SYNOPSYS®

DesignWare IBM PowerPC 405-S CPU Core

Design View coreKit User Guide

Database Release 1.1a
December 2006

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

"This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, and Vera are registered trademarks of Synopsys, Inc.

Trademarks (™)

Active Parasitics, AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BOA, BRT, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, DC Expert, DC Professional, DC Ultra, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, Direct RTL, Direct Silicon Access, Discovery, Dynamic-Macromodeling, Dynamic Model Switcher, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Formal Model Checker, FoundryModel, Frame Compiler, Galaxy, Gatran, HANEX, HDL Advisor, HDL Compiler, Hercules, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSIM^{plus}, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Milkyway, ModelSource, Module Compiler, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Power Compiler, PowerCODE, PowerGate, ProFPGA, ProGen, Prospector, Raphael, Raphael-NES, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, Softwire, Source-Level Design, Star-RCXT, Star-SimXT, Taurus, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS Express, VCSi, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (℠)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Preface

About This Manual

This manual describes how to work with the DesignWare IBM PowerPC 405-S CPU Core Design View coreKit. The DesignWare IBM PowerPC 405-S CPU Core is a Synopsys DesignWare coreKit implementation of the PPC405F5 CPU core from International Business Machines (IBM) Corporation. Throughout this manual, the term PowerPC 405-S CPU refers to the DesignWare IBM PowerPC 405-S CPU Core.

The Design View coreKit includes a simulation model and a timing model for the PowerPC 405-S CPU, and a testbench for functional simulation. An Implementation View coreKit, which includes synthesizable RTL, is available for implementation purposes.

This manual is intended for use by chip designers who are considering the DesignWare IBM PowerPC 405-S CPU Core for integration into a chip-level design. Readers are assumed to be familiar with HDL-based chip design methodologies and tools. This manual provides setup and simulation procedures for the DesignWare IBM PowerPC 405-S CPU Core Design View coreKit.

Related Documents

This manual is part of the DesignWare IBM PowerPC 405-S CPU Core document set. The complete documentation set resides in the doc directory of your coreConsultant workspace.

Manual Overview

This manual contains the following chapters and appendixes:

Preface	Describes the manual and lists the typographical conventions and symbols used in it; tells how to get technical assistance.
Chapter 1 "Introduction"	Provides a general description of the coreKits available for the DesignWare IBM PowerPC 405-S CPU Core and an introduction to the Synopsys coreConsultant tool.
Chapter 2 "Getting Started"	Describes how to install the coreKit, create a coreConsultant workspace, and install the PowerPC 405-S CPU VMC model.
Chapter 3 "Simulation – Basic"	Describes how to simulate the PowerPC 405-S CPU VMC model using the supplied testbench and test programs.
Chapter 4 "Integration"	Describes how to instantiate the PowerPC 405-S CPU VMC model in an application-specific test environment and provides information about the timing model.
Chapter 5 "Simulation – Advanced"	Provides information for modifying the testbench and simulating custom test programs.

Typographical and Symbol Conventions

The following conventions are used throughout this document:

Table 1: Documentation Conventions

Convention	Description and Example
%	Represents the UNIX prompt.
Bold	User input (text entered by the user). `% cd $LMC_HOME/hdl`
`Monospace`	System-generated text (prompts, messages, files, reports). `No Mismatches: 66 Vectors processed: 66 Possible"`
Italic or *Italic*	Variables for which you supply a specific value. As a command line example: `% setenv LMC_HOME prod_dir` In body text: In the previous example, *prod_dir* is the directory where your product must be installed.

Table 1: Documentation Conventions (Continued)

Convention	Description and Example
| (Vertical rule)	Choice among alternatives, as in the following syntax example: `-effort_level low | medium | high`
[ ] (Square brackets)	Enclose optional parameters: *`pin1 [pin2 ... pinN]`* In this example, you must enter at least one pin name (*pin1*), but others are optional ([*pin2* … *pinN*]).
TopMenu > SubMenu	Pulldown menu paths, such as: **File > Save As …**

Getting Help

For customer support:

- Logon to http://solvnet.synopsys.com using your Solvnet ID and password
  a. Under Support Resources, Click Enter a Call to the support center located at the right top corner
  b. Select Designware STAR IP from product entry
  c. Select Processors from sub product category
  d. Specify the subject and details of issues/errors
  e. Fill in all relevant details and then click Submit
- Send an e-mail message to support_center@synopsys.com.
- Telephone your local support center:
  - United States:

    Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific Time, Monday through Friday.
  - Canada:

    Call 1-650-584-4200 from 7 AM to 5:30 PM Pacific Time, Monday through Friday.
  - All other countries:

    Find other local support center telephone numbers at the following URL:

    http://www.synopsys.com/support/support_ctr

Additional Information

General information about Synopsys and its products is available at this URL:

http://www.synopsys.com

1 Introduction

The DesignWare IBM PowerPC 405-S CPU Core is a DesignWare version of IBM Corporation's PPC405F5 32-bit RISC CPU core.

Synopsys delivers the DesignWare IBM PowerPC 405-S CPU Core (referred to in this user guide as PowerPC 405-S CPU) as a DesignWare coreKit. Separate coreKits are available for evaluation (Design View coreKit) and implementation purposes (Implementation View coreKit).

This manual describes how to work with the Design View coreKit. If this is your first time working with a Design View coreKit, it is important to understand the following concepts before getting started:

PowerPC 405-S CPU coreKits

There are two coreKits available for the PowerPC 405-S CPU:

- Design View coreKit – Includes:
  - A cycle-accurate VMC model the PowerPC 405-S CPU
  - A timing model for an example PowerPC 405-S CPU implementation
  - An automated verification environment for functional simulation of the PowerPC 405-S CPU

The Design View coreKit is available to all DesignWare users to evaluate the PowerPC 405-S CPU for use in system-level applications.

- Implementation View coreKit – Includes synthesizable RTL source code for the PowerPC 405-S CPU, plus the same verification environment provided with the Design View coreKit. The Implementation View coreKit is available only to licensed users.

  To purchase a license for the Implementation View coreKit, send email to designware@synopsys.com.

This user guide contains user procedures for the Design View coreKit.

The *DesignWare IBM PowerPC 405-S CPU Core Implementation View coreKit User Guide*, included in the Implementation View coreKit, contains detailed procedures, guidelines, and examples for:

- RTL simulation in an automated verification environment
- Synthesis, including physical synthesis and clock tree synthesis
- Scan/ATPG
- Formal verification
- Power analysis
- Integration

Synopsys coreConsultant

coreConsultant is a Synopsys design reuse tool that automates the following design tasks for designs that have been packaged as coreKits:

- Installation
- Configuration (for configurable designs)
- Simulation using a verification environment that is packaged into the coreKit
- Synthesis by leveraging design knowledge packaged into the coreKit to directly drive Synopsys synthesis tools (Implementation View coreKits only)

coreConsultant provides a guided flow to execute the above tasks in a logical order. coreConsultant is available to all DesignWare users for use with DesignWare coreKits. Download instructions for the required version of coreConsultant are included in the email that was sent to you in response to your coreKit request.

The procedures in this user guide are of the following types:

- General coreConsultant procedures that are built into coreConsultant and are common to all coreKits. This user guide refers you to the *coreConsultant User Guide* for information that is common to coreConsultant and not unique to the PowerPC 405-S CPU coreKit. You can access the *coreConsultant User Guide* by selecting **Help > coreConsultant Tool Help > User's Guide** in the coreConsultant console.
- Design-specific coreConsultant procedures that are unique to the PowerPC 405-S CPU coreKit. The PowerPC 405-S CPU automated simulation procedures are examples of design-specific coreConsultant procedures, and are explained fully in this user guide.
- Command line procedures that you execute directly from a UNIX command line, and not through coreConsultant. For example, this user guide includes procedures to run simulations directly from a UNIX command line using scripts and utilities included in the coreKit.
- Information to integrate the PowerPC 405-S CPU VMC model into an application-specific testbench.

AMBA Support

The PowerPC 405-S CPU Design View coreKit includes a VMC simulation model for a PLB-to-AHB bidirectional bridge. You can use the PLB4XAHB bridge to connect the Processor Local Bus (PLB) to an AMBA high-performance bus (AHB).

The coreKit also includes an example verification environment that you can use both to test the bridge and as an example for integration. To use the bridge, follow the instructions in the *PLB4XAHB Bridge Application Note*, which is available in your <*workspace*>/PLB4XAHB/docs directory (filename: PLB4XAHB_application_note_<version>.pdf).

Note

Throughout this manual, <*workspace*> refers to the root directory of the coreConsultant workspace that you create using **File > New Workspace** menu in coreConsultant (for example, /usr/cores/ppc_405/config1). To reload your workspace after quitting coreConsultant, restart coreConsultant and specify the workspace name, for example: **`coreConsultant config1`**.

Typical Design Flow

Figure 1 shows a typical evaluation task flow using the Design View coreKit for evaluation of the PowerPC 405-S CPU.

Figure 2 shows a typical implementation task flow using the Implementation View coreKit to implement the PowerPC 405-S CPU in a system-level design.

In both diagrams, the shaded blocks indicate tasks that are automated through coreConsultant. For the non-shaded tasks, the coreKit includes guidelines and/or example scripts.

