PrimeTime®
Distributed Multi-Scenario Analysis
User Guide

Version D-2009.12, December 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2009 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

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, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (^{SM})

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.

Contents

Preface

This preface includes the following sections:

What's New in This Release

Information about new features, enhancements, and changes, along with known problems and limitations and resolved Synopsys Technical Action Requests (STARs), is available in the *PrimeTime Release Notes* in SolvNet.

To see the *PrimeTime Release Notes*,

1. Go to the Download Center on SolvNet located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.
2. Select PrimeTime Suite, then select a release in the list that appears at the bottom.

About This User Guide

The *PrimeTime Distributed Multi-Scenario Analysis User Guide* describes how to setup a distributed multi-scenario analysis flow, explains compute and license resource management, describes the commands and variables used during the flow, and also explains the merged reporting features.

Audience

This user guide is for design engineers who use PrimeTime for static timing analysis.

Related Publications

For additional information about PrimeTime, see Documentation on the Web, which is available through SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

You might also want to refer to the documentation for the following related Synopsys products:

- PrimeTime SI, PrimeTime PX, and PrimeTime VX
- Design Compiler
- Library Compiler

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as `pin1 [pin2 ... pinN]`
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to https://solvnet.synopsys.com (Synopsys user name and password required), and then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

Distributed Multi-Scenario Analysis

You can have PrimeTime analyze several scenarios in parallel. The ability to process multiple scenarios in parallel is called distributed multi-scenario analysis. This chapter describes the distributed multi-scenario analysis capabilities, including the flow, setup, saving and restoring your session, and reporting.

Introduction

Verifying a chip design typically requires several PrimeTime analysis runs to check correct operation under different operating conditions (such as extreme temperatures and operating voltages) and in different chip operating modes (for example, mission or test mode). A specific combination of operating condition and operating mode for a given chip design is called a scenario for that design. You can have PrimeTime analyze several scenarios in parallel, which is called distributed multi-scenario analysis.

You can have PrimeTime analyze several scenarios in parallel. The ability to process multiple scenarios in parallel is called distributed multi-scenario analysis.

Instead of analyzing each scenario separately, you can set up a master PrimeTime process that sets up, executes, and controls multiple slave processes, one for each scenario. You can distribute the processing of scenarios onto different hosts running in parallel, thus reducing the overall turnaround time to finish analysis of timing results from multiple scenarios. In addition, total runtime is reduced when you share common data between different scenarios.

A single script can control many scenarios, making it easier to set up and manage different sets of analyses. This capability does not use *multithreading* (breaking a single process into pieces that can be executed in parallel). Instead, it provides an efficient way to specify the analysis of different operating conditions and operating modes for a given design and to distribute those analyses onto different hosts. For descriptions of the distributed multi-scenario analysis commands, see Appendix A, “Master Command List.

Definition of Terms

The following terms are used to describe aspects of distributed processing:

baseline image

Image that is produced by combining the netlist image and the common data files for a scenario.

command focus

Current set of scenarios to which analysis commands are applied. The command focus can consist of all scenarios in the session or just a subset of those scenarios.

current image

Image that is automatically saved to disk when there are more scenarios than hosts and the slave process must switch to work on another scenario.

master

Manages the distribution of scenario analysis processes.

scenario

Specific combination of operating condition and operating mode.

session

Current set of scenarios selected for analysis.

slave

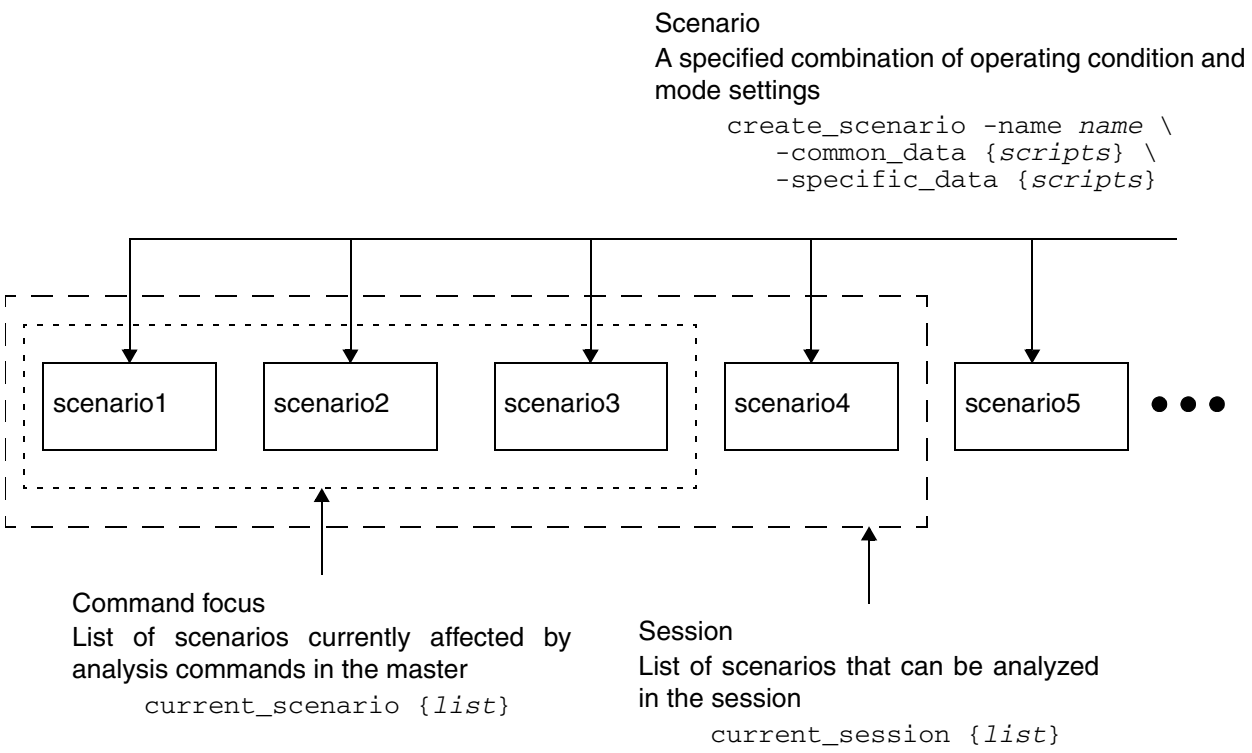
Started and controlled by the master to perform timing analysis for one scenario.

task

Self-contained piece of work defined by the master for a slave to execute.

Figure 1-1 shows the relationships between the configuration, scenarios, session, and command focus.

Figure 1-1 Configuration, Scenarios, Session, and Command Focus



A scenario describes the specific combination of operating condition and operating mode to use when analyzing the design specified by the configuration. There is no limit to the number of scenarios that you can create (subject to memory limitations of the master). Each scenario is created using the `create_scenario` command. The command specifies the scenario name and the names of the scripts that apply the analysis conditions and mode settings for the scenario.

