspeed
```

The following table describes each argument:

Table 10: Simulation Arguments

Argument	Argument Name	Potential Values
\$language	Programming Language	<ul style="list-style-type: none"> • vhdl • vlog
\$simulator	Simulator	<ul style="list-style-type: none"> • model (for ModelSim) • ncsim • vcs
\$lane	Number of Lanes	<ul style="list-style-type: none"> • x1 • x4 • x8
\$coretype	Core Type	<ul style="list-style-type: none"> • xebp (Virtex-5 Endpoint Block Plus) • xhip (Virtex-6/Spartan-6 Hard IP)
\$datapath	Data Path	<ul style="list-style-type: none"> • 32b (Spartan-6 only) • 64b • 128b (Virtex-6 x8 5.0 Gbps only)
\$linkspeed	Link Speed	<ul style="list-style-type: none"> • Gen1 • Gen2 (Virtex-6 only)

3.3.3 Launching the Simulation

3.3.3.1 1-Bit Simulation

Proper simulation requires installing ISE and installing, configuring, and compiling smart models for the targeted simulator. Refer to the Xilinx Synthesis and Verification Design Guide chapters 5 and 6 for more details.

3.3.3.2 ModelSim

A **.do** file is provided for simulating the project with ModelSim.

Follow the steps below to run a simulation:

1. Open the Modelsim simulator
2. Select File -> Change Directory and browse to `./ref_design/simulation/endpoint`
3. Type the following command lines:
 - `do plda_simulate.tcl model vhdl x1` for VHDL simulation
 - `do plda_simulate.tcl model vlog x1` for Verilog simulation

3.3.3.3 NCSim

Note: A Cadence Advanced Encryption Standard-64bit (28020) license is required for simulation with Xilinx SecureIP models. This license can be obtained from Cadence for free.

A script file is provided for simulating the project with NCSim. Follow the steps below to run the simulation:

1. Change directory to: `./ref_design/simulation/endpoint`
2. At the prompt, type:
 - `sh plda_simulate.tcl ncsim vhdl x1` for VHDL simulation
 - `sh plda_simulate.tcl ncsim vlog x1` for Verilog simulation
3. The simulation is launched and a log file is created in the project directory.

3.3.4 Annotated Explanation of the Simulation

The simulation performs the following tasks:

- Configures the BFM
- Trains and Initializes the Link
- Configures the Reference Design
- Tests transfer of a Memory Write to a mailbox register
- Tests Internal SRAM
- Programs DMA Transfers
- Handles Interrupts

3.3.4.1 Configuring the BFM

BFM initialization configures the log file format, sets BFM maximum payload size to 256 bytes, and sets three memory spaces inside the BFM:

- IO space: 256B, range DDDD0000h...DDDD00FFh
- 32-bit addressing memory space: 64KB, range AAAA0000h...AAAAFFFFh
- 64-bit addressing memory space: 64KB, range BBBBBBBBCCCC0000h...BBBBBBBBCCCCFFFFh

These memory spaces are used when performing DMA transfers from the EZ Reference Design.

Configuring the BFM

```
refdesign_stimuli.vhd
xbfm_print_comment (0, "### Initialise BFM");
xbfm_init (0, x"00000000", x"AAAA0000", x"BBBBBBBBCCCC0000");
xbfm_set_requesterid (0, x"0008");
xbfm_set_maxpayload (0, 256);
```

transaction log file

```
-----
--- Initialise BFM
-----
```

```
-----
Configuration parameters          : Value
-----
```

```
Device   Type           : Root Complex
Port     Type           : Downstream
Memory   Space 32 Bits   (Bytes) : 65536
Memory   Space 64 Bits   (Bytes) : 65536
Config   Space           (Bytes) : 4096
I/O      Space           (Bytes) : 256
-----
```

3.3.4.2 Training and Initializing the Link

The BFM and Reference Design perform Link training as soon as a simulation is started. This phase starts with the exchange of training set packets followed by the exchange of init flow control packets.

No commands can be sent until the Link is fully configured.

Training and Initializing the Link**refdesign_stimuli.vhd**

```
-- Wait for link to get initialised
  xbfm_wait_linkup (0);
```

transaction log file