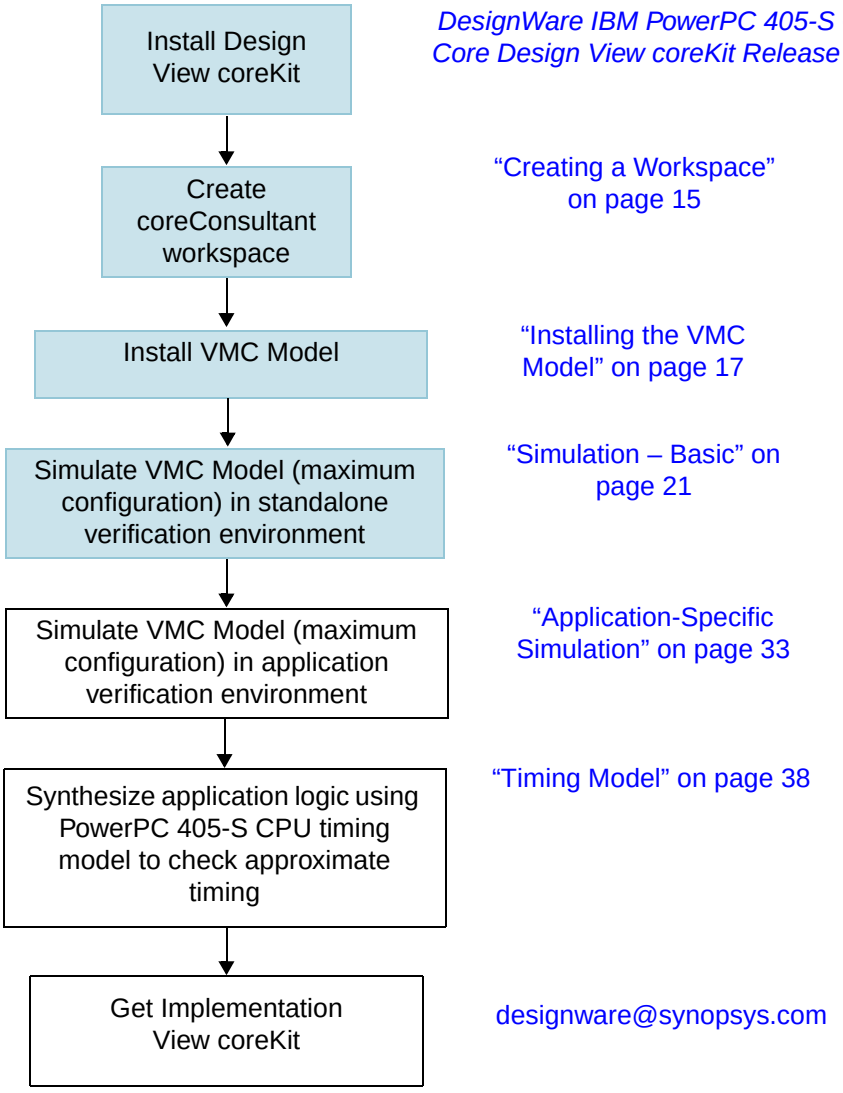


Figure 1: Typical Evaluation Flow with Design View coreKit

Finding Information

Both of the DesignWare IBM PowerPC 405-S CPU Core coreKits include complete documentation sets to support various hardware and software design tasks. The complete documentation set resides in the doc directory of your coreConsultant workspace.

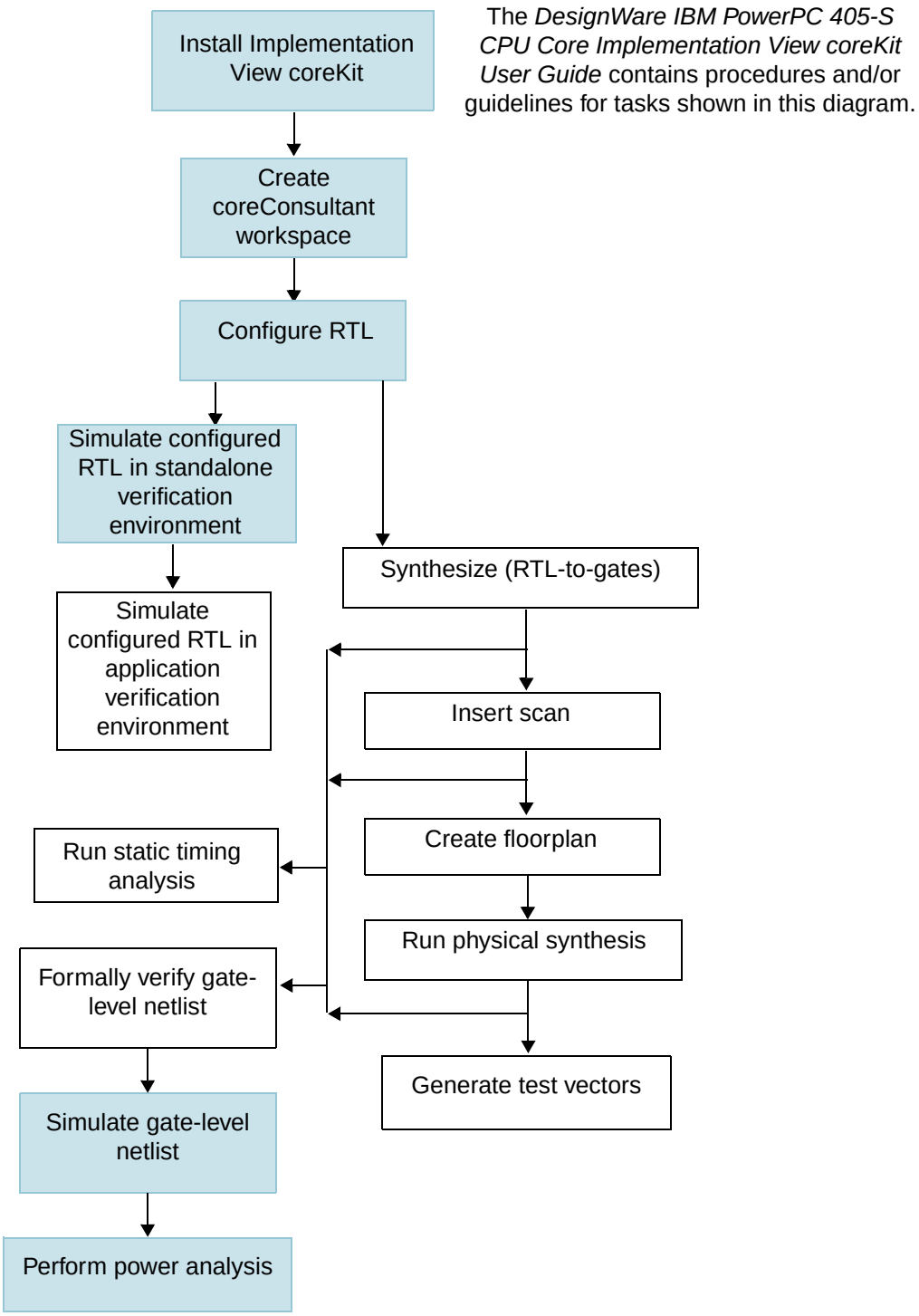


Figure 2: Typical Implementation Flow with Implementation View coreKit

2 Getting Started

This chapter gets you started working with the PowerPC 405-S CPU Design View coreKit. The topics are:

- Installing the coreKit
- Creating a Workspace
- Installing the VMC Model
- PowerPC 405-S CPU Configuration

Installing the coreKit

To install the Design View coreKit, follow the installation instructions that Synopsys sent to you in response to your coreKit request.

For additional information regarding supported tool versions, environment variable settings, new features, and known problems and limitations, refer to the *DesignWare DesignWare IBM PowerPC 405-S CPU Core Design View coreKit Release Note*.

Creating a Workspace

A coreConsultant workspace is a local, modifiable copy of your coreKit installation in which you work with the deliverables included in the coreKit. After you install the coreKit, you must create a workspace to begin working. You can create several workspaces so that you can experiment with different design alternatives, as shown in Figure 3.

To create a workspace:

1. Invoke coreConsultant:

```
% coreConsultant
```

2. Select **File > New Workspace** in the coreConsultant console, then enter the requested information in the New Workspace dialog. See the *coreConsultant User Guide* for details.

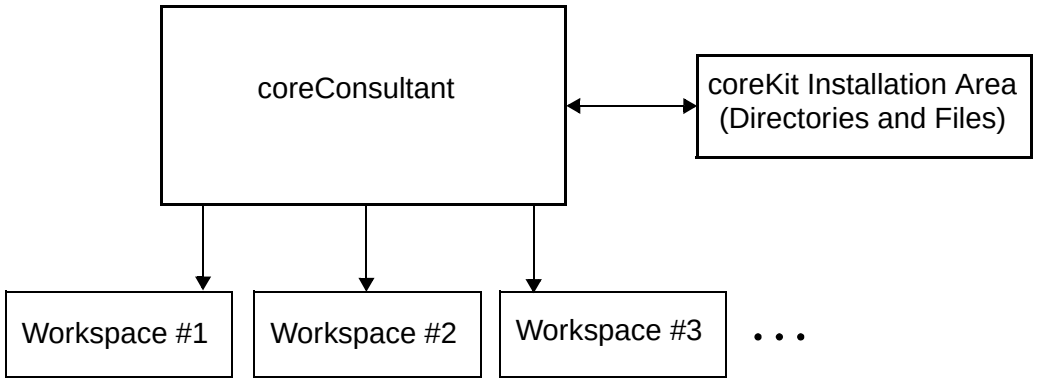


Figure 3: coreKit and Workspaces

At the completion of workspace creation, coreConsultant displays the Activity List. For the Design View coreKit (Figure 4), the first activity to execute is PowerPC 405-S VMC Model Installation. Go to "Installing the VMC Model" on page 17 to continue.

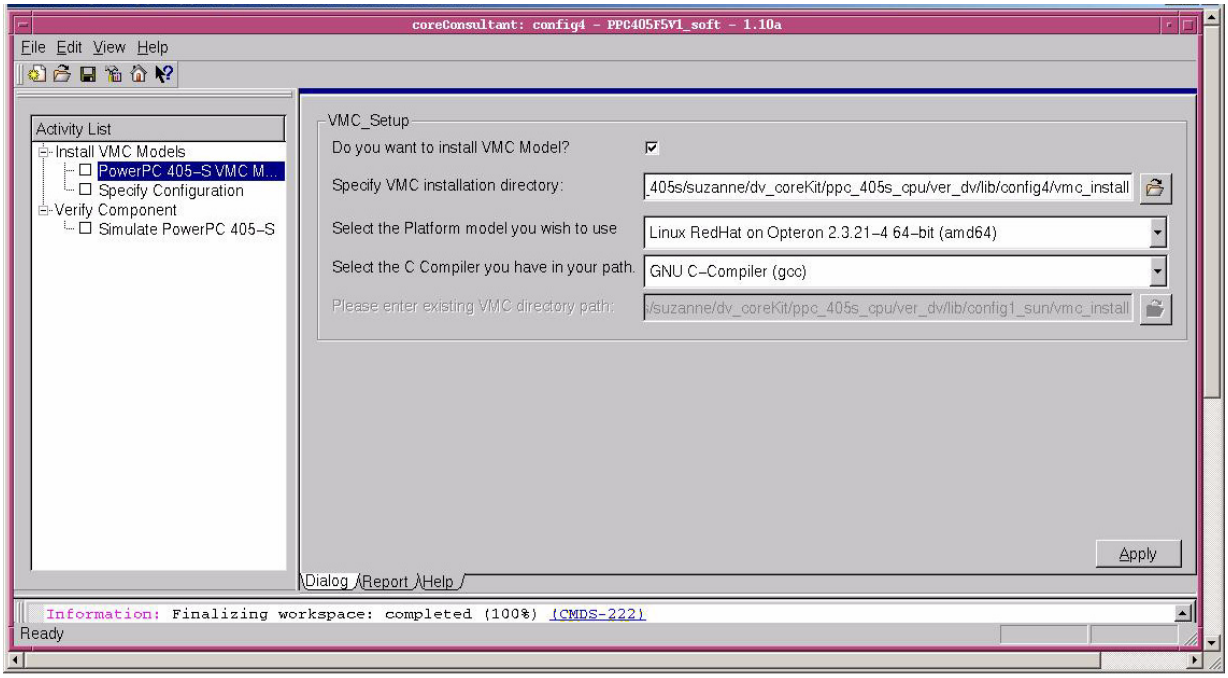


Figure 4: Design View coreKit VMC_Setup Activity

Installing the VMC Model

The Design View coreKit includes a cycle-accurate VMC model for the DesignWare IBM PowerPC 405-S CPU Core. The VMC model is a SWIFT R41-compliant model that is compatible with the most commonly used HDL simulation tools. By default, the testbench in the Design View coreKit instantiates the VMC model, encapsulated in the SWIFT template required for your simulator.

