

Introduction



EXCALIBUR™

The Excalibur™ device performance designs are two basic designs that are used for embedded stripe throughput benchmarks. The designs are used to develop metrics for many different types of transactions originating from both the embedded processor and a PLD master in Excalibur devices. The designs are also useful as a starting point for resolving other complex issues arising during system development for Excalibur devices.

The embedded stripe performance designs allow you to determine the feasibility of using an Excalibur device. They also provide a perspective on how to tune burst lengths to increase system bandwidth.



Refer to [“Revision History” on page 10](#) to see changes made for this version of the document.

Related Documents

You should read the following related documents:

- [Wrapper Latency White Paper](#)
- [AN142: Using the Embedded Stripe Bridges](#)
- [AN181: Multi-Master Reference Design](#)

Embedded Stripe Performance



The embedded stripe performance designs are Verilog HDL implementations that model the following functions:

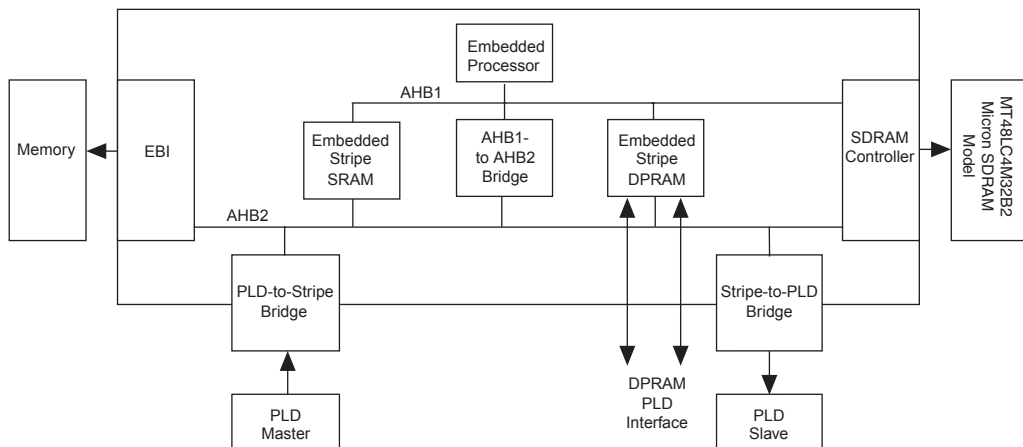
- A PLD master initiating a transaction to slaves in the embedded stripe
- The embedded processor initiating a transaction to slaves in the embedded stripe and the PLD

The two designs are very similar; the only difference is whether the PLD master or embedded processor initiates the transaction.

[Figure 1 on page 2](#) shows a block diagram of the embedded stripe design.

Design Overview

Figure 1. Embedded Stripe Performance Design



The DPRAM interface is not used in the embedded stripe design shown in [Figure 1](#).

Installation

This section details the software requirements and the directory structure.

Software Requirements

The following software is required to implement the embedded stripe performance designs:

- The Quartus® II software, version 2.2
- ARM Developer Suite for Altera (Altera ADS-Lite) software, version 1.1, or Redhat GNUPro toolkit
- Model Technology™ ModelSim® software, version 5.6d

Directory Structure

There are multiple projects in the **an192.zip** file.

To install the embedded stripe performance designs, unzip **an192.zip** into the installation directory of your choice.

After the installation, the directory structure is as follows:

```

<Installation Directory>\stripe_performance
    \pld_master
        \Ads
            \simulation
                \Modelsim
            \software
        \Gnu
            \simulation
                \Modelsim
            \software
        \rtl
        \testbench
    \microprocessor
        \Ads
            \simulation
                \Modelsim
            \software
        \Gnu
            \simulation
                \Modelsim
            \software
        \rtl
        \testbench
  
```

The top-level directories for the two designs are **\pld_master** and **\microprocessor**, but the subordinate directory structure for each is identical. Within each of the top level directories there are two separate Quartus II projects. The **\Ads** directory contains a Quartus II project with the ADS tool chain settings and the **\Gnu** directory contains a Quartus II project with the GNU tool chain settings.