```
-----
          D      T  SYMBOL  F CNT BYTE CNT          F L
          I SEQ A   /    M /   / HI     LO          PACKET  B B
TIME (100 ps) R # G  PACKET T LEN ADDRESS  ADDRESS  HI     LO          E E
-----
          108 ns D --- -- SKIP      - - - - - - - - - - - - - - - - - -
          180 ns D --- -- Training - - - - - - - - - - - - - - - - - -
                        Started
          180 ns D --- -- TS1      - - - - - - - - - - - - - - - - - -
          244 ns D --- -- TS1      - - - - - - - - - - - - - - - - - -
          .....
          6596 ns U --- -- TS2      - - - - - - - - - - - - - - - - - -
          6644 ns D --- -- TS2      - - - - - - - - - - - - - - - - - -
          6760 ns D --- -- IDLE     - 28 - - - - - - - - - - - - - - - - - -
          6764 ns D --- -- SKIP      - - - - - - - - - - - - - - - - - -
          6776 ns D --- -- Training - - - - - - - - - - - - - - - - - -
                        Done
          6776 ns D --- -- L0 State - - - - - - - - - - - - - - - - - -
          6780 ns D --- -- IFC1_P   - - - - - - - - - - - - - - - - - -
          6788 ns D --- -- IFC1_NP  - - - - - - - - - - - - - - - - - -
          .....
          7260 ns U --- -- IFC2_NP  - - - - - - - - - - - - - - - - - -
          7268 ns U --- -- IFC2_CPL - - - - - - - - - - - - - - - - - -
          7400 ns U --- -- IDLE     - 32 - - - - - - - - - - - - - - - - - -
          7404 ns U 000 -- ACK      - - - - - - - - - - - - - - - - - -
          7456 ns U --- -- IDLE     - 12 - - - - - - - - - - - - - - - - - -
          7460 ns U --- -- UFC_P    - - - - - - - - - - - - - - - - - -
-----
```

3.3.4.3 Configuring the Reference Design

The Reference Design Configuration Space must be configured before performing memory or DMA transfers. Refer to the *PCI Express Specifications* for a detailed description of configuration registers and plug-n-play configuration processes.

The Reference Design device is configured as follows:

3.3.4.4 Testing Registers

The first test performs a memory write to a mailbox register then reads it back.

Note that memory writes are posted and do not result in a completion whereas successful memory reads result in a completion with data CPLD, which return read data.

Testing registers

refdesign_stimuli.vhd

```
xbfm_print_comment (0,"## BAR0/1 : write mailbox register");
databuf(0):=x"76543210"; -- write mailbox register at address 24h
xbfm_burst (0,XBFM_MWR,x"1111111111110024",4,databuf,"000","00");
xbfm_wait (0);

xbfm_print_comment (0,"## BAR0/1 : read main status & mailbox registers");
databuf(0):=x"00000020"; -- expect to read "version 2.0"
databuf(1):=x"76543210"; -- expect to read what was written
xbfm_burst (0,XBFM_MRD,x"1111111111110020",8,databuf,"000","00");
xbfm_wait (0);
```

transaction log file

```
-----
-- BAR0/1 : write mailbox register
-----
*** 9222 ns : MWr @11111111.11111124 be=0F bc=4 bytes tc=0 attr=00b
*** write 76543210
      9280 ns D --- -- IDLE      - 50 -----
      9304 ns D 00A -- MWR       3 001 11111111 11111124 0000090F 60000001 F 0
                                      11111124 11111111
                                      ----- 10325476
      9312 ns D --- -- IDLE      - 1 -----
      9316 ns D 008 -- ACK       - --- -----
      9520 ns U --- -- IDLE      - 124 -----
      9524 ns U 00A -- ACK       - --- -----
-----
-- BAR0/1 : read main status & mailbox registers
-----
*** 9526 ns : MRd @11111111.11111120 be=FF bc=8 bytes tc=0 attr=00b
*** expect 00000012 76543210
      9584 ns U --- -- IDLE      - 14 -----
      9584 ns D --- -- IDLE      - 66 -----
      9588 ns U --- -- UFC_P     - --- -----
      9604 ns D 00B 0A MRD      1 002 11111111 11111120 00000AFF 20000002 F F
                                      11111120 11111111
      .....
      9944 ns U 009 0A CPLD     2 002 008      20      ABC00008 4A000002 - -
                                      ----- 00000A20
                                      10325476 12000000
-----
```

3.3.4.5 Testing Internal SRAM

This test performs single data read/write to internal SRAM memory then performs a larger 1KB transfer. The EZ Module Maximum Payload Size is 256 Bytes, so it is not possible to write more than 256 Bytes to the Reference Design in a single transfer. However, maximum read request size is not limited so it is possible read up to 4KB from the Reference Design in a single request.

Testing Internal SRAM

refdesign_stimuli.vhd