If you want to create more than one workspace, you can save disk space by installing the VMC model once for the first workspace, then pointing to that installation in additional workspaces.

When you install the VMC model for the PowerPC 405-S CPU, coreConsultant also installs the VMC models for two of the models used in the testbench:

- IBM PLB slave
- IBM PLB Nebula arbiter

The above models represent IP for PLB-compatible designs and are delivered as VMC models to protect that IP.

VMC Model Installation

To install the VMC models, go to the PowerPC 405 VMC Model page of the coreConsultant Activity List (Figure 4), then select the following installation options (see Table 2 for details):

- **Do you want to install VMC Model?** – Select this option to install the VMC models.
- **Specify VMC installation directory** – The absolute path to where you want to install the VMC models. Either enter the path name or use the [...] button to navigate and select and directory.
- **Select the Platform model you wish to use** – Select your platform from the list of supported platforms.

Note

coreConsultant must be currently invoked and operating on the platform you select.

- **Select the C Compiler you have in your path** – Choose the C compiler in your path.

After you select your installation options, click **Apply** to complete the VMC Model Installation activity. coreConsultant installs the VMC models for the platform you selected into your selected directory.

Note

The automated verification procedures automatically set LMC_HOME to point to your VMC model installation directory. It also automatically sets LD_LIBRARY_PATH to $LMC_HOME/lib/<platform>.lib. To use the VMC models in a different context, you need to explicitly set LMC_HOME to point to your VMC model installation directory. and LD_LIBRARY_PATH to the LMC_HOME shared libraries

Table 2: VMC Model Installation Options

Option	Definition
Do you want to install VMC Model?	Specifies whether to install the VMC models or use previously installed VMC models, and enables the corresponding option below.
Specify VMC installation directory	If you choose to install the VMC models, specify the directory into which you want to install them. The default path is <*workspace*>/vmc_install.
Select the Platform model you wish to use	Whether you are installing the VMC models or pointing to a previously installed version, select the platform you are using from the supported list.
Please enter existing VMC directory path	If you choose to point to previously installed VMC models, enter the path to the existing installation directory. For example, if you previously installed the VMC models into the vmc_install subdirectory of workspace config1, select <*config1_workspace_path*>/vmc_install for this option.

Pointing to an Existing VMC Model Installation

If you want to set up your workspace to point to previously installed VMC models instead of creating an additional installation:

- Go to the PowerPC 405 VMC Model page in the coreConsultant Activity List (Figure 4), then select the following installation options:
  - **Do you want to install VMC Model?** – Do not select this option.
  - **Select the Platform model you wish to use** – Select the platform from the list of supported platforms.

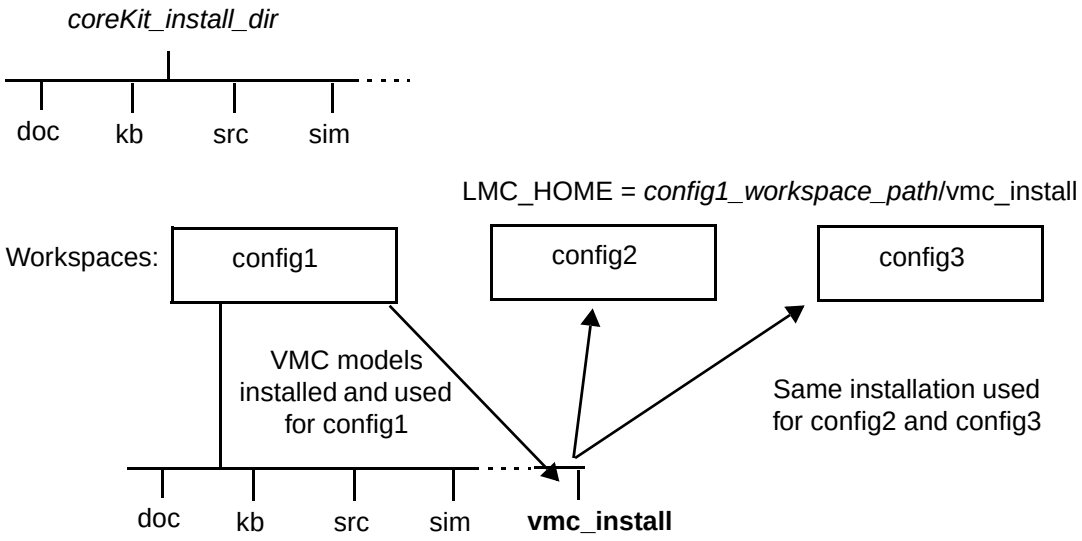
Note

coreConsultant must be currently invoked and operating on the platform you select.

- **Select the C Compiler you have in your path** – Select the compiler in your path.
- **Please enter existing VMC directory path** – The absolute path to the existing VMC model installation (for example, <*config1_workspace_path*>/vmc_install. Either enter the path name or use the [...] button to navigate and select a directory. If you set the LMC_HOME environment variable before invoking coreConsultant, this path will become the default.

After you select your installation options, click **Apply** to complete the VMC Model Installation activity. coreConsultant sets up your workspace to point to the selected VMC model installation directory.

Figure 5 shows an example setup where the user chooses to install the VMC models in the vmc_install subdirectory of the config1 workspace, then uses that installation for additional workspaces (config2 and config3).



NOTE: This example shows the VMC models installed in the vmc_install directory of config1. However, this is not a requirement. You can install the VMC models anywhere.

Figure 5: Using a Single VMC Model Installation for Multiple Workspaces

PowerPC 405-S CPU Configuration

The PowerPC 405-S CPU VMC model in the Design View coreKit is not configurable. The Specify Configuration step appears in the coreConsultant activity flow for the PowerPC 405-S CPU Design View coreKit, but does not allow you to change any configuration parameters.

3
Simulation – Basic

The PowerPC 405-S CPU Design View coreKit includes a functional verification environment that enables you to quickly evaluate the PowerPC 405-S CPU core by running the provided test programs. The verification environment is intended as an example platform for running simulation of the PowerPC 405-S CPU in a simple testbench, not as an extensive environment for full functional verification.

A similar verification environment is included in the Implementation View coreKit. In the Design View coreKit, the device under test is the PowerPC 405-S CPU VMC model; in the Implementation View coreKit, the device under test is the RTL source code for the PowerPC 405-S CPU.

This chapter describes how to perform automated simulation procedures using the supplied verification environment. The topics are:

For information about how to perform more advanced simulation tasks, such as customizing the testbench and executing custom test programs, refer to “Simulation – Advanced” on page 39.

Verification Environment Overview

The verification environment consists of:

- A testbench (Figure 6) that instantiates the PowerPC 405-S CPU VMC model as the device under test, plus the models listed in Table 3 attached to the PowerPC 405-S CPU interfaces.
- A set of test programs to verify the functionality of the PowerPC 405-S CPU. Each of the tests is self-checking and returns a pass or fail status.
- Scripts to automatically run the selected test programs on your HDL simulator.

You can run simulation either through the coreConsultant GUI or directly from the UNIX command line using the supplied scripts.

Simulation Methods

There are two ways to run simulations with the PowerPC 405-S CPU verification environment:

- Verification Through coreConsultant involves running simulations through the coreConsultant GUI. coreConsultant prompts you for simulation options, including which test programs you want to run, then automatically generates a script to run the simulation according to your selections. coreConsultant also invokes the run script, displays simulation status during the simulation, and displays the results when the simulation is complete.
- Command Line Verification involves executing the run script from the UNIX command line. Before you can run command line verification, you must run at least one simulation through coreConsultant to set up the verification environment in your coreConsultant workspace.