You can ignore or delete the projects for the software tool chain that is not in use.

Depending on the software tool chain that is used, the **\Ads** or the **\Gnu** directory will be the working Quartus II project directory. The Quartus II project directory contains Quartus II project settings files, MegaWizard® Plug-In generated files, and a batch file that is used to produce simulation initialization files (**sbd2sim_asm.bat**).

Tables 1 to 5 detail the files in the directory structure for the two designs.

Table 1. File in the <Installation Directory>\stripe_performance\PLD_master or microprocessor\<Ads or Gnu>\software	
File	Description
Performance.s	Assembly file that is used to for embedded stripe register setup in both cases and execute transactions in uP case. The assembly files for ads and gnu differ

Table 2. Files in the <Installation Directory>\stripe_performance\PLD_master or microprocessor\rtl	
File	Description
Ahb_include.v	Include file that contains a definition of parameters for the design. In the case of the PLD master it is also used to change the address, data and burst length of transactions
bfm_master.v	Master port of the BFM
Burst_slave.v	PLD slave that accepts transaction from both the PLD master and the uP
Mt48lc4m32b2.v	128 Myte x 32 Micron SDRAM model. www.micron.com
Regfile.v	Simple memory model that is attached to the EBI
Slaveregfile.v	Register file for PLD slave
Stripe_performance.v	Top-level design file

The contents of the .\simulation\Modelsim directory of the two designs differ slightly. The contents are summarized in Tables 3 and 4.

Table 3. Files in the <Installation Directory>\stripe_performance\microprocessor\<Ads or Gnu>\simulation\Modelsim	
File	Description
Compile_and_run_rtl_fullmodel_v.do	Compiles and runs all the files necessary for <i>functional</i> simulation
Wave.do	Modelsim waveform file
Modelsim.mpf	Modelsim project file

Table 4. Files in the the <Installation Directory>\stripe_performance\PLD_master\<Ads or Gnu>\simulation\Modelsim (Part 1 of 2)	
File	Description
Compile_and_run_rtl_fullmodel_v.do	Compiles and runs all the files necessary for <i>functional</i> simulation
Wave.do	Modelsim waveform file

Table 4. Files in the the <Installation Directory>\stripe_performance\PLD_master<Ads or Gnu>\simulation\Modelsim (Part 2 of 2)

File	Description
Modelsim.mpf	Modelsim project file
Slavememory.cfg.dat	BFM required file. Not used in this application
Input_dpram.dat	Input stimulus file that targets DPRAM
Input_ebi.dat	Input stimulus file that targets the EBI
Input_pld_slave.dat	Input stimulus file that targets the slave in the pld
Input_sdram.dat	Input stimulus file that targets the SDRAM controller
Input_sram.dat	Input stimulus file that targets SRAM

Table 6 identifies the file in the .\stripeperformance\testbench directory.

Table 5. File in the <Installation Directory>\stripe_performance\testbench

File	Description
Stripeperformance_tb.v	Design testbench

Embedded Stripe Performance Designs

The embedded stripe performance designs are Verilog HDL implementations that model a PLD master and the embedded processor. The designs communicate to various slaves in the embedded stripe and a slave in the PLD. The designs take advantage of the fact that the embedded stripe model is a cycle-accurate model so that cycles can be counted to determine performance.

The embedded processor and the PLD master move a total of 64 words of data to different slaves in the design. The burst length of the transactions is varied to measure their effect on sustained throughput. One IDLE cycle is inserted after each burst. For example, two of the burst-length options for the PLD master are to initiate one burst of length 64 or 16 bursts of length 4 with one IDLE cycle between each burst.

The purpose of the one IDLE cycle is to allow time for the buffers involved in the transactions to clear. Depending on the peripherals involved in the transactions, inserting IDLE cycles can increase throughput. The embedded stripe performance designs allow you to vary the burst length and IDLE cycles between bursts, to observe their effects on throughput. See “Changing Designs” on page 9 for details.