```
-- Perform single data read/write to SRAM
xbfm_print_comment (0,"## BAR2/3 : Single data read/write to SRAM");
databuf(0):=x"01234567";
xbfm_burst (0,XBFM_MWR,x"2222222222220000",4,databuf,"000","00");
xbfm_burst (0,XBFM_MRD,x"2222222222220000",4,databuf,"000","00");

databuf(0):=x"89ABCDEF";
```

```

xbfm_burst (0,XBFM_MWR,x"2222222222220004",4,databuf,"000","00");
xbfm_burst (0,XBFM_MRD,x"2222222222220004",4,databuf,"000","00");
xbfm_wait (0);

-- prepare a ramp in data buffer
xbfm_buffer_fill (256,databuf);

-- Maximum payload size is 256 bytes so transfer must be split into 4 blocks
xbfm_print_comment (0,"## BAR2/3 : Write 1KB to SRAM");
xbfm_burst (0,XBFM_MWR,x"2222222222220000",256,databuf,"000","00");
xbfm_burst (0,XBFM_MWR,x"2222222222220100",256,databuf,"000","00");
xbfm_burst (0,XBFM_MWR,x"2222222222220200",256,databuf,"000","00");
xbfm_burst (0,XBFM_MWR,x"2222222222220300",256,databuf,"000","00");
xbfm_wait (0);

-- read data from SRAM and check all data with corresponding data_buf value
xbfm_print_comment (0,"## BAR2/3 : Read 1KB from SRAM");
xbfm_burst (0,XBFM_MRD,x"2222222222220000",1024,databuf,"000","00");
xbfm_wait (0);

```

3.3.4.6 Programming DMA Transfers

This test performs a 1 KB memory transfer using DMA channels:

- DMA0 is programmed to read 1KB of data from the BFM 64-bit address space at address BBBBCCCC0000h. Data is written to the FIFO.
- DMA1 is programmed to read 1KB of data from the same FIFO and write it to the BFM 32-bit address space at address AAAA0000h.

DMA transfers starts automatically as soon as the local address register is written.

Programming DMA Transfers

refdesign_stimuli.vhd

```

xbfm_print_comment (0,"## DMA0 : program read transfer (write to FIFO)");
databuf(0):=x"CCCC0000"; -- pci address (31..0) : read data from NVS BFM 64-bit mem space
databuf(1):=x"BBBBBBBB"; -- pci address (63..32)
databuf(2):=x"00000400"; -- transfer size (1KB)
databuf(3):=x"00000004"; -- write '1' to bit 3 to start transfer
xbfm_burst (0,XBFM_MWR,x"1111111111110000",16,databuf,"000","00");

xbfm_print_comment (0,"## DMA1 : program write transfer (read from FIFO)");
databuf(0):=x"AAAA0000"; -- pci address (31..0) : write data to NVS BFM 32-bit mem space
databuf(1):=x"00000000"; -- pci address (63..32)
databuf(2):=x"00000400"; -- transfer size (1KB)
databuf(3):=x"00000004"; -- write '1' to bit 3 to start transfer
xbfm_burst (0,XBFM_MWR,x"1111111111110010",16,databuf,"000","00");

```

3.3.4.7 Handling Interrupts

The Reference Design automatically fires an interrupt when DMA transfers are complete, and the stimuli module waits for an "INTA pin asserted" message. Upon reception, the interrupt register is read to verify that bit 0 is set to 1. A 1 is written to this bit in order to clear the interrupt and an "INTA pin deasserted" message results.

Handling Interrupts

refdesign_stimuli.vhd

```

-- Wait for "INTA pin asserted" message
xbfm_wait_event(0,XBFM_INTAA_RCVD);

-- Read interrupt register content
xbfm_print_comment (0,"## Interrupt : read & clear interrupt register");
databuf(0):=x"00000001";
xbfm_burst (0,XBFM_MRD,x"1111111111110034",4,databuf,"000","00");

```

```
-- Clear interrupt register content
xbfm_burst (0,XBFM_MWR,x"1111111111110034",4,databuf,"000","00");

-- wait for "INTA pin de-asserted" message
xbfm_wait_event(0,XBFM_INTAD_RCVD);
```

transaction log file

```
18572 ns U 01C -- MSG      1 000 ----- ABC00020 34000000 - ASRT A
                                           00000000 00000000

-----
--- Interrupt : read & clear interrupt register
-----

18636 ns D --- -- IDLE      - 19 -----
18640 ns D --- -- UFC_P      - --- -----
18664 ns D 01B 16 MRD      1 001 11111111 11111134 0000160F 20000001 F 0
                                           11111134 11111111

18692 ns D 01C -- MWR      3 001 11111111 11111134 0000170F 60000001 F 0
                                           11111134 11111111
                                           ----- 01000000