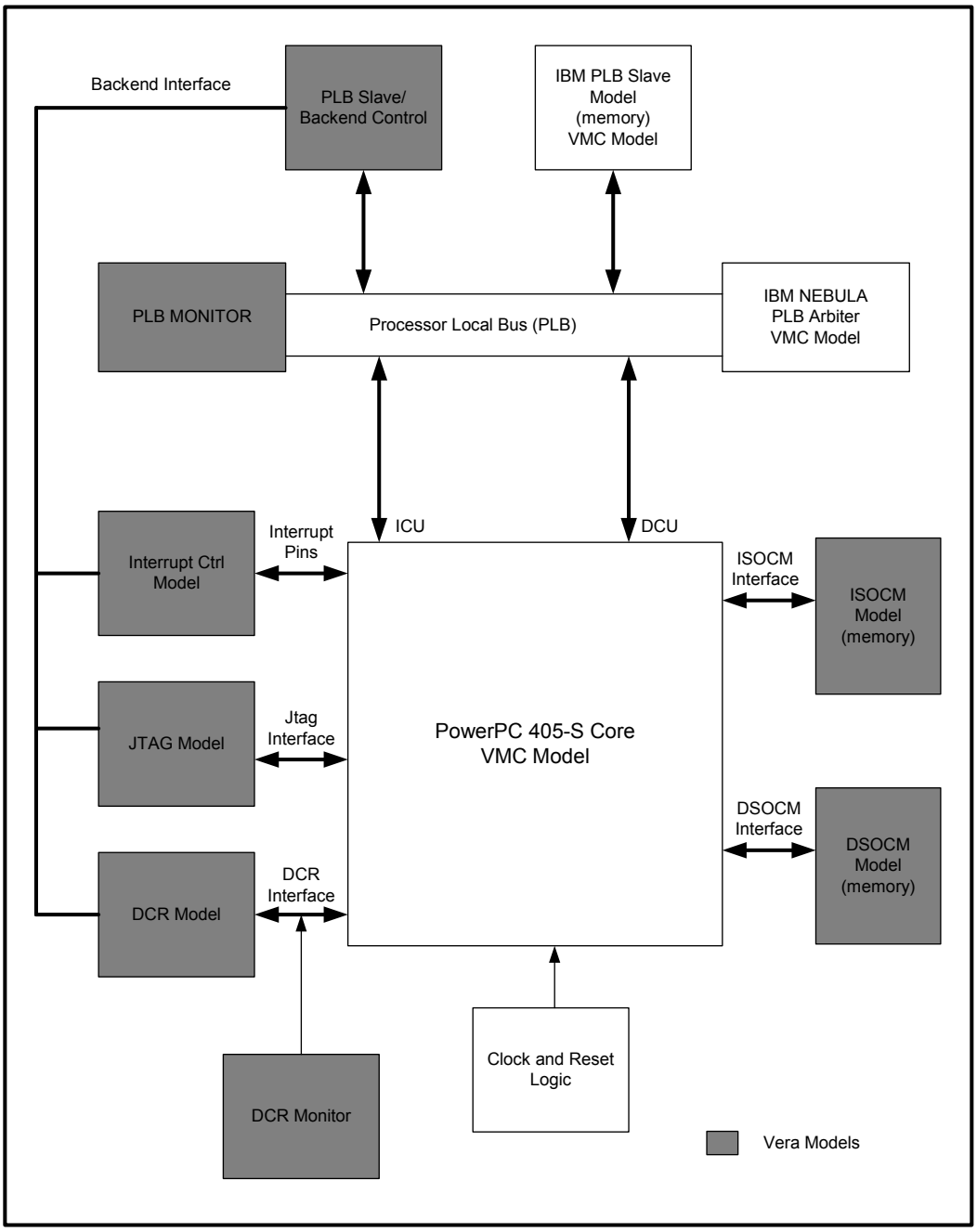


Figure 6: Testbench Block Diagram

Table 3: Models Used in the Testbench

Model	Format	Function
PLB slave/device controller	Vera	Controls the PLB monitor, interrupt controller model, JTAG model, and DCR model. Test programs running on the PowerPC 405-S CPU VMC model write to memory-mapped registers in the PLB slave/device controller to control the operation of the models connected through the control interface. The slave/device controller model does not support the complete set of PLB transactions. It only uses the PLB transactions needed to provided the necessary control functions.
IBM PLB slave model	VMC model	IBM PLB slave model that supports the complete set of PLB transactions, used for functional testing of PLB operation. The IBM PLB slave model also contains the instruction and data memory for the PowerPC 405-S CPU.
IBM PLB Nebula arbiter	VMC model	IBM PLB Nebula arbiter.
PLB monitor	Vera	Logs PLB transactions and detects PLB protocol errors.
Interrupt controller model	Vera	Generates interrupt signals to the PowerPC 405-S CPU, as controlled by the test program through the PLB slave/device controller model.
JTAG model	Vera	Stimulates the JTAG interface pins to the PowerPC 405-S CPU, as controlled by the test program through the PLB slave/device controller model.
DCR model	Vera	Provides a simple DCR model to exercise the **mfdcr** and **mtdcr** instructions.
DCR monitor	Vera	Logs transactions occurring on the DCR interface.
Clock logic	Verilog	Accepts SystemClock from top-level testbench and provides all necessary clocks to the PowerPC 405-S CPU. The clock logic also includes the logic to generate the clock enable signals used for sleep mode control.
Reset logic	Vera	Logic to drive the reset signals to the PowerPC 405-S CPU.

Table 3: Models Used in the Testbench

Model	Format	Function
ISOCM Model	Vera	The Instruction Side On-Chip Memory (ISOCM) only supports single cycle mode.
DSOCM Model	Vera	The Data Side On-Chip Memory (DSOCM) only supports single cycle mode.

Verification Through coreConsultant

To run the simulation through coreConsultant, go to the Simulate PowerPC 405 page of the coreConsultant Activity List (Figure 8), then select simulation options and execute the simulation as described in the following sections.

64-Bit Mode Execution

The PowerPC 405-S verification environment supports 64-bit mode execution of the supported simulators. To enable 64-bit mode:

- Make sure you have the appropriate environement variables and your path is pointing to your 64-bit simulator prior to running coreConsultant.
- In coreConsultant, select Tool Installation Roots from the Edit pull-down menu at top left.
- Check the “64-bit” box for your simulator and Vera.

See Figure 7 for details.

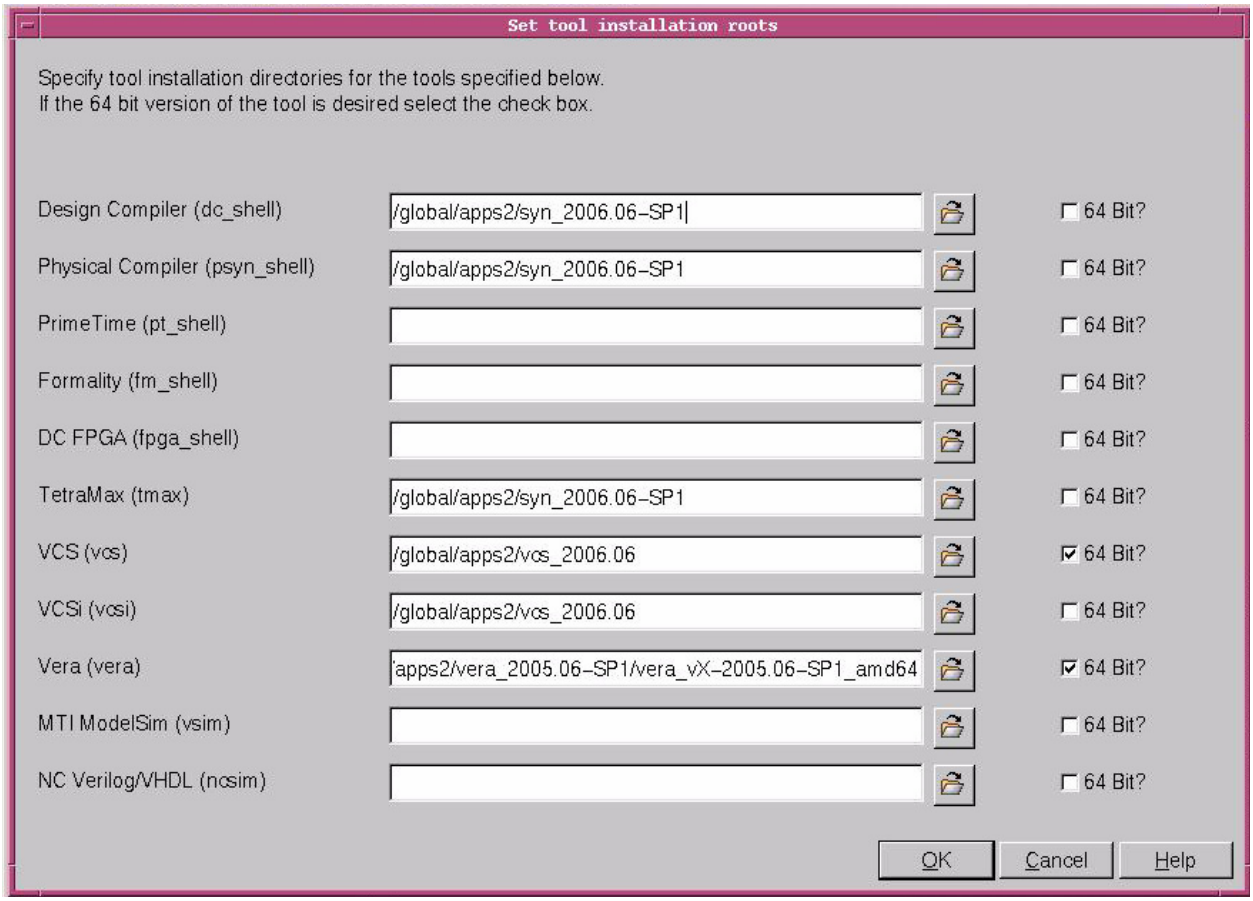


Figure 7: coreConsultant Tool Installation Roots Selection Page

General Simulation Options

The first two pages of simulation options are standard simulation options provided by coreConsultant:

- Simulator options – Options to select and set up your simulation tool. These options are common to several DesignWare coreKits. Use the coreConsultant online help for more information about individual options.
- Execution options – Options related to simulation launch and execution. These options are also common to several DesignWare coreKits. Use the coreConsultant online help for more information about individual options.

Note

Do not click **Apply** in the Simulate PowerPC 405 dialog (Figure 8) until you have confirmed your selections on all three pages of the dialog.

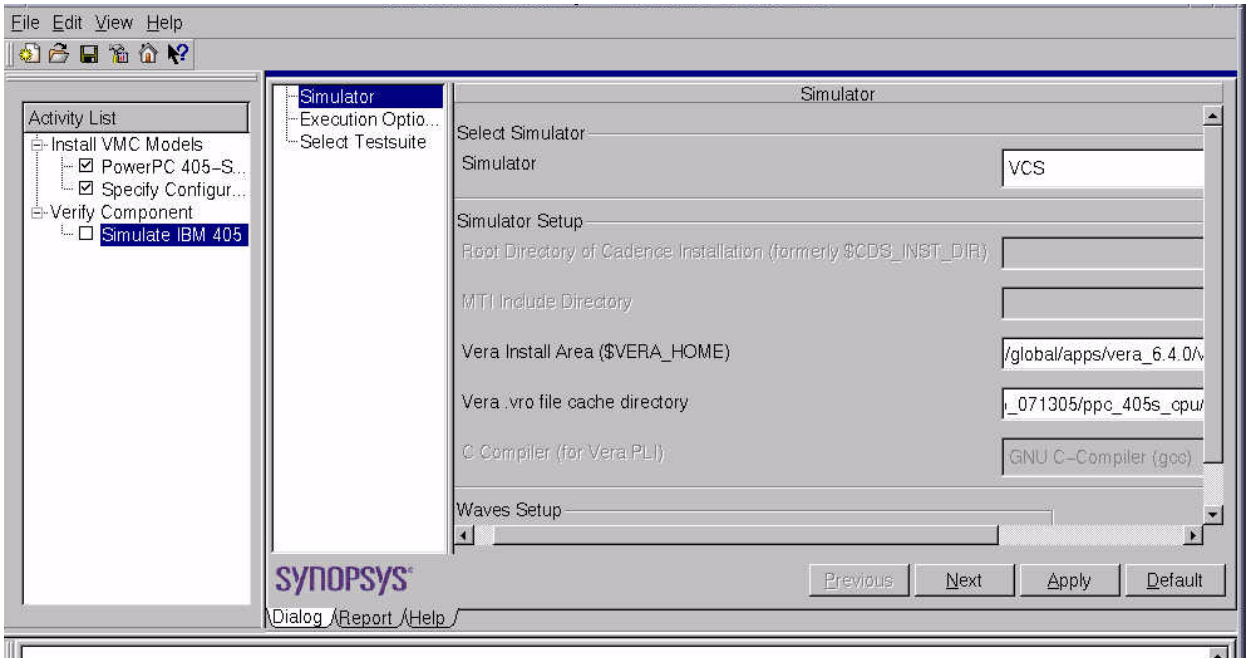


Figure 8: Simulate Dialog – Simulator Options

Test Selection Options

Using the coreConsultant test selection dialog (Figure 9), you can select to either run all of the provided test programs (the default selection) or just execute selected individual tests (**Run_User_Specified_Tests** option).