The scripts are divided into two groups: common-data scripts and specific-data scripts. The common-data scripts are shared between two or more scenarios, whereas the specific-data scripts are specific to the particular scenario and are not shared. This grouping helps the master process manage tasks and to share information between different scenarios, minimizing the amount of duplicated effort for different scenarios.

A session describes the set of scenarios you want to analyze in parallel. You can select the scenarios for the current session using the `current_session` command. The current session can consist of all defined scenarios or just a subset of those scenarios.

The command focus is the set of scenarios affected by PrimeTime analysis commands entered at the PrimeTime prompt in the master process. The command focus can consist of all scenarios in the current session or just a subset of those scenarios. A subset of the scenarios in the current session can be specified using the `current_scenario` command. By default all scenarios in the current session are in command focus, unless you change them.

Multi-Scenario Flow Overview

The following sections describe the basics of the multi-scenario flow.

Before You Begin

Before you start your multi-scenario analysis, you must set the search path and create a .synopsys_pt.setup file.

Setting Your Search Path

In multi-scenario analysis, you can set the `search_path` variable only at the master. When reading in the search path, the master resolves all relative paths in the context of the master. The master then automatically sets the fully resolved search path at the slave. For example, you might launch the master in the /remote1/test/ms directory, and set the `search_path` variable with the following command:

```
set search_path ". .. ../scripts"
```

The master automatically sets the search path of the slave to the following:

```
/remote1/test/ms /remote1/test /remote1/ms/scripts
```

The recommended flow in multi-scenario analysis is to set the search path to specify the following:

- Location of all files for your scenarios and configurations

- Location of all Tcl scripts and netlist, SDF, library, and parasitics files to be read in a slave context

Note:
For best results, avoid using relative paths in slave context scripts.

.synopsys_pt.setup File

The master and slaves both source the same set of .synopys_pt.setup files in the following order:

1. PrimeTime install setup file at:

   install_dir/admin/setup/.synopsys_pt.setup

2. Setup file in your home directory

   ~/.synopsys_pt.setup

3. Setup file in the master launch directory at:

   $sh_launch_dir/.synopsys_pt.setup

You can use the `pt_shell_mode` variable to control whether commands are executed in the current pt_shell mode. You can set the value of this variable to `primetime`, `primetime_master`, or `primetime_slave`. For example,

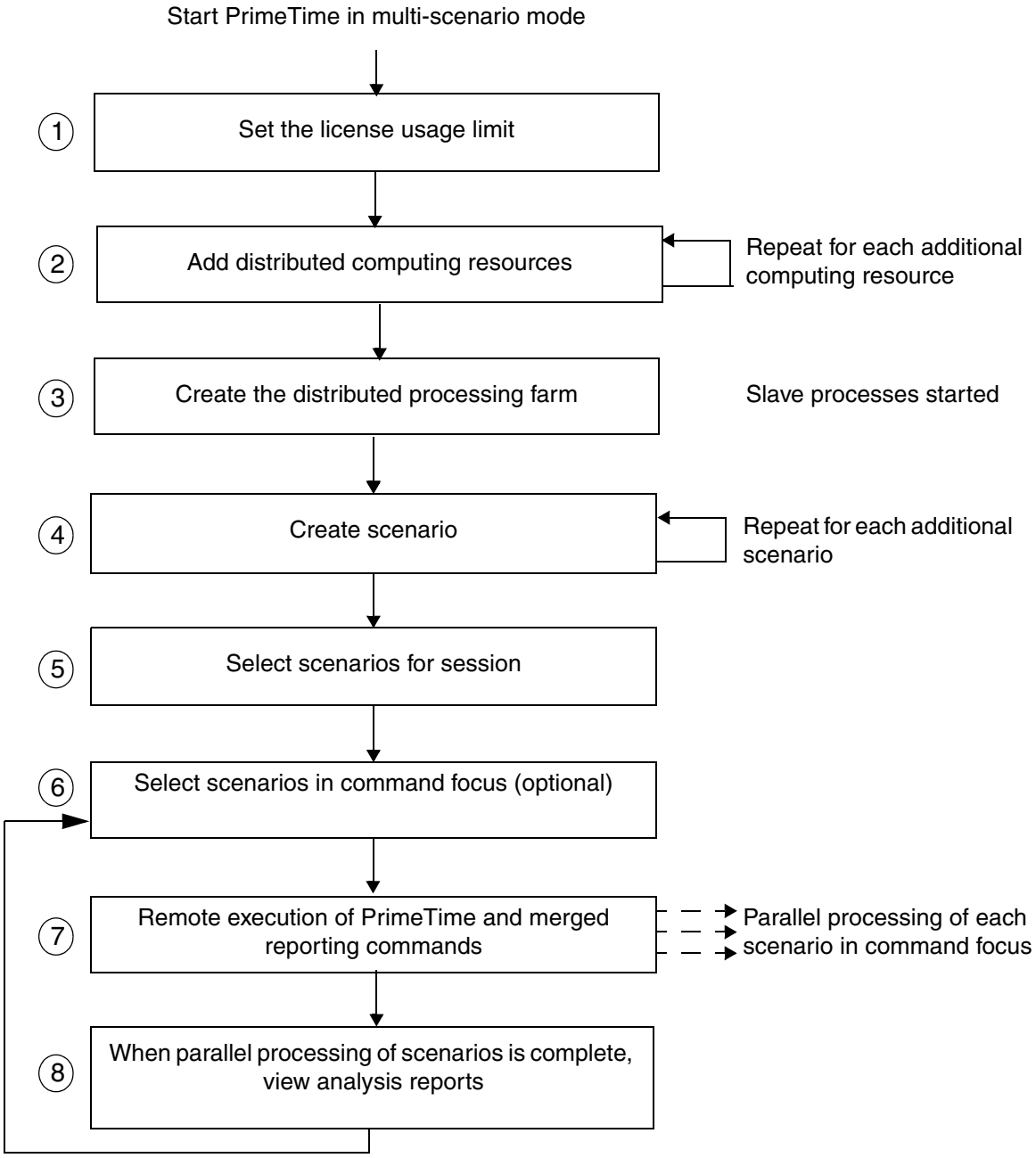
```
.synopsys_pt.setup

if { $pt_shell_mode == "primetime_master" } {
   set multi_scenario_working_directory "./work"
}
```

Usage Flow

To start PrimeTime in multi-scenario analysis mode, use the `pt_shell` command with the `-multi_scenario` option. From the pt_shell prompt of the master PrimeTime process, you can initiate and control up to 256 slave processes (see “Limitations” on page 1-50). Figure 1-2 shows the sequence of steps in a typical multi-scenario analysis.

Figure 1-2 Distributed Processing Usage Flow



A typical multi-scenario analysis has the following steps:

1. Specify the maximum number of PrimeTime or PrimeTime SI licenses for the slave processes to use by running the `set_multi_scenario_license_limit` command.