Table 6 shows the clock speeds for the designs.

Table 6. Embedded Stripe Performance Clock Speeds		
Clock	Frequency (MHz)	Description
CLK_REF	50	Stripe reference clock
MASTER_HCLK	100	Stripe-to-PLD bridge clock
SLAVE_HCLK	100	PLD-to-stripe bridge clock
SDRAM_CLK	100	SDRAM clock
EBI_CLK	100	External EBI clock
AHB1_CLK	200	AHB1 clock
AHB2_CLK	100	AHB2 clock

Embedded Stripe Performance Processor Design

In the embedded stripe performance processor design, the embedded processor is a master initiating transactions to the following locations:

- Embedded processor → SRAM
- Embedded processor → dual-port SRAM
- Embedded processor → SDRAM (SDR running at 100 MHz)
- AHB1-2 Bridge → EBI
- Embedded processor → AHB1-2 bridge → stripe-to-PLD bridge → PLD slave

Before the embedded processor starts to initiate transactions, the code executing the transactions is locked into the cache for fast access and the SDRAM controller is initialized. The transactions are simple in-line load and store operations. One slave is targeted per simulation run, and a different slave can be targeted by changing the MOD_UNDERTEST variable in the embedded software. The code below declares the MOD_UNDERTEST variable. It is located in **performance.s** file in the **./software** directory. Only one line should be un-commented per simulation run. [Figure 2 on page 7](#) shows the ADS version of the declaration; [Figure 3](#) shows the GNU version.

Figure 2. ADS Version of MOD_UNDERTEST Declaration

```

;*****
;
;           Module undertest definition
;*****
;           Only uncomment one MOD_UNDERTEST definition
;*****
;
;MOD_UNDERTEST EQU      EXC_PLD_BLOCK0_BASE      ;;PLD Base
;MOD_UNDERTEST EQU      EXC_SDRAM_BLOCK0_BASE   ;;SDRAM Base
;MOD_UNDERTEST EQU      EXC_EBI_BLOCK0_BASE     ;;EBI Base
;MOD_UNDERTEST EQU      EXC_SPSRAM_BLOCK1_BASE  ;;SRAM1 Base
;MOD_UNDERTEST EQU      EXC_DPSRAM_BLOCK0_BASE  ;;DPRAM0 Base

```

Figure 3. GNU Version of MOD_UNDERTEST Declaration

```

/*-----
           Module undertest definition
-----

           Only uncomment one MOD_UNDERTEST definition
-----*/

//#define MOD_UNDERTEST      EXC_PLD_BLOCK0_BASE      //PLD Base
#define MOD_UNDERTEST      EXC_SDRAM_BLOCK0_BASE     //SDRAM Base
//#define MOD_UNDERTEST      EXC_EBI_BLOCK0_BASE     //EBI Base
//#define MOD_UNDERTEST      EXC_SPSRAM_BLOCK1_BASE  //SRAM1 Base
//#define MOD_UNDERTEST      EXC_DPSRAM_BLOCK0_BASE  //DPRAM0 Base

/*-----*/

```

Embedded Stripe Performance PLD Master Design

The embedded stripe performance PLD master design has a modified version of the bus functional model (BFM) attached to the PLD-to-stripe bridge. Input stimulus files are provided to simulate transactions to the following locations:

- BFM_Master → PLD-to-stripe bridge → SRAM
- BFM_Master → PLD-to-stripe bridge → DPRAM
- BFM_Master → PLD-to-stripe bridge → SDRAM (SDR running at 100 MHz)
- BFM_Master → PLD-to-stripe bridge → EBI



See the *Bus Functional Model User Guide* for details on the BFM and the input stimulus files.

Simulating Embedded Stripe Performance Designs

Before the BFM_Master starts to initiate transactions, the code executing the transactions is locked into the cache and the SDRAM controller is initialized. After SDRAM initialization, the embedded processor writes to a register in the PLD to enable the BFM_Master. When the BFM_Master has been enabled, the embedded processor is put into an endless loop, preventing it from affecting the performance of the BFM_Master.