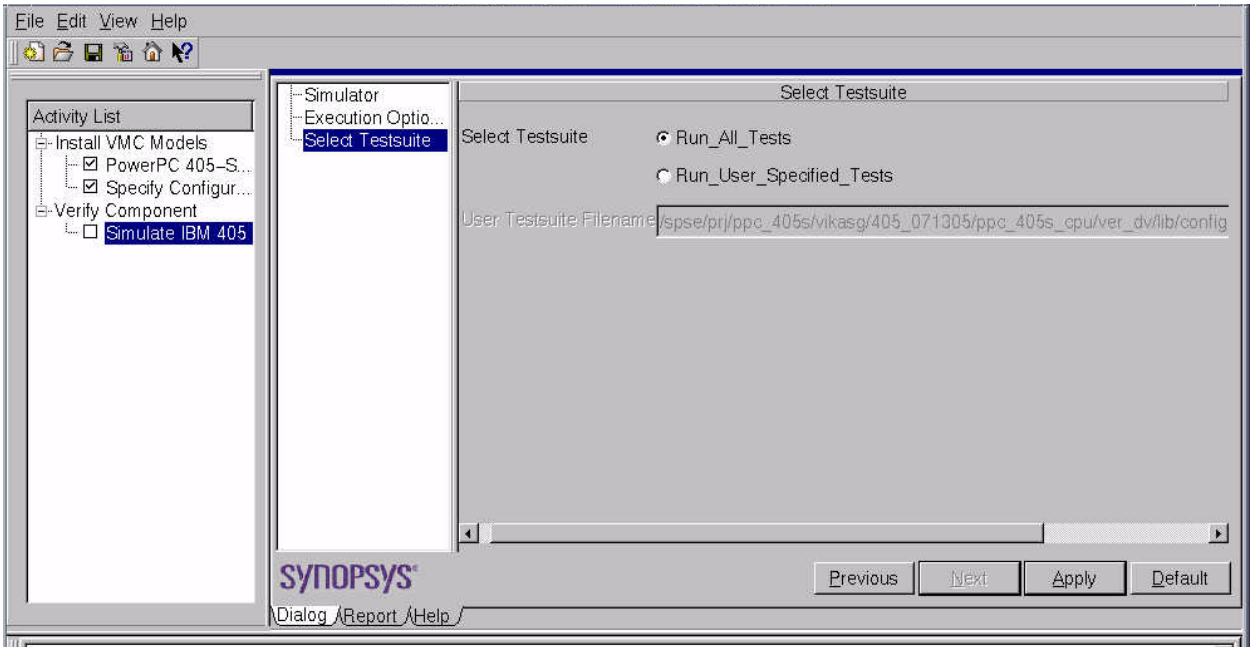


Figure 9: Simulate Dialog – Test Selection

The recommended strategy for test execution is:

1. Select and run a single test to make sure that your simulation is set up properly and the test can run without encountering any simulator setup errors.
2. Select and run all tests to verify that all tests pass.
3. If any test fails, contact support_center@synopsys.com. There is no known reason for any test to fail.

To select individual tests:

1. Make an editable copy of the supplied testlist file (<*workspace*>/sim/testcase.list). For example:

   ```
   % cd <workspace>/sim
   % cp testcase.list userlist
   ```

   You can choose any name and location for the editable test list file. The default user test list is <*workspace*>/sim/userlist, but you can specify a different path and file name.

2. Edit your copy of the test list file. Insert a semicolon (;) at the beginning of the line for each test that you do not want to execute, as shown in the following example:

```
fvt.exe.1
;fvt.exe.2
;fvt.icu.1
;fvt.icu.2
;fvt.ifb.1
...
```

   In the above example, the only test selected for execution is `fvt.exe.1`.

3. In the Testsuite Selection dialog (Figure 9), select **Run_User_Specified_Tests** and enter the file name (including full path) of your custom test list file. The default is <*workspace*>/sim/userlist.

Launching the Simulation

After you select your simulation options and tests, click **Apply** to launch the simulation. If you selected the **Do Not Launch Simulation** option, coreConsultant will not invoke the simulation; go to your <*workspace*>/sim directory and execute run.scr to run the simulation.

Checking Simulation Results

After you click **Apply** to start the simulation, the Simulate Report page appears (Figure 10) to display the status of the simulation job: running, done, or killed.

To update the status information in Figure 10, click the Reload ( ) button. When the simulation is complete, clicking the Reload button causes the simulation report (Figure 11) to appear below the status information.

Figure 11 shows an example of the simulation report. The simulation report includes pass/fail status for each test in the most recent simulation run.

The report also contains links to the simulation log for each completed test. See “Simulation Log Files” on page 30 for information about the contents of the log files.

Figure 10: Simulate Status Dialog

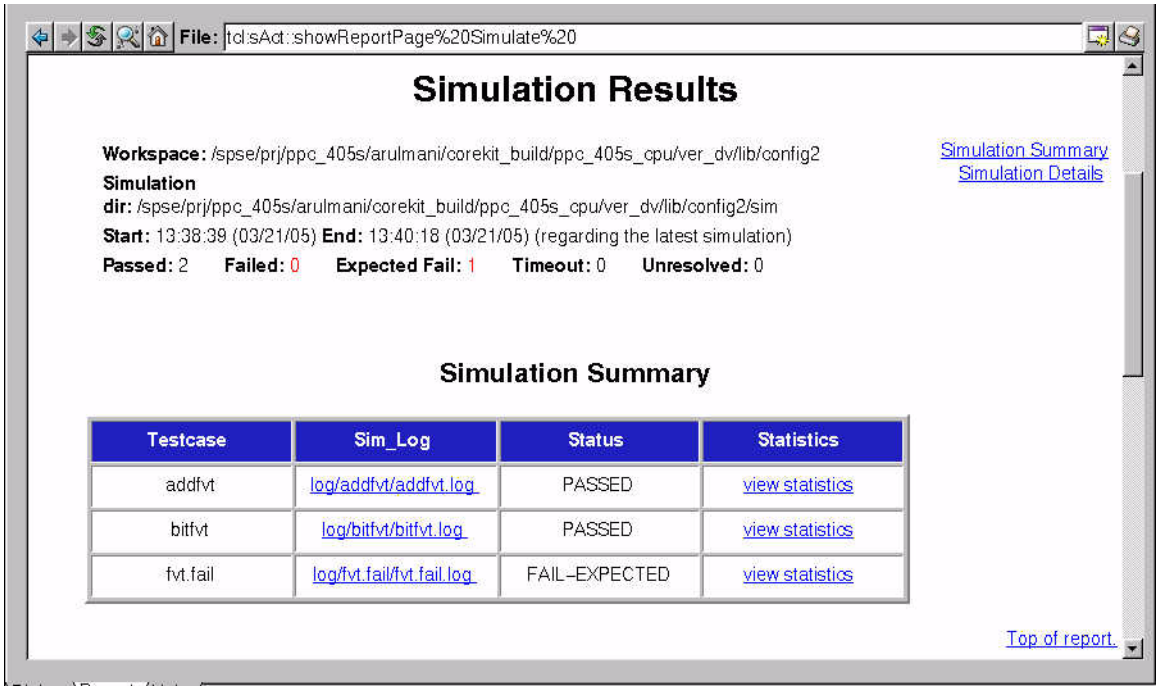


Figure 11: Simulate Report

Note

An fvt.fail testcase failure is expected.

Simulation Log Files

Simulation of one or more test programs, creates the following log files in your <*workspace*>/sim/log/<*testname*> directory, where <*testname*> is a test name as it appears in the test list file:

- <*workspace*>/sim/log/<*testname*>/<*testname*>.log – Simulation execution messages for <*testname*>. This is the log file that is linked to the simulation summary results (Figure 11). If a test fails, examine the <*testname*>.log file to determine the cause of the failure. If there is a problem invoking your selected simulation tool, the corresponding error messages will appear in <*testname*>.log.
- <*workspace*>/sim/test.log – Log file similar to <*testname*>.log, but includes log messages for all tests executed.

Command Line Verification

To run a simulation directly from the UNIX command line, perform the following steps:

1. Run at least one simulation through coreConsultant to set up your workspace for simulation, select tests, and generate a simulation run script (run.scr). Select the "Do Not Launch Simulations" option to generate run.scr without running the simulation. If you want to execute only selected tests, be sure to select the **Run_User_Specified_Tests** and enter the file name (including full path) of your custom test list file.
2. Run the simulation. To do so, go to your <*workspace*>/sim directory and execute the run script:

   ```
   % cd <workspace>/sim
   % ./run.scr
   ```

   The run.scr script performs the following tasks:

   - Invokes Vera to re-compile the testbench configuration files.
   - Invokes your selected simulator to compile the testbench files, generate a simulator executable, and run the simulation.
3. Check the simulation results by examining the following log files for each test that you selected:
   - <*workspace*>/sim/test.log – PASSED or FAILED status for <*testname*>.
   - <*workspace*>/sim/log/<*testcase_name*>/<*testcase_name*>.log – Detailed messages about test execution (see "Simulation Log Files" on page 30).

To change your selection of tests to be executed or any other simulation parameter, use coreConsultant to make your selections in GUI mode. Select **Do Not Launch Simulation**, and click **Apply**. coreConsultant generates a new run.scr, which you can then execute from the UNIX command line as described above.

If you had previously selected the **Run_User_Specified_Tests** option, you do not need to return to the GUI to change your test selection. Just edit the test list file and execute run.scr.

Viewing VMC Model Signals

When simulating the VMC model on a Verilog simulator, the contents of several internal registers and internal signals are available through VMC model windows. Table 4 lists the internal SFRs and GPRs.

The windows listed in Table 4 are enabled by default in the supplied testbench and are available for viewing if you create a dump file by choosing the **Generate 'waves' file** option in coreConsultant. To enable the VMC model windows for viewing in an application-specific simulation environment, see "Enabling VMC Model Register Windows" on page 37.