2. Add the slave hosts that are used as part of the distributed computing farm by running the `add_distributed_hosts` command. You can also add managed systems, such as LSF and GRD, to the distributed farm.
3. Optionally, you can configure the distributed environment by using the `set_distributed_parameters` command. For user configurations to take effect, you must issue the `set_distributed_parameters` command before the `create_distributed_farm` command. For more information, see the man page.
4. Create the distributed processing farm upon which the scenarios are executed by running the `create_distributed_farm` command.

   You can use the `create_distributed_farm` command to dynamically manage and allocate your resources.
5. Create the scenarios with the `create_scenario` command. Each `create_scenario` command specifies a scenario name and the PrimeTime script files that apply the conditions for that scenario.

   Note:
   You must complete steps 1 through 4 before you can go to the next step. Otherwise, the `current_session` command fails.
6. Select the scenarios for the session using the `current_session` command. The command specifies a list of scenarios previously created with the `create_scenario` command.
7. Optional—Change the scenarios in the current session that are in command focus, using the `current_scenario` command. The command specifies a list of scenarios previously selected with the `current_session` command. By default, all scenarios in the current session are in command focus.
8. Start processing in the scenarios by executing the `remote_execute` or `merged_reporting` command at the master.
9. When the processing of all scenarios by the slave processes is complete, you can view the analysis reports. You can find the reports generated by the `remote_execute` command under the directory you specify for the `multi_scenario_working_directory` variable, as described in “Distributed Processing Setup” on page 1-8. Alternatively, if you issue the `remote_execute` command with the `-verbose` option, all information is displayed directly to the console at the master. All merged reporting commands output are displayed directly to the console at the master.
10. Use ECO commands to fix any timing violations.

Note:

You can use the distributed multi-scenario capability not only for timing analysis in PrimeTime and PrimeTime SI, but also for power analysis in PrimeTime PX. This dramatically reduces the turn-around time. For more information about using distributed multi-scenario in PrimeTime PX, see the Distributed Peak Power Analysis section in the *PrimeTime PX User Guide*.

Distributed Processing Setup

For distributed processing of multiple scenarios, you must first invoke PrimeTime in distributed processing mode and define the scenarios. The scenarios are processed on a virtual distributed farm so you must specify the remote hosts to be part of the farm and create that distributed farm.

Invoking PrimeTime for Distributed Processing

To invoke PrimeTime in distributed processing mode (master), use the `-multi_scenario` option when you start pt_shell:

```
% pt_shell -multi_scenario
```

PrimeTime starts up and displays the pt_shell prompt just as in the single-core analysis mode. A multi-scenario analysis is carried out by one master PrimeTime process and multiple PrimeTime slave processes. The master and slave processes interact with each other using full-duplex network communications.

The master process generates analysis tasks and manages the distribution of those tasks to the slave processes. It does not perform timing analysis and does not hold any data other than the netlist and the multi-scenario data entered. The command set of the master is therefore restricted to commands for carrying out master functional capabilities. The following variable specifies the directory in which to store working files for distributed processing:

```
multi_scenario_working_directory
```

This variable should be set to the name of a directory that is write-accessible by you, the master process, and all slave processes. Typically, the slave processes are running on different hosts than the master, so the working directory needs to be network accessible, and therefore cannot be a local directory on the master. If you do not explicitly set this variable, the current working directory of the master is used.

Note:

You must set the `multi_scenario_working_directory` variable before creating the distributed farm.

The `multi_scenario_merged_error_log` variable specifies the file in which to write all error, warning, and information messages issued by the slaves. Any error, warning, or information message that is issued by more than one slave is merged into a single entry in the merged error log. Each entry contains the scenario name from which the data originated. If a file is specified for this variable, it appears in the directory specified by the `multi_scenario_working_directory` variable. For more information, see "Merged Error Log" on page 1-45. This variable limits the number of messages of a particular type written to the log on a per task basis. The default value is 100 (a maximum of 100 RC messages is written to the merged error log per task).

Note:

You must set the `multi_scenario_merged_error_log` variable before creating the distributed farm.

Upon startup, every PrimeTime process, both master and slave, looks for and executes the .synopsys_pt.setup startup file just as in normal mode. Each process also writes its own separate log file. For more information, see "Log Files" on page 1-44.

Compute Resource Management

You can dynamically manage your compute resources and allocate slave processes by adding hosts and creating farms. Managed computing resources allocate pooled computing resources as they become available. If an LSF, GRD or proprietary (generic) computing management capability is available for load balancing, you can allocate hosts from these systems to the virtual distributed farm used for processing scenarios.

The distributed processing mode supports the following hardware management architectures:

- LSF (Load Sharing Facility from Platform Computing, Inc.)
- GRD (Global Resource Director from Gridware, Inc.)
- Generic computing farm

Unmanaged computing resources are computed servers/workstations that are not load balanced, and are described as a Network Of Workstations. They can be added to the virtual distributed farm used for processing scenarios and are internally load balanced during multi-scenario analysis.

Adding Distributed Hosts

You can allocate slave hosts to be used in the virtual farm using the `add_distributed_hosts` command. For example, suppose you have the following computing resources available for use:

- 20 processor 64-bit LSF farm

- 20 processor 32-bit GRD farm
- 8 processor 64-bit server/workstation
- 2 processor 32-bit server/workstation

If you want to add them all to the virtual distributed processing farm for multi-scenario analysis, you can specify this using the following syntax:

```
pt_shell> add_distributed_hosts -num_of_hosts 20 \
         -farm lsf -setup_path "/bin/lsf/"

pt_shell> add_distributed_hosts -32bit -num_of_hosts 20 \
          -farm grd -setup_path "/bin/grd/"

pt_shell> add_distributed_hosts -num_of_hosts 8 \
          -farm now platinum1

pt_shell> add_distributed_hosts -32bit -num_of_hosts 2 \
          -farm now platinum2
```

Each slave process invoked by the master is an ordinary invocation of PrimeTime except that only the master process can directly interact with these processes. It is not possible to manually launch a PrimeTime process in slave mode.

When the master begins the distributed multi-scenario analysis on the slaves, it generates tasks for the slave processes to execute. Any one task can apply to only one scenario and any one slave process can be configured for only one scenario at a time. If there are more scenarios than slave processes to host them, some slave processes have to swap out one scenario for another so that all tasks get executed.

The process of swapping out one scenario for another is an expensive, time-consuming operation. It is also important, however, to avoid overloading hosts by specifying more processes on a host than there are available CPUs in the host. If more processes are specified than there are CPUs available, a penalty is incurred for swapping in and out slave processes on a CPU. Therefore, the slave processes do not run concurrently.

For optimal system performance of the distributed multi-scenario mechanism, match the number of slave processes, the number of scenarios in command focus. and the number of available CPUs. Optimal system performance is achieved when every scenario is executed in its own slave process on its own CPU.

Note:

The `add_distributed_hosts` command allows you to specify the hosts you want to add to the virtual farm. Adding hosts to the farm, however, does not start any processes.