To run a simulation of an embedded stripe performance design, perform the following steps:

1. Open the appropriate Quartus II project:
 - `<Installation Directory>\pld_master\ads` or `gnu\stripe_performance.quartus`
 - `<Installation Directory>\microprocessor>\ads` or `gnu\stripe_performance.quartus`
2. Set the toolset selection to the ADS standard tools or GNUPro for ARM.
3. Start a software build.
4. Open the appropriate ModelSim project:
 - `<Installation Directory>\pld_master\ads` or `gnu\simulation\Modelsim`
 - `<Installation Directory>\microprocessor>\ads` or `gnu\simulation\Modelsim`
5. Run the simulation script by typing the following command at the ModelSim prompt:

```
do Compile_and_run_rtl_fullmodel_v.do
```

In the **pld_master** simulation, a menu displays to allow you to select which slave you would like the BFM_Master to target. See [Figure 4 on page 9](#). Select a number from 1 to 5 to select the peripheral to be targeted.

Figure 4. Slave Targeting Dialogue

```

*****
"Please enter an integer between 1 to 5 to select the slave targeted by the
BFM_master"

1 BFM_master->PLD-to-Stripe Bridge -> SRAM
2 BFM_master->PLD-to-Stripe Bridge -> SDRAM
3 BFM_master->PLD-to-Stripe Bridge -> Stripe-to-PLD Bridge ->PLD Slave
4 BFM_master->PLD-to-Stripe Bridge -> DPRAM
5 BFM_master->PLD-to-Stripe Bridge -> EBI

*****

```

The simulation run terminates without further interaction.

7. Observe the waveform output.



For detailed information on the Excalibur device hardware design flow, refer to the *Excalibur Hardware Design Tutorial*.

Changing Designs

At present, the embedded stripe performance designs are configured to perform a specific set of transactions for evaluating performance. The embedded stripe performance designs are also very flexible and allow for other types of operations to be performed.

The real power of the designs is their flexibility—they can be used to resolve issues other than those relating to performance.

Changing the Embedded Stripe Performance Processor Design

The transactions in the embedded stripe performance processor design are initiated by embedded software, facilitating change. Any assembly command can be added to the current design, recompiled, and simulated. Changes to the memory map can be made by running the MegaWizard; no other changes to the design files are necessary. The only thing to note is that the simulation stops when the value 64 is loaded into register r10. The instruction that loads 64 into r10 is currently the last line of code executed in **performance.s**.

Changing the Embedded Stripe Performance PLD Master Design

The BFM emulates a master connected to the PLD-to-stripe bridge. It provides a flexible master that can be used to initiate transactions to any location in the memory map. By choosing a target slave using the menu shown in [Figure 4](#), a different input stimulus file (which is located in the `/simulation/modelsim` directory) is passed to the bus translator. To make changes to the design, change the appropriate input stimulus file (adhering to the bus functional language syntax) and select that file from the menu. If you change the memory map, you must change the memory locations in the input stimulus file to match.



Currently, the testbench waits for 64 to be valid on the read-data port of the BFM master. The transactions are set up with the last transaction occurring to a location to which 64 was written. If this portion of the design is changed, simulation must either be stopped manually, or the testbench must change to accommodate the changes.

Conclusion

The embedded stripe performance designs provide designers with a starting point for determining the feasibility of implementing their application in the Excalibur devices. The design is a Verilog HDL implementation that bursts 64 words of data to different slaves in the system. Changes are easily implemented by modifying embedded software or input stimulus files.

Revision History

[Table 7](#) shows the document revision history.

Date	Description
November 2002	Updated for Quartus II version 2.2.
July 2002	Modifications to accommodate GNUPro tools.
March 2002	First publication.



Notes:



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
<http://www.altera.com>
[Applications Hotline:](#)
(800) 800-EPLD
[Literature Services:](#)
lit_req@altera.com

Copyright © 2002 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



I.S. EN ISO 9001