Table 4: VMC Model Windows – Internal Registers

Register	Description
dcdAddr	Decode Address
dcdData	Decode Data
exeFull	Instruction fetch full
exeAddr	Instruction fetch Address
exeAReg	Execution A Register
exeBReg	Execution B Register
exeResult	Execution Result
GPR0-GPR31	GPR Register 0-31
CR	Condition Register
CTR	Count Register
DAC1-DAC2	Data Address Compare 1-2
DVC1-DVC2	Data Value Compare 1-2
XER	Fixed Point Exception Register
DBCR0-DBCR1	Debug Control Register 0-1
DBSR	Debug Status Register
DBDR	Debug Data Register
DCCR	Data Cache Cachability Register
DEAR	Data Error Address Register
DCWR	Data Cache Write through Register
ICCR	Instruction Cache Cachability Register

Table 4: VMC Model Windows – Internal Registers

Register	Description
SLER	Stroage Little Endian Register
SU0R	Storage User-defined 0 Register
SGR	Storage Guarded Register
ZPR	Zone protection Register
PID	Process ID
CCR0-CCR1	Core Configuration Register
ICDBDR	Instruction Cache Debug Data Register
IAC1-4	Instruction Address Compare 1-4
LR	Link Register
EVPR	Exception Vector Prefix Register
SPRG0-7	SPR General 0-7
SRR0-3	Save/Restore Register 0-3
MSR	Machine State Register
PVR	Processor Version Register
ESR	Exception Syndrome Register
PIT	Programmable Interval Timer
TBH	Timer Base Higher
TBL	Timer Base Low
TSR	Timer Status Register
TCR	Timer Control Register

4 Integration

This chapter provides information and guidelines for the following integration-related topics:

- Application-Specific Simulation
- Timing Model

Application-Specific Simulation

Simulation of the PowerPC 405-S CPU VMC model in an application-specific verification environment involves:

- Instantiating the VMC Model
- Simulator Requirements for the VMC Model
- Enabling VMC Model Register Windows

Instantiating the VMC Model

To instantiate the VMC model in an application-specific testbench, you need to instantiate the HDL wrapper module defined in the SWIFT template for VMC model. Follow these steps to instantiate the VMC model:

1. Execute the Simulate VMC Model activity at least once to set up the verification environment. See “Verification Through coreConsultant” on page 25.
2. Instantiate the HDL module PPC405F5V1_soft in your application-specific testbench. Follow the example in the supplied testbench file (<*workspace*>/sim/testbench/p405s_test_top.v). In the Verilog testbench, the instance name for PPC405F5V1_soft is dut.

3. Include the SWIFT template file for the VMC model in your simulation file list:
   - For VCS, use <*workspace*>/src/vmc/p405s_vmcmodel_vcs.v
   - For NC-Verilog, use <*workspace*>/src/vmc/p405s_vmcmodel_nc.v
   - For MTI-Verilog, use <*workspace*>/src/vmc/p405s_vmcmodel_mti.v
4. Set the following environment variables before you run the simulation:
   - LMC_HOME – Point to the existing VMC model installation directory. For example:

     ```
     % setenv LMC_HOME <config1_workspace_path>/vmc_install
     ```
5. Include the simulator-specific variables and options as described in Simulator Requirements for the VMC Model below when you run your simulation.

Simulator Requirements for the VMC Model

Include the required LD_LIBRARY_PATH and options for the SWIFT PLIs when you execute your simulator-specific compile and run commands. Table 5 shows some examples for the supported simulators. For more complete information about how to use VMC models with your simulator, refer to:

- Your simulator documentation
- *Simulator Configuration Guide for Synopsys Models* (available at www.synopsys.com/products/designware/docs/doc/smartmodel/manuals/simcfg.pdf

Table 5: Simulator-Specific Command Switches for 32-Bit Sun Solaris

Simulator	Required Compile/Run Options
VCS	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/sun4Solaris.lib setenv LMC_USE_32BIT 1 **vcs command:** Include -lmc-swift
NC-Verilog	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/sun4Solaris.lib setenv LMC_USE_32BIT 1 **ncvlog command:** Include +loadpli1=swiftpli:swift_boot +incdir+$LMC_HOME/sim/pli/src+ncaccess+r+w
ModelSim Verilog	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/sun4Solaris.lib setenv LMC_USE_32BIT 1 **vlog command:** Include +incdir+$LMC_HOME/sim/pli/src **vsim command:** Include -pli $LMC_HOME/lib/sun4Solaris.lib/swiftpli.mti.so

Table 6: Simulator-Specific Command Switches for 64-Bit Sun Solaris

Simulator	Required Compile/Run Options
VCS	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/sparc64.lib **vcs command:** Include -lmc-swift -full64
NC-Verilog	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/sparc64.lib **ncvlog command:** Include +loadpli1=swiftpli:swift_boot +incdir+$LMC_HOME/sim/pli/src+ncaccess+r+w
ModelSim Verilog	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/sparc64.lib **vlog command:** Include +incdir+$LMC_HOME/sim/pli/src **vsim command:** Include -pli $LMC_HOME/lib/sparc64.lib/swiftpli.mti.so

Table 7: Simulator-Specific Command Switches for Red Hat Enterprise Linux v.3 Operating System for 64-Bit Opteron Processor

Simulator	Required Compile/Run Options
VCS	**SHLIB_PATH:** Include $LMC_HOME/lib/amd64.lib **vcs command:** Include -lmc-swift -full64
NC-Verilog	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/amd64.lib **ncvlog command:** Include +loadpli1=swiftpli:swift_boot +incdir+$LMC_HOME/sim/pli/src+ncaccess+r+w
ModelSim (Verilog)	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/amd64.lib **vlog command:** Include +incdir+$LMC_HOME/sim/pli/src **vsim command:** Include -pli $LMC_HOME/lib/amd64/swiftpli_mti.sl

Table 8: Simulator-Specific Command Switches for Red Hat Enterprise Linux v.3 Operating System for 32-bit Processors

Simulator	Required Compile/Run Options
VCS	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/linux.lib setenv LMC_USE_32BIT 1 **vcs command:** Include -lmc-swift
NC-Verilog	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/linux.lib setenv LMC_USE_32BIT 1 **ncvlog command:** Include +loadpli1=swiftpli:swift_boot +incdir+$LMC_HOME/sim/pli/src+ncaccess+r+w

Table 8: Simulator-Specific Command Switches for Red Hat Enterprise Linux v.3 Operating System for 32-bit Processors

Simulator	Required Compile/Run Options
ModelSim (Verilog)	**LD_LIBRARY_PATH:** Include $LMC_HOME/lib/linux.lib setenv LMC_USE_32BIT 1 **vlog command:** Include +incdir+$LMC_HOME/sim/pli/src **vsim command:** Include -pli $LMC_HOME/lib/linux.lib/swiftpli_mti.so

Enabling VMC Model Register Windows

As described in "Viewing VMC Model Signals" on page 30, there are several internal registers signals of the PowerPC 405-S CPU VMC model that are available for viewing through register windows if you are using a Verilog simulator.

To view VMC model register window signals, you must explicitly enable the register windows as required for your simulation tool. In the PowerPC 405-S CPU Design View coreKit, the register windows are enabled by default using the following mechanisms:

- For VCS, the top-level testbench file (<*workspace*>/sim/testbench/p405s_test_top_tb.v) includes the following line:

```
initial $swift_window_monitor_on("dut.PPC405D4")
```

  The above line globally enables the register windows for the dut instance of PPC405F5V1_soft in the testbench.

- For MTI-Verilog and NC-Verilog, each register window must be enabled individually. The simulator-specific SWIFT template file (<*workspace*>/src/vmc/p405s_vmcmodel_<*simulator*>.v) for each supported simulator includes a line in the following format for each register window:

```
$lm_monitor_vec_map(<register>, <instance_path_to_vmc_model>, "<window_signal>");
```

  For example:

```
...
initial
begin
$lm_monitor_vec_map(GPR0,dut.PPC405D4,"GPR0");
$lm_monitor_vec_map(GPR1,dut.PPC405D4,"GPR1");
$lm_monitor_vec_map(GPR2,dut.PPC405D4,"GPR2");
$lm_monitor_vec_map(GPR3,dut.PPC405D4,"GPR3");
$lm_monitor_vec_map(GPR4,dut.PPC405D4,"GPR4");
$lm_monitor_vec_map(GPR5,dut.PPC405D4,"GPR5");
$lm_monitor_vec_map(GPR6,dut.PPC405D4,"GPR6");
$lm_monitor_vec_map(GPR7,dut.PPC405D4,"GPR7");
$lm_monitor_vec_map(GPR8,dut.PPC405D4,"GPR8");
```

  $lm_monitor_vec_map(GPR9,dut.PPC405D4,"GPR9");

```
...
end
...
```

  To view the VMC model window signals, you must enable the register windows in a similar manner in your application-specific testbench.

Timing Model

The Design View coreKit also includes a Stamp timing model that you can use to derive approximate timing numbers for an example PowerPC 405-S CPU implementation. The timing model is in the stamp_model directory of your coreConsultant workspace for the Design View coreKit.

Note

The timing model provided with the Design View coreKit contains timing information only. It is not synthesizable and contains no functional logic.

The timing model files in <*workspace*>/stamp_model are:

- PPC405F5V1.lib - Library timing model
- PPC405F5V1_constr.pt - Timing exception file

5
Simulation – Advanced

This chapter provides additional information you can use for working with the testbench and simulation scripts, modifying the testbench, and running custom test programs. The topics are:

Testbench Details

The PowerPC 405-S CPU verification environment is intended as a starting point for a designer to evaluate, simulate, and start designing and writing software for a PowerPC 405-S CPU based system. The testbench for functional simulation (Figure 11) includes a VMC model for the PowerPC 405-S CPU core, functional models connected to the various interfaces of the PowerPC 405-S CPU, the IBM PLB Nebula arbiter and PLB slave model (both provided as VMC models), and a set of test programs that exercise different functional blocks and core interfaces. These test programs serve both as post-installation checks and as sample code for developing your own custom test programs.