.....
19004 ns U 01D 16 CPLD      2 001 004      34      ABC00004 4A000001 - -
                                           ----- 00001634
                                           ----- 01000000

19012 ns U --- -- UFC_P      - --- -----
19032 ns U --- -- IDLE      - 4 -----
19052 ns U 01E -- MSG      1 000 ----- ABC00024 34000000 - - DSRT_A
                                           00000000 00000000
```

3.3.4.8 Performing a Read Transfer in DMA0 Scatter Gather Mode

This part of the script creates a chained list of descriptors and stores it in BFM memory.

It then starts DMA0 in scatter-gather mode to perform a read transfer, pointing to the beginning of the descriptor chain.

Read transfer in DMA0 Scatter Gather mode

refdesign_stimuli.vhd

```
-----
-- DMA0 : program scatter-gather read DMA transfer
-----

-- Prepare read descriptors
-- read descriptor #1
databuf(0):=x"CCCC0400"; -- pci address (31..0)
databuf(1):=x"BBBBBBBB"; -- pci address (63..32)
databuf(2):=x"00000100"; -- page size
databuf(3):=x"CCCC0014"; -- next page pointer (31..1) & End chain bit
databuf(4):=x"BBBBBBBB"; -- next page pointer (63..32)

-- read descriptor #2
databuf(5):=x"CCCC0500";
databuf(6):=x"BBBBBBBB";
databuf(7):=x"00000100";
databuf(8):=x"CCCC0028";
databuf(9):=x"BBBBBBBB";

-- read descriptor #3
databuf(10):=x"CCCC0600";
databuf(11):=x"BBBBBBBB";
databuf(12):=x"00000100";
databuf(13):=x"CCCC003C";
databuf(14):=x"BBBBBBBB";

-- read descriptor #4
databuf(15):=x"CCCC0700";
databuf(16):=x"BBBBBBBB";
databuf(17):=x"00000100";
databuf(18):=x"00000001"; -- bit 0 is '1' => end of chain
```

```

databuf(19):=x"00000000";

-- Write read descriptor to BFM 64-bit memory space
xbfm_memory_write (0,XBFM_MEM64,x"00000000",20,databuf);

-- Write data to BFM 64-bit memory space
xbfm_buffer_fill (256,databuf);
xbfm_memory_write (0,XBFM_MEM64,x"00000400",256,databuf);

-- Program DMA transfer
xbfm_print_comment (0,"## DMA0 : program SG read transfer (write to FIFO)");
databuf(0):=x"CCCC0000"; -- pci address (31..0)
databuf(1):=x"BBBBBBBB"; -- pci address (63..32)
databuf(2):=x"00000400"; -- transfer size
databuf(3):=x"00000014"; -- write '1' to bit 2 to start transfer
xbfm_burst (0,XBFM_MWR,x"1111111111110000",16,databuf,"000","00");

```

3.3.4.9 Performing a Write Transfer in DMA1 Scatter Gather Mode

This part of the script creates a chained list of descriptors and stores it in BFM memory.

It then starts DMA1 in scatter-gather mode to perform a write transfer, pointing it to the beginning of the descriptor chain.

Write transfer in DMA1 Scatter Gather mode

refdesign_stimuli.vhd

```

-----
-- DMA1 : program scatter-gather write DMA transfer
-----

-- write descriptor #1
databuf(0):=x"AAAA0400"; -- pci address (31..0)
databuf(1):=x"00000000"; -- pci address (63..32)
databuf(2):=x"00000200"; -- page size
databuf(3):=x"AAAA0014"; -- next page pointer (31..1) & End chain bit
databuf(4):=x"00000000"; -- next page pointer (63..32)

-- write descriptor #2
databuf(5):=x"AAAA0600";
databuf(6):=x"00000000";
databuf(7):=x"00000200";
databuf(8):=x"00000001"; -- bit 0 is '1' => end of chain
databuf(9):=x"00000000";

-- Write read descriptor to BFM 32-bit memory space
xbfm_memory_write (0,XBFM_MEM32,x"00000000",10,databuf);

-- Program DMA transfer
xbfm_print_comment (0,"## DMA1 : program SG write transfer (read from FIFO)");
databuf(0):=x"AAAA0000"; -- pci address (31..0)
databuf(1):=x"00000000"; -- pci address (63..32)
databuf(2):=x"00000400"; -- transfer size
databuf(3):=x"00000014"; -- write '1' to bit 2 to start transfer
xbfm_burst (0,XBFM_MWR,x"1111111111110010",16,databuf,"000","00");

```