For more information about specifying processes, see the `add_distributed_hosts` man page.

Creating a Distributed Farm

Generally, there is a time lag between when a remote host is launched and when it becomes available for use by the master process. This is especially noticeable when using managed resources, such as Load Sharing Facility (LSF) or Global Resource Director (GRD). Loading on the managed resource could incur long waiting times before the requested resources are made available.

To control the wait time for remote processes, use the `-timeout` and `-min_hosts` options of the `create_distributed_farm` command. The `-timeout` option allows you to specify how long the `create_distributed_farm` command should wait for remote processes to become available. If you have not specified the `-timeout` option, a default time-out value of 21,600 seconds (6 hours) is used. The `-min_hosts` option allows you to specify the minimum number of hosts needed to enable the `create_distributed_farm` command to return and allow execution to proceed.

The `create_distributed_farm` command completes when one of the following conditions is met:

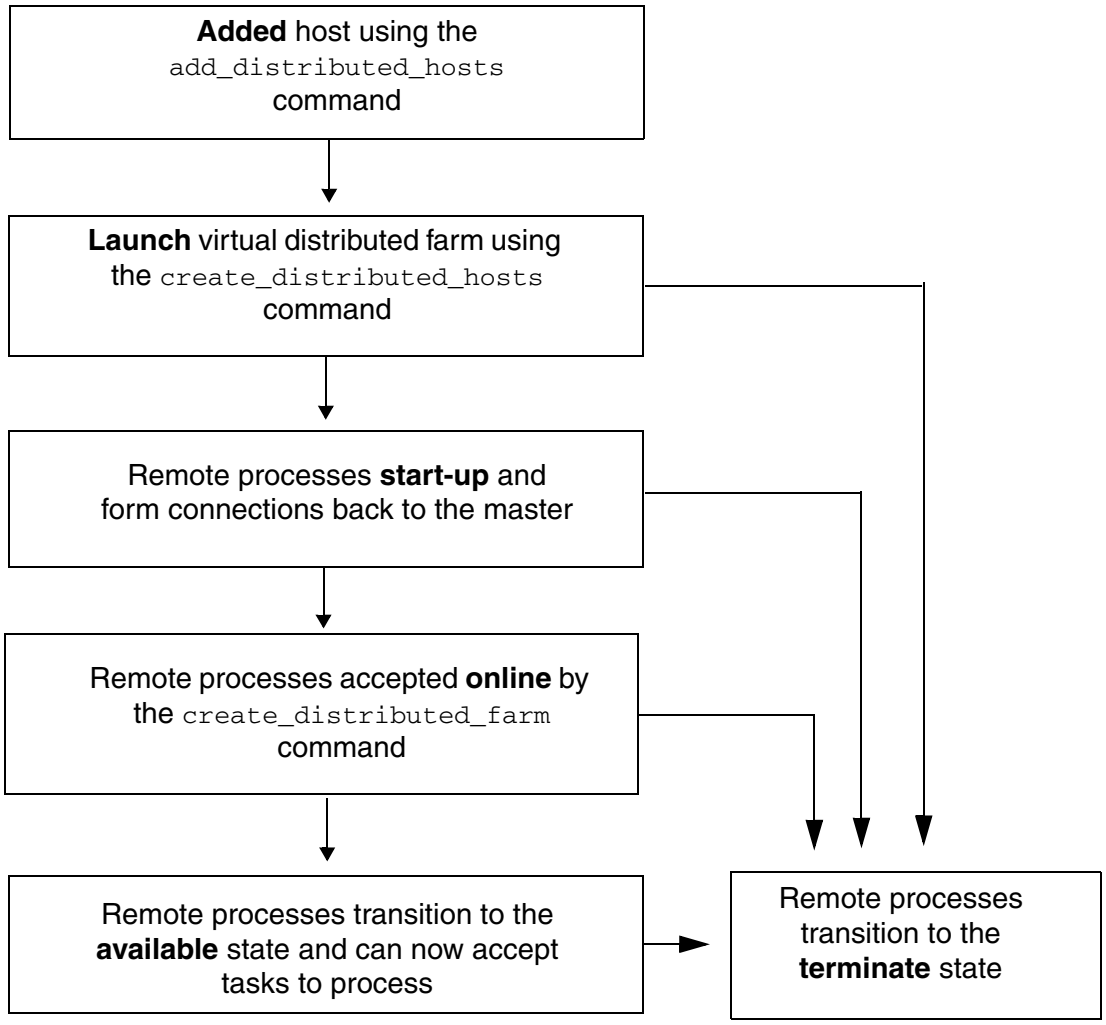
- All remote host processes become available.
- If specified, the `-min_host` minimum number of remote host processes becomes available.
- The time-out expires.

When the `create_distributed_farm` command returns, execution proceeds as long as at least one host is available. As additional remote processes become available during the analysis, they are dynamically made available for use to the analysis. If no remote processes were available when the `create_distributed_farm` command returns, an error is issued.

Although you can set the session to begin analysis when just one host is available, remember that the number of available hosts directly impacts the ability of PrimeTime to simultaneously analyze processes. When there is an imbalance between scenarios and hosts, expensive image swapping occurs until sufficient hosts become available for each scenario to occupy a remote host. If you set the `-min_hosts` option to a low value or you set the `-timeout` option to a small value, an imbalance is likely and leads to image swapping until sufficient hosts become available. The `create_distributed_farm` command is limited to a maximum of 256 remote host processes.

When the `create_distributed_farm` command is called, the previously added hosts begin to transition through the states of their lifecycle that is shown in Figure 1-3.

Figure 1-3 Lifecycle States



License Resource Management

By default, the master takes care of its own licensing regardless of the licenses required to perform multi-scenario analysis. After the master is launched, it acquires any needed PrimeTime licenses from the license pool in the same way as ordinary PrimeTime. The master then dynamically distributes the licenses to the slave processes. You must specify an upper limit for the number of licenses you want the master to use for the slave processes.

The `set_multi_scenario_license_limit` command sets the upper limit for the number of licenses of a particular feature the master can check out from the license pool to use for multi-scenario processing. Any licenses that the master checks out are made available for slave usage. For instance, five PrimeTime licenses allow one master and five slaves to operate concurrently.

Setting the limit on the number of licenses to use does not cause PrimeTime to check out any licenses from the central pool. The licenses are checked out after the slaves begin processing tasks. To specify that the licenses are checked out immediately, use the `-force_checkout` option. For instance, in the following example, the limit on the number of PrimeTime licenses to use is set to five. The master process checks out no more than four additional licenses (but might check out fewer depending on the availability in the main licensing pool) making up to a maximum of five licenses available for slave usage:

```
pt_shell> set_multi_scenario_license_limit -feature
PrimeTime 5
```

If any analysis being performed involves crosstalk analysis, you must set the limit on the number of PrimeTime SI licenses to use during multi-scenario analysis. If the limit is not set before the analysis begins, the analysis proceeds just as in PrimeTime, without the crosstalk features. For example, the following command sets the limit on the number of PrimeTime SI licenses for slave usage to five. Since the master has not checked out a PrimeTime SI license, it checks out up to five PrimeTime SI licenses for slave usage during multi-scenario analysis:

```
pt_shell> set_multi_scenario_license_limit -feature
PrimeTime-SI 5
```

Since the master dynamically allocates licenses to the slave processes on an as-needed basis, a slave process uses a license only when it is actively executing a task. This is important in the context of maximizing the overall performance of the multi-scenario mechanism. Therefore, it is possible to have more slave processes added than licenses available for them to use. The master distributes tasks across all of the slave processes, and as each begins to execute a task, it checks out the appropriate licenses from the master for the duration of time it takes to execute the task. After the task is complete, the slave process returns the licenses to the master for other slave processes to use.

The benefit of this mechanism is that if you have more slave processes than licenses, you can maximize the multi-scenario system throughput by minimizing the expensive process involved in swapping out a scenario.

For optimal system performance of the distributed multi-scenario mechanism, match the license limit to the number of slave processes added. In this way, all slave processes can execute concurrently, thereby maximizing the system throughput.

Note:
You cannot specify fewer licenses than have already been allocated.

The license management capabilities enable you to do the following:

- Lower the license limit. Any licenses over the limit are returned to the license server pool. For more information, see “Incremental License Handling.”
- License spanning is supported, so that it is possible to check out licenses of a single feature from multiple license servers. For more information, see “License Spanning” on page 1-15.
- Instruct the master to automatically reduce the number of licenses for features checked out to a minimum when those licenses are not needed by using dynamic license reduction. For more information, see “License Auto-Reduction” on page 1-15.

Incremental License Handling

By default, the master does not allow you to reduce the licensing limit for a feature. With this configuration, when the master checks out licenses of a feature, they are not returned to the central license server until the master exits. You can configure the master to enable license reduction by enabling incremental license handling, which is not enabled by default. To enable incremental license handling, set the `SNPSLMD_ENABLE_INCREMENTAL_LICENSE_HANDLING` environmental variable to 1. The syntax is as follows:

```
setenv SNPSLMD_ENABLE_INCREMENTAL_LICENSE_HANDLING 1
```

If you do not set this environmental variable or set it to anything other than 1, the master is not able to check licenses back into the central license server until it exits. Changing the environmental variable from within the master session has no impact.

When you enable incremental license handling, you can use the `set_multi_scenario_license_limit` command to both increase and decrease the license upper limit of a feature. When the upper limit is being lowered, the upper limit is immediately reduced and the master checks in sufficient licenses of the feature to the central license server, if needed, to meet the new target limit. If no licenses have been checked out from the central license server or the number checked out is less than the new limit, the command lowers the upper limit.

Note:
There must always be a minimum of 1 license checked out for all features in use at all times. Therefore, you cannot use the `set_multi_scenario_license_limit` command to reduce the upper limit to 0.

License Spanning

License spanning provides the ability to check out licenses of a single feature from multiple license servers. You can activate license spanning by enabling incremental license handling. Before the master is launched, set the `SNPSLMD_ENABLE_INCREMENTAL_LICENSE_HANDLING` environmental variable to 1. For more information about this variable, see the man page.

License Auto-Reduction

License auto-reduction instructs the master to automatically reduce the number of licenses for features checked out to a minimum level when those licenses are not being used to perform analysis work. You can enable or disable license auto-reduction at any time from within the master by setting the `enable_license_auto_reduction` global variable to `true` to enable or `false` to disable. When auto-reduction is enabled, if a scenario task is completed and no additional scenarios are waiting for the license, the license is returned to the central license pool.

It is important to use this feature carefully because although it allows licenses to return to the central license server, there is no guarantee that subsequent analysis is able to reacquire those licenses for further use. If the master cannot reacquire the licenses or can only partially reacquire them, the level of concurrency in the distributed multi-scenario analysis is reduced. This can impact performance and time-to-results. Also, the time needed to repeatedly check in and check out the licenses themselves can significantly slow down interactive work, such as merged reporting or the execution of Tcl scripts or procedures that communicate heavily with the scenarios. In these cases, it might be desirable to enable auto-reduction during the initial timing update and then disable it during interactive work as follows:

```
# get all scenarios to update their timing
set enable_license_auto_reduction  true
remote_execute {update_timing}

# generate merged reports
set enable_license_auto_reduction  false
report_timing ...
report_qor ;# Tcl proc from SolvNet (Doc ID 008602)
dmsa_fix_hold ... ;# Tcl proc from SolvNet (Doc ID 018510)
```

Database Support

File reading and support at the slave is supported without restriction. This means that support for the `read_ddc` and `read_milkyway` commands in distributed multi-scenario analysis is identical to the support for these commands in PrimeTime.

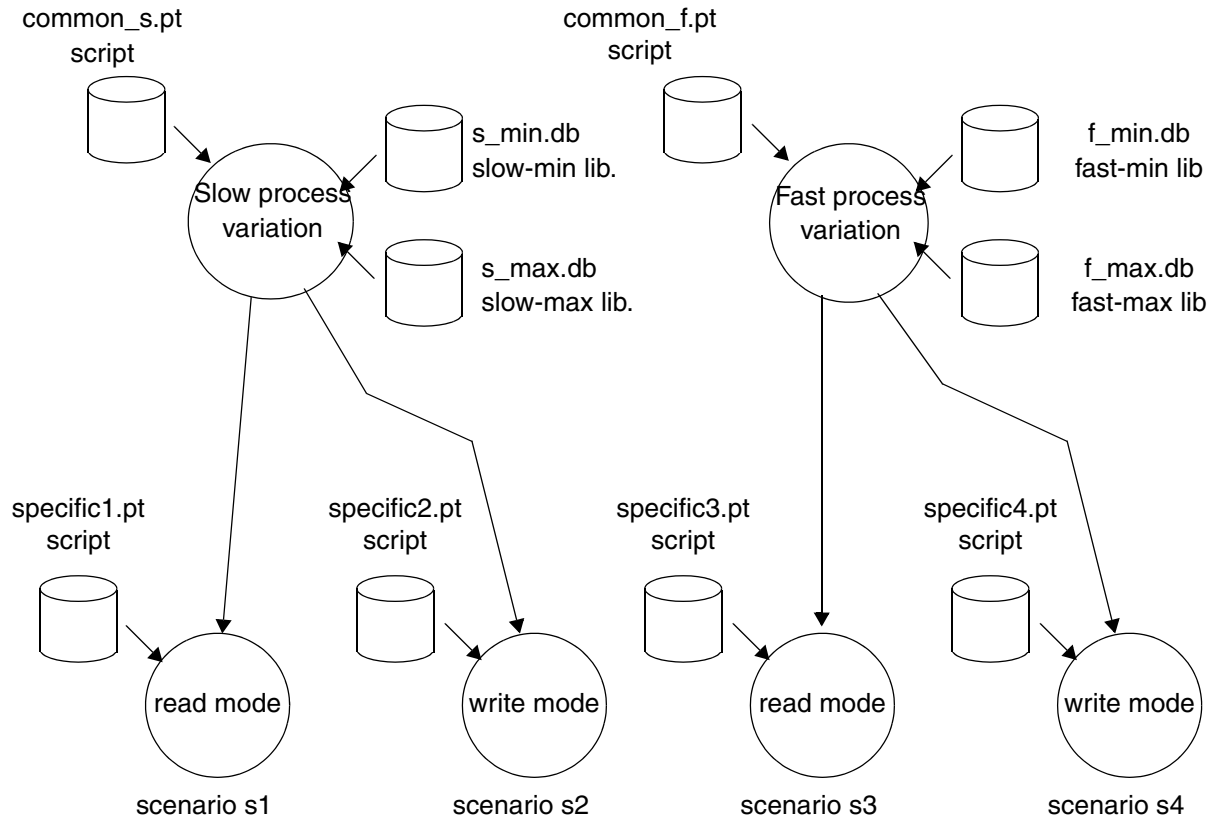
Scenarios

A scenario is a specific combination of operating condition and operating mode for a given configuration. You create a scenario with the `create_scenario` command, as in the following example:

```
pt_shell> create_scenario -name scen1 \
            -common_data {common_s.pt} \
            -specific_data {specific1.pt}
```

This command specifies a scenario name, the common-data script files, and the specific-data script files. The scripts associated with the scenario are divided into two categories: common data (shared by two or more scenarios) and specific data (not shared by any other scenarios). Common-data scripts are included in the baseline image generation process, whereas specific-data scripts are applied only to the specified scenario. Organizing scripts into categories allows the master to share design data between groups of scenarios where appropriate, thereby reducing the total runtime for all scenarios. The baseline image generation process is described in greater detail in the “Distributed Analysis Process” on page 1-20. For example, consider the scenarios illustrated in Figure 1-4. The figure shows the types of scripts that can be applied as common-data and specific-data scripts.

Figure 1-4 Multi-Scenario Setup Example



This design is to be analyzed at two process variation extremes, called slow and fast, and two operating modes, called read and write. Thus, there are four scenarios to be analyzed: slow-read, slow-write, fast-read, and fast-write. You can add more conditions and modes by creating several scenarios as indicated in the figure.

The `common_all.pt` script reads in the design and applies constraints that are common to all scenarios. The `common_s.pt` script is shared between some, but not all, scenarios. The `specific1.pt` script directly contributes to specifying conditions of an individual scenario and is not shared at all between different scenarios.

Removing Scenarios

To remove a scenario previously created with the `create_scenario` command, use the `remove_scenario` command:

```
pt_shell> remove_scenario s1
```

Current Session and Command Focus

The current session is the set of scenarios to be analyzed and is established using the `current_session` command, as in the following example:

```
pt_shell> current_session {s2 s3 s4}
```

The command lists the scenarios to be part of the current session, all of which must be valid scenario names already defined by the `create_scenario` command. By default, all listed scenarios are in command focus. The command focus is the set of scenarios in the current session to which analysis commands are applied.

After using `current_session`, you can further narrow the command focus by using the `current_scenario` command to select a subset of the scenarios in the current session:

```
pt_shell> current_scenario {s3}
```

This command is useful during interactive analysis. For example, you might want to modify the timing characteristics of a path in one scenario by annotating a load on a particular net, and then run a new delay calculation on the net's driving cell or recheck the timing of the whole path for that scenario alone.

After narrowing the command focus, you can restore the command focus to all scenarios in the session by using the `-all` option:

```
pt_shell> current_scenario -all
```

Note:

Changing the *current-session* list requires images to be rebuilt and is a zero-cost operation.

Checking the Setup

To check the distributed processing setup before you begin multi-scenario analysis, use the `report_multi_scenario_design` command. For more information about this command, see the man page.

Remote Execution

To use the `remote_execute` command, enclose the list of commands in a Tcl string. The commands should be separated by semicolons as this allows them to be executed one at a time rather than as a group. To evaluate subexpressions and variables remotely, generate the command string using curly braces. For example,

```
remote_execute {
```

```
  # variable all_in evaluated at slave
  report_timing -from $all_in -to [all_outputs]
}
```

In this example, the `report_timing` command is evaluated at the slave, and the `all_in` variable and the `all_outputs` expression are evaluated in a slave context.

To evaluate expressions and variables locally at the master, enclose the command string in quotes. All master evaluations must return a string. For example,

```
remote_execute "
  # variable all_out evaluated at master
  report_timing -to $all_out
"
```

In this example, the `report_timing` command is evaluated at the slave, and the `all_out` variable is evaluated at the master.

Upon issuing the `remote_execute` command, the master generates tasks for execution in all scenarios in command focus.

If you use the `-pre_commands` option with the `remote_execute` command, the listed commands are executed before the command specified for remote execution.

If you use the `-post_commands` option, the listed commands are executed after the commands specified for remote execution.

Example 1: Slave and master context variables

```
remote_execute {set x 10}
# Send tasks for execution in all scenarios in command focus
set x 20
remote_execute {report_timing -nworst $x}
# Leads to report_timing -nworst 10 getting executed at the slaves

remote_execute "report_timing -nworst $x"
# Leads to report_timing -nworst 20 getting executed at the slaves
```

Example 2: `-pre_commands` option used with the `remote_execute` command

```
remote_execute -pre_commands {cmd1; cmd2; cmd3}
“report_timing”
# On the slave host, execute cmd1, cmd2, and cmd3 before
# executing report_timing on the master
```

Example 3: `-post_commands` option used with the `remote_execute` command

```
remote_execute -post_commands {cmd1; cmd2; cmd3}
“report_timing”
# On the slave host, execute cmd1, cmd2, and cmd3 after
```

```
# executing report_timing on the master
```

You can use the `remote_execute` command with the `-verbose` option to return all slave data to the master terminal screen, instead of piping it to the `out.log` file in the working directory of the distributed multi-scenario analysis file system hierarchy.

The ability to execute netlist editing commands remotely extends to all PrimeTime commands except the following:

- explicit `save_session` (slave only)
- explicit `restore_session`
- `remove_design`

Distributed Analysis Process

After you set up the analysis with the `create_scenario`, `current_session`, and (optionally) `current_scenario` commands, you can start analysis of the scenarios in command focus.

The `remote_execute` command is used in the master to explicitly evaluate commands, variables, and expressions in the slave context. There is a slight difference in the way that you can use the command, depending on whether you are executing commands in batch or interactive mode.

In batch mode, the commands are contained in scripts. For example, you might execute the following command at the UNIX prompt to invoke PrimeTime and execute a multi-scenario analysis in batch mode:

```
% pt_shell -multi_scenario -f script.tcl
```

The following is a sample script:

Note:
You must specify the `multi_scenario_working_directory` and `multi_scenario_merged_error_log` variables before creating the distributed farm.

```
script.tcl
################################
# Start of specifying the set-up
# Start of specifying the first task
################################

# Add 2 32 bit hosts to the pool of hosts for
# the distributed farm from hosts platinum1 and platiunm2
add_distributed_hosts -32bit -farm now platinum1
add_distributed_hosts -32bit -farm now platinum2
```

```
# Create the distributed farm for processing the scenarios
create_distributed_farm

set_multi_scenario_license_limit -feature PrimeTime 2
# Set the limit on the number of PrimeTime licenses usable
# by slaves to 2

create_scenario \
   -name scenario1 \
   -common_data  common_data_scenarios_s1_s3.tcl \
   -specific_data  specific_data_scenario_s1.tcl
create_scenario \
   -name scenario2 \
   -common_data  common_data_scenarios_s2_s4.tcl \
   -specific_data  specific_data_scenario_s2.tcl

create_scenario \
   -name scenario3 \
   -common_data  common_data_scenarios_s1_s3.tcl \
   -specific_data  specific_data_scenario_s3.tcl

create_scenario \
   -name scenario4 \
   -common_data  common_data_scenarios_s2_s4.tcl \
   -specific_data  specific_data_scenario_s4.tcl

##############################
# End of specifying the setup
##############################

current_session {scenario1 scenario2}

remote_execute -pre_commands {"cmd1" "cmd2" "cmd3"} \
   {report_timing}
# cmd1,cmd2,cmd3, are placeholders for typical
# PrimeTime commands or scripts that need to be
# executed on scenario1 and scenario2

##################################
# End of specifying the first task
##################################

quit

####################################
```

A task is a self-contained block of work that a slave process needs to execute in order to perform some function. A slave process starts execution of a task for the specified scenario as soon as it gets the licenses appropriate to that task. Otherwise, it waits for the licenses it requires.

In the `script.tcl` example, the main task that is user directed consists of cmd1, cmd2, cmd3 and the `report_timing` command. However, there are other internal tasks that need to happen before you can execute the task containing these commands. This process is called common image generation. The following are the processes that take place prior to executing your commands but after specifying all scenarios, sessions, and so on.

The first set of tasks the master generates and delegates to arbitrary slave processes is to generate the baseline images. Every scenario in command focus requires access to a baseline image which consists of the netlist image and the common data files for that scenario. The slave process assigned to execute a baseline image generation task executes the common data files for that scenario in the order in which they were specified at the time the scenario was created using the `create_scenario` command. After this, the slave process generates the baseline image for that scenario.

Before any other types of tasks can proceed in the scenario that is having its baseline image generated, the baseline image generation process must complete successfully. Given this dependence, failure of this process results in failure of the multi-scenario analysis only for those scenarios that depend on that failed image generation. The MS-020 warning message is issued and resources are reduced. Scenarios that have access to a successfully generated baseline image can proceed normally.

For each scenario in command focus that provides a baseline image, that image is written to the scenario_name/baseline_image directory under the multi-scenario working directory.

The third set of tasks that the master generates and delegates to slave processes is the execution of the user-specified commands in the scenarios that are in command focus. The slave process assigned a command execution task uses the baseline image for the scenario in which these commands need to be executed (this configures the slave process for the scenario), executes the specific data files for that scenario, and then executes the commands specified.

Where there are more scenarios in command focus than processes to execute the tasks for those scenarios, the slave process needs to save out a current image for one scenario while it proceeds with task execution for a different scenario. If this swapping is necessary, the slave process saves the current image in the scenario_name/current_image directory under the multi-scenario working directory.

Achieving Optimal Performance

It is important to remember that the automatic reconfiguring of slave processes to run different tasks in different scenarios can be an expensive operation and should be minimized to maximize system throughput. The multi-scenario process operates to minimize the swapping out of scenarios, but is subject to the limitations imposed by the number of

scenarios in command focus and the number of slave processes added. If there is a significant imbalance of scenarios to slave processes, as a whole the system performance is degraded.

You can achieve optimal performance from the distributed multi-scenario mechanism if there is a slave process for every scenario in command focus, a CPU available to execute each slave process, and a license for each feature needed for every slave process. This results in all slave processes executing concurrently, allowing maximum throughout for the system.

Saving and Restoring Your Session

Baseline images cannot be restored by a normal PrimeTime session; however, current images can be restored. You can also issue a `save_session` command from the master to generate an image for each one of the scenarios in command focus. For example,

```
current_session {scen1  scen2}
save_session -replace /images
```

You can find the images generated by the master-issued `save_session` command in the $sh_launch_dir/images/scen1 and $sh_launch_dir/images/scen2 directories. You can then reload these images into a normal PrimeTime session.

The save and restore features allow you to explicitly restore a saved image into a distributed multi-scenario analysis scenario. This is achieved with the addition of the following switch:

```
create_scenario -image
```

Using this switch, you can load any single-core analysis PrimeTime image that has been produced by a `save_session` command (either inside or outside of distributed multi-scenario analysis). Load this image into distributed multi-scenario analysis directly as a scenario. You can then operate upon it in the normal way within the context of distributed multi-scenario analysis.

Manipulating Variables

You can control the value of a variable in either the master or any of its slaves during distributed multi-scenario analysis.

Master Context Variables

The following is an example of a master context variable:

```
set x {"ffa/CP ffb/CP ffc/CP"}
report_timing -from $x
```

Notice how x is forced to be a string.

Slave Context Variables and Expressions

The master has no knowledge of variables residing on the slave, but can pass variable or expression strings to the slave for remote evaluation. To pass a token to the slave for remote evaluation, use curly braces. For example, say you had the following scenarios:

- Scenario 1: x has value of ff1/CP
- Scenario 2: x has value of ff2/CP

The `report_timing -from {$x}` command would report the worst paths from all paths starting at ff1/CP in the context of scenario 1, and all paths starting at ff2/CP in the context of scenario 2.

Setting Distributed Variables

With the `set_distributed_variables` command you can set the values of variables at the slaves from the master. The variable you set can be a Tcl scalar variable, Tcl collection, Tcl array, or Tcl list.

To set variables in multiple scenarios, you first create a Tcl array of the desired values for those scenarios, then use the `set_distributed_variables` command to distribute the settings to the scenarios. For example, suppose that you have two scenarios, s1 and s2, running under the control of a master. From the master, you want to set the value of two variables x and y at the slave level, so that x=10 and y=0 in s1, and x=0 and y=15 in s2. You create an array, indexed by scenario name, containing the values to set in the slaves. From the master you execute:

```
pt_shell> array set x {s1 10 s2 0}
pt_shell> array set y {s1 0 s2 15}
pt_shell> set_distributed_variables {x y}
```

This sends the values specified in the arrays x and y from the master to the slaves, and creates the variables x and y at the slaves if they do not already exist, and sets them to the desired values.

When variables are pushed from the master into the scenarios, nonarray variables at the master are supported. The specified variable is created at each of the scenarios. This avoids the need for creating a temporary array at the master to push a simple variable value. For example,

```
pt_shell> set test_var 123
123

pt_shell> set_distributed_variables test_var
```

For more information about the `set_distributed_variables` command, see the man page.

Getting Distributed Variable Values

Using the `get_distrubuted_variables` command provides the ability to get the values of variables set in scenarios in command focus. The variables retrieved by the command can be a Tcl variable, collection, array, or list. The retrieved values are returned to the master process and stored in a Tcl array, indexed by scenario name. For example, the following session retrieves slack attribute values from two scenarios:

```
current_session {scen1 scen2}

remote_execute {
     set pin_slack1 [get_attribute -class pin UI/Z slack];
     set pin_slack2 [get_attribute -class pin U2/Z slack]
               }

get_distributed_variables {
                pin_slack1 pin_slack2
               }
```

The session returns two arrays, pin_slack1 and pin_slack2. Indexing pin_slack1(scen1) returns the scalar value of slack at U1/Z in context of scenario scen1. Indexing pin_slack1(scen2) returns the scalar value of slack at U1/Z in the context of scenario scen2. When retrieving a collection, you must indicate what attributes you are interested in for the command to generate the desired list. The following session retrieves timing paths from two scenarios:

```
current_session {scen1 scen2}
remote_execute {
                set mypaths [get_timing_paths -nworst 100]
               }
get_distributed_variables mypaths -attributes (slack)
```

The session returns an array called mypaths. Indexing mypaths(scen1) yields a collection of the 100 worst paths from scenario scen1, ordered by slack. Similarly, indexing paths(scen2) yields a collection of the 100 worst paths from scen2.

When retrieving variables at the master, any existing array variables are updated with the retrieved values. For more information, see the man page.

Merging Distributed Variable Values

You can use the `-merge_type` and `-null_merge_method` options of the `get_distributed_variables` command to merge variable values that come from the scenarios as they are retrieved. This is especially useful when you are writing customized distributed multi-scenario analysis scripts.

By default, the `get_distributed_variables` command brings back a scenario variable as an array. With the `-merge_type` option, the values can be merged back into a variable during retrieval. The merge types that are available are `min`, `max`, `average`, `total`, `list`, `sum`. and `unique_list`. For example, to keep the numerically smallest value of the `worst_slack` variable from all scenarios, use the following commands:

```
pt_shell> get_distributed_variables {worst_slack} -merge_type min
1
pt_shell> echo $worst_slack
-0.7081
```

You can also merge arrays of one or more dimensions and keep the worst value for each array entry. For example, if you have the following scenarios:

```
scenario best_case:
set slacks(min,pinA) -0.2044
set slacks(max,pinA) 0.3084

scenario type_case:
set slacks(min,pinA) -0.0523
set slacks(max,pinA) 0.0901

scenario worst_case:
set slacks(min,pinA) 0.1015
set slacks(max,pinA) -0.7081
```

You would use the following commands to obtain the merge results listed:

```
pt_shell> get_distributed_variables {slack} -merge_type min
1
pt_shell> echo $slacks(min, pinA)
-0.2044
pt_shell> echo $slacks(max, pinA)
-0.7081
```

The `-merge_type` option can also merge together lists of values from the scenarios. For example, to obtain a unique list of all clock names from every scenario, use the following commands:

```
pt_shell> remote_execute {set clock_names \
          [get_object_name [all_clocks]]}
pt_shell> get_distributed_variables clock_names -merge_type list
1
pt_shell> echo $clock_names
CLK33 CLK66 CLK50 CLK100 ATPGCLOCK JTAGCLK
```

The merged list is built from the list contents from each scenario with any duplicate list entries of identical values omitted.

To merge lists of object name lists in a multi-scenario analysis flow, a specialized `unique_list` merging mode is available. This mode is useful when you want to bring back multiple lists of design object names, such as cell, pin, or net names, and there might be small variations between the scenario results due to differing constants, constaints, or interface logic model (ILM) netlists, or differences in the design. Where possible, the `unique_list` merging mode keeps the sublists separate so that gaps can be filled, but merges these sublists together as needed. Using this merging mode is similar to applying the list merging type to each of the sublists, but additionally merges together any sublists that share common items. For example, consider following variable variables,

```
scenario best_case:
set cells {{U1 U2} {U4 U5} {U7 U8}}

scenario type_case:
set cells {{U2 U3} {U6 U7}}

scenario worst_case:
set cells {{U1} {U5 U6}}
```

Retrieving them with the `unique_list` merging mode would provide the following result:

```
{{U1 U2 U3} {U4 U5 U6 U7 U8}}
```

The merged list is built from the list contents from each scenario, with any duplicate list entries of identical value omitted.

The `-null_merge_method` option specifies what behavior should be used when a null (empty) value of {} is encountered during the numerical merge operations. The available options are:

- `ignore` (the default)

  Ignore any null values and only process the non-null values

- `error`

  Abort the merging process and issue an error

- `override`

  Null values win the merging process and a null is returned

For example, you might have the libraries in the type_case scenario misconfigured such that the BUFFX2 buffer cell cannot be resolved, and the value for this array entry was set to null. With the default null merging method of ignore, the null is ignored and the remaining BUFFX2 values are averaged as shown in the following example.

```
scenario best_case:
set buffer_delay(BUFFX1) 0.105
set buffer_delay(BUFFX2) 0.210

scenario type_case:
set buffer_delay(BUFFX1) 0.130
set buffer_delay(BUFFX2) {}

scenario worst_case:
set buffer_delay(BUFFX1) 0.140
set buffer_delay(BUFFX2) 0.280
```

If it is possible that some data entries can be null, you can specify the -null_merge_method override option to detect these cases of incomplete data array entries. In this situation, the null wins during the numerical merging process and overrides the other numerical values for that array entry, allowing the missing data to be detected in a controlled manner. For example, you would specify the following commands:

```
pt_shell> get_distributed_variables {buffer_delay} -merge_type average \
          -null_merge_method override
1
pt_shell> echo $buffer_delay(BUFFX2)

pt_shell> echo $buffer_delay(BUFFX1)
0.125
```

Alternatively, you can use the null merging method of error so that the get_distributed_variables command aborts with an error. For example,

```
pt_shell> get_distributed_variables {buffer_delay} -merge_type average \
          -null_merge_method error
Error: Found null value in merged_object when -null_merging_method was

set to error (ZDPP-022)
```

The error null merging method can be used as an assertion to trap cases that should never happen. For example, if the array data should always be complete, it can be used to assert an error if some data is null for some unexpected reason.

Note:

A `null` data value is different from when a value is undefined. If array entries are undefined, the `-null_merge_method` option does not apply and the merging process operates normally on the remaining data values. For example