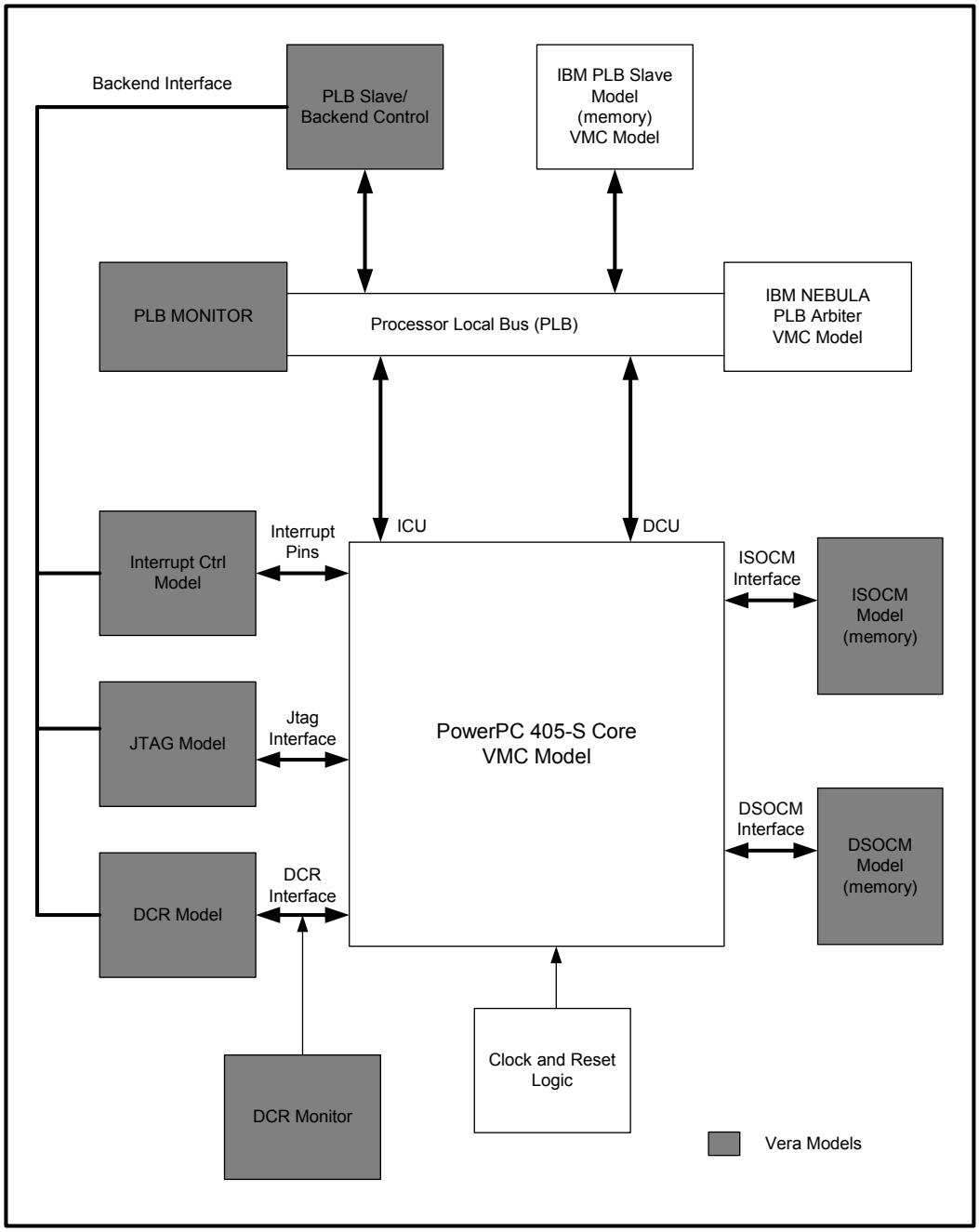


Figure 11: Testbench Block Diagram

Vera Models

The functional Vera models are memory mapped. Test programs control the Vera models by executing writes to the PLB slave/device control model, which, in turn, sends the appropriate commands to the Vera models through the control interface. The PLB slave model does not support the full set of PLB transactions that the PLB compliance checklist specifies. It only uses those transactions required to provide the necessary control functions for the Vera models.

Each Vera model has its own range of memory locations that are allocated as control registers. Controlling any model is then a matter of writing to specific locations in memory.

There are also two Vera blocks that serve as monitors: the DCR monitor and the PLB monitor. The DCR monitor checks transactions on the DCR bus and the PLB monitor checks transactions on the PLB.

The testbench execution flow logic resides partly in the PLB slave model and partly in the main Vera program for the testbench. The testbench execution flow logic performs the following functions:

- Parses test programs and loads them into the memory model.
- Starts the simulation by allowing the clock and reset logic to start.
- Stops the simulation when appropriate (test time-out, test pass, or test fail).

IBM PLB Models

The IBM PLB slave model and Nebula arbiter are included in the testbench as Verilog models. The Nebula arbiter supports arbitration between master PLB 3.X (64 bit PLB bus) and slave PLB4.X (128 bit PLB bus) devices. The IBM PLB slave model supports all transactions defined by the PLB compliance checklist and provides the memory model function for the instruction and data memory attached to the PLB. The IBM PLB slave model is configured to be at PLB address range 0x0000000000004000 to 0x00000000FFFFFFFF. I

Vera Model Details

The following sections provide more detail about the function of the Vera models used in the testbench.

PLB Slave Model

The PLB slave model connects to the IBM PLB Nebula arbiter as a PLB 4.X slave that controls the execution of the testbench and other Vera models. It is essentially a memory controller that controls a memory space that maps other Vera models through a control interface and a pass/fail mechanism.

The control interface to the Vera models communicates CPU commands to the Vera models (JTAG, DCR, EIC). This is a simple interface that is synchronous to the main clock and consists of a 10-bit address bus, read and write commands (strobes), and 32-bit data-in and data-out buses. The entire address space of this interface is 1024 bytes and can be configured to reside in any 1024-byte boundary of the PLB memory space.

By default, this space is initially mapped to the address range 0x000000000 – 0x0000003FF in the PLB memory space. This can be changed by editing the p405s_config.vrh configuration file (<*workspace*>/sim/vera/src/p405s_config.vrh) and changing the SLAVE_ADDR_HI and SLAVE_ADDR_LO definitions. This configuration file also contains the locations of the memory-mapped registers of the other Vera models. The default values are listed in Table 9.

Table 9: Default PLB Addresses for Vera Models

Model	Register	Default Address	Address Parameter Name
Interrupt Controller	Control Register	0x0B0	INTR_CTRL_REG
Interrupt Controller	Data Register	0x0B4	INTR_DATA_REG
DCR	Control Register	0x0F0	DCR_BD_ADDR
JTAG	Control Register	0x030	JTAG_CMD_REG
JTAG	Data Register	0x040	JTAG_DATA_REG

By default, the pass/fail result of any test is posted at offset 0x010 of this address space. Pass is indicated by the value 0xC00DF00D, and fail by the value 0xDEADBEEF. Care should be taken so that this region does not overlap (in address decoding) with any other region(s) in the address space required by the program.

Note that program code and data reside in the IBM PLB slave model, not the Vera PLB slave model.

PLB Monitor

The PLB monitor is connected to all the devices that are connected to the PLB Nebula arbiter. Its inputs are the PLB pins, both instruction-side and data-side PLB. Its single output is the *testbench_terminate* signal. This signal allows the monitor to request a simulation termination of the main Vera program. Whenever an activity occurs on the bus, the monitor writes pertinent information from that activity into the <*testname*>.log file. The monitor also checks for protocol violations on the master PLB 3.X bus and slave PLB 4.X bus. When a protocol error occurs, the monitor writes error information to an error log file and stops simulation execution.

The PLB monitor checks all transactions occurring on all devices connected to the arbiter. These devices are the ICU read, DCU read, DCU write (masters that are part of the PowerPC 405-S CPU), and the PLB slave models. When other devices are connected to the arbiter, the monitor will also check the transactions of those devices.

The PLB monitor is coded such that when you replace the PLB model with real logic, the monitor can still serve as a valuable debugging tool. The functionality of the PLB monitor is based on the IBM PLB monitor.

Instruction Side On Chip Memory(ISOCM) Model

The instruction side on-chip memory (ISOCM) model is connected to the ISOCM interface of PPC405-S core. The ISOCM model supports the ISOCM protocol specified in the “PPC405F5 Core Support Manual”. The ISOCM model only supports the Single Cycle mode.

When a request is presented across the interface, the ISOCM model will respond in the next clock cycle

If the address is not in the ISOCM address space, the request is not serviceable by ISOCM model and it will respond with default values.

If the address is in the ISOCM address space, the request is serviceable but the ISOCM model and it will asserts the appropriate signals.

The ISOCM model by default supports the 1KB boundaries. The ISOCM Vera model by default is configured to be at address range 0x0000_0400 to 0x0000_07FF.These default values can be changed by editing the p405s_config.vr configuration file and changing the ISOCM_ADDR_HI and ISOCM_ADDR_LO definitions.

Note: The Software must ensure the each instruction address has a single access path into the PPC405-S core for given software process.Each instruction address that is requested should be found in either the ISOCM address space or in instruction side PLB (ISPLB) space, but not in both.

Data Side On Chip Memory (DSOCM) Model

The Data side on-chip memory (DSOCM) model is connected to the DSOCM interface of PPC405-S core. The DSOCM model supports the DSOCM protocol specified in the “PPC4055 Core Support Manual”. The DSOCM model only supports the Single Cycle mode. Only two state pipeline is implemented in the DSOCM model for the Store Operations.

When a request is presented across the interface, the DSOCM model will responds in the next clock cycle

If the address is not in the DSOCM address space, the request is not serviceable by ISOCM model and it will respond with default values.

If the address is in the DSOCM address space, the request is serviceable by the ISOCM model and it will asserts the appropriate signals.

The DSOCM model by default supports the 1KB boundaries. The DSOCM Vera model by default is configured to be at address range 0x0000_0800 to 0x0000_0BFF.These default values can be changed by editing the p405s_config.vr configuration file and changing the DSOCM_ADDR_HI and DSOCM_ADDR_LO definitions.

Note: The Software must ensure the each instruction address has a single access path into the PPC405-S core for given software process.Each instruction address that is requested should be found in either the DSOCM address space or in data side PLB (DSPLB) space, but not in both.

Device Control Register (DCR) Model

The DCR model provides a simple DCR slave device connected to the DCR interface of the core. This model is intended to exercise the mfdcr and mtdcr instructions. Communication between this model and the PLB model is done through the “back-end” interface (described in “PLB Slave Model” on page 42).

The DCR model supports an address range specified by the base_addr_lo and base_addr_hi buses.

The DCR model has one programmable register that is located at the address specified by the value on the bd_base_addr bus. This register has two fields that are a single bit wide. The layout of this register is shown in Figure 12:

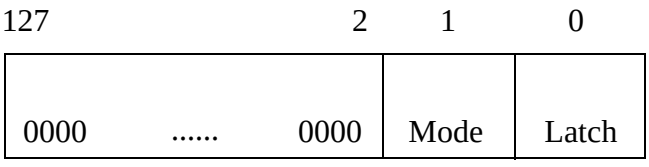


Figure 12: Layout of Programmable Register in DCR model

The following table decodes the two fields:

Mode	Latch	
0	0	Mode 0, Combinational ACK (default value)
0	1	Mode 0, Latched ACK
1	X	Mode 1, (The latch bit is ignored)

Multiple instances of the DCR are supported.

DCR Monitor

The DCR monitor attaches itself to the DCR bus. It looks at the transactions happening on the DCR bus and prints them onto the screen. It also does protocol checking. The DCR monitor is designed such that when the DCR model is replaced with real DCR slaves by the ASIC designer, the monitor can still serve as a debugging tool. Only one monitor is needed per DCR bus.

Interrupt Controller Model

The external interrupt controller (EIC) model connects to the EIC interface of the PowerPC 405-S CPU. Two memory locations are mapped to control the EIC model: the EIC control register and the EIC data register. Figure 13 shows the layout of the data register.

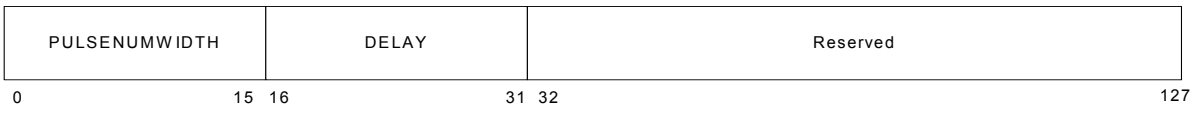


Figure 13: EIC Model Data Register

The sequence to control the EIC model is:

1. Write a value to the EIC data register.
2. Write to the EIC control register.

Table 10 defines write-pairs recognized by the EIC model.

Table 10: EIC Control-Data Write Pairs

Value written to EIC control register	Description of event after a subsequent write to EIC data
0x01	Trigger a critical interrupt after DELAY number of clock cycles. The interrupt will stay active until cleared by writing 0x03 to the EIC control register (see below). For example, to generate a critical interrupt pulse 100 clock cycles from now, set the DELAY to 100 (0x64), then write 0x01 to the control register.
0x02	Trigger a non-critical interrupt after DELAY number of clock cycles. The interrupt will stay active until cleared by writing 0x04 to the EIC control register (see below). For example, to generate a non-critical interrupt pulse 100 clock cycles from now, set the DELAY to 100 (0x64), then write 0x02 to the control register.
0x03	Clears the critical interrupt signal.
0x04	Clears the non-critical interrupt signal.
0x05	Simultaneously trigger both critical and non-critical interrupts after DELAY number of clock cycles. To clear them, write 0x03 and 0x04 to the control register.
0x06	Trigger a critical interrupt after DELAY number of clock cycles, followed by a non-critical interrupt PULSENUMWIDTH clock cycles after that.
0x07	Trigger a non-critical interrupt after DELAY number of clock cycles, followed by a critical interrupt PULSENUMWIDTH clock cycles after that.

JTAG Model

The JTAG implements a simple FSM that stimulates the JTAG signals to the PowerPC 405-S CPU to simultaneously shift data in (on TDI) and out (on TDO) of the core. The software view of this mechanism is a pair of registers, one for control, and the other for data (in/out). Data to be shifted into the core is an input to the JTAG model. Data shifted out of the core is stored in a local buffer in the model, and may be read at any time. To read this local buffer, write 0x04 to the JTAG model control register, and read the data register.

The sequence to control the JTAG model is: write a value to the JTAG data register, then write to the JTAG control register. Table 11 defines the various write-pairs recognized by the JTAG model.

Table 11: JTAG Model Control-Data Write Pairs

Value written to JTAG control register	Description of event
0x01	The value of the data register will determine the duration of the clock high. When it is 0 (default), the clock high duration is equal to the clock high duration of the SystemClock. When it is any other positive integer *n*, the clock high duration is *n* times the clock high duration of the SystemClock.
0x02	The value of the data register will determine the duration of the clock low. When it is 0 (default), the clock low duration is equal to the clock low duration of the SystemClock. When it is any other positive integer *n*, the clock low duration is *n* times the clock low duration of the SystemClock.
0x04	The value stored in the JTAG model's FIFO is copied into the data register (so that it can be read by software).
0x06	Initiates the shifting of data present in the JTAG data register into the TDI input of the PowerPC 405-S CPU, and shifting out of data from the TDO output of the PowerPC 405-S CPU into the JTAG model's FIFO. A clock is applied to the TCK input with clock high and low durations as defined by the clock high and clock low (see above entries on setting the clock high and clock low), for 33 clocks. After these 33 clocks, the data that was present in the JTAG data register should have been completely shifted through the chain and into the model's FIFO.

Clock, Reset, and Sleep Logic

The clock, reset, and sleep logic accepts a single clock, *SystemClock*, from the top-level Verilog testbench file, and provides the PowerPC 405-S CPU core with all the necessary clocks. These clocks include the *CPM_c405Clock*, the PLB clock, and the JTAG clock (*JTG_c405TCK*).

Also included is logic required for proper sleeping and waking of the PowerPC 405-S CPU. This logic computes the sleep request from the *C405_cpmMsrCE* and *C405_cpmMsrEE* outputs from PowerPC 405-S CPU, as well as interrupt inputs, timer reset request, and debug halt.

Simulation Directory Structure

Figure 14 shows the directory organization of the verification environment. Table 12 describes the contents and usage of the directories.

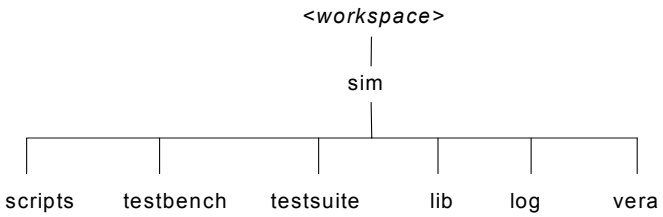


Figure 14: Simulation Directory Organization

Table 12: Simulation Directories

Directory	Usage
<*workspace*>/sim	Root directory of the test environment. Simulation scripts are invoked from <*workspace*>/sim.
<*workspace*>/sim/testbench	Holds testbench-related Verilog files, primarily the top-level testbench file (p405s_test_top_.v).
<*workspace*>/sim/testsuite	Holds all testcases provided in the test environment. Each testcase consists of an assembly source file (<*testname*>.s) and its assembled S-record formatted counterpart (<*testname*>.rec).
<*workspace*>/sim/log	Holds the log files generated by simulation runs. Under this directory, there is one directory for each testcase, in which the testcase executes. This is done to provide support for multiple simulations running simultaneously.
<*workspace*>/sim/vera	Soft link to the vera directory under the coreKit installation directory; contains all Vera related files. Vera source code for the Vera models is in sim/vera/src; compiled object files are in sim/vera/lib.
<*workspace*>/sim/scripts	Contains the necessary scripts to assemble tests, and to compile and run simulations. These scripts require perl5. See Simulation Scripts below for more detail about the individual scripts.

Simulation Scripts

To use the verification environment as delivered, run the simulations from coreConsultant or from the command line (using run.scr) as described in "Command Line Verification" on page 31. The following sections describe the scripts that exist in <*workspace*>/sim/scripts and are automatically invoked by run.scr. This information is useful if you want to modify and use the testbench for application-specific simulation purposes.

hexFormat script

The hexFormat.pl script is a Perl script that converts the test program S-record format file to a format that is understood by the IBM PLB slave model. This format includes two files: one file contains the addresses and the corresponding valid bits, and the other file contains the data, which is 128 bits wide. The IBM PLB slave model assumes the address-data pair by the line order. For example, if the second line of the address file contains the address 0x1000, then the second line of the data file will be assumed to be located at address 0x1000.

The hexFormat.pl script is automatically called by the runTB script prior to the loading of each testcase.

runTB script

Note

In the Design View coreKit, the runTB script depends on setup information generated by coreConsultant. Use <*workspace*>/sim/run.scr to run simulations in command line mode; do not execute runTB directly.

The runTB script compiles all necessary Verilog files, generates a simulator executable, and runs the simulation. The executable is produced by compiling all source files for the testbench and all the related testbench elements. The run405.config file contains setup information used by runTB, such as search paths and testbench file name.

The runTB script can accept the testcase name as the argument, or alternatively, a text file containing a list of testcases to execute. Table 13 describes all the command line options for runTB. The following example compiles all necessary files for VCS and executes testcase avs.sub.1.3.3:

```
% cd <workspace>/sim
% /scripts/runTB -sim vcs -t fvt.exe.1
```

Table 13: Command Line Options for runTB

Option	Definition
-comp	Compile all necessary files into an executable, but do not run the simulation.
-t <*testcase*>	Execute <*testcase*>.
-list <*filename*>	Execute all tests listed in <*filename*>.
-sim <*simulator*>	Specifies which simulator. For Verilog: vcs for VCS (the default), xl for Verilog-XL, mti for MTI-Verilog, or ncv for NC-Verilog.
-dump	Create hooks for waveform dump.

Assembling Test Programs

The provided test programs are delivered in both assembly source code and compiled S-record formats. There is no need to compile the provided test programs.

If you want to create your own test programs, you will need first install the required utilities:

- Assembler utility for PowerPC. Synopsys uses the binutils/2.13 GNU assembler utility, configured to target powerpc-elf. You can download the 2.13 GNU assembler from http://ftp.gnu.org/gnu/binutils. After expanding the tar file downloaded from this website, you can target the assembler for powerpc-elf as follows:

  a. Go to http://ftp.gnu.org/gnu/binutils and download binutils-2.13.tar.gz.

  b. Unpack the downloaded tar file.

  c. Go to the directory where you unpacked the tar file:

  ```
  > cd <path>/binutils-2.13
  ```

  d. Execute the following commands:

```
> ./configure --target=powerpc-elf --prefix=<my_compiler_installation_dir>
> make
> make install
```

  This will install the appropriate assembler, linker, and other utilities such as object dump, for the PowerPC architecture.

Note

To install the utilities into a common directory with restricted write privileges, you may need help from your system administrator.

Note

It has been verified that the installation of the PowerPC GNU assembler works with gcc version 2.95.2. Synopsys is not responsible for the proper function of GNU software.

- GNU make. You can obtain GNU make from the GNU website, http://www.gnu.org/
- Perl, Version 5.0.

After you install the required utilities and add the associated executables to your $path, use the following steps to compile your test program and make it available for simulation:

1. Place your test program source file in your <*workspace*>/sim/testsuite directory. Use .s as the filename extension (for example, mytest.s).
2. Edit the <*workspace*>/sim/testsuite/Makefile:
   a. Add your test program name (for example, mytest) to the `all : clean` section at the beginning of the Makefile. For example:

```
all : clean addfvt avs.dsocm.11.1.1 ...
       ...
       ...
subfvt syncfvt wrtfvt mytest
```

   b. Add the following lines for your test. You can copy, paste, and edit these lines from an existing test, as shown below:

```
wrtfvt : wrtfvt.o
        powerpc-elf-ld -d -T ${LINKMAP} -o $@.exe $@.o
        powerpc-elf-ld -d -T ${LINKMAP} --oformat=srec -o $@.rec $@.o
```

Copy, paste, and edit.

```
mytest : mytest.o
        powerpc-elf-ld -d -T ${LINKMAP} -o $@.exe $@.o
        powerpc-elf-ld -d -T ${LINKMAP} --oformat=srec -o $@.rec $@.o
```

These must be tabs, not spaces.

3. Run the make command from the <*workspace*>/sim/testsuite directory:

```
% make
```

The above command cleans up the testsuite directory and recompiles all of the test programs, including your new test program. You can now specify mytest to be executed by adding it to the testlist file as described in “Test Selection Options” on page 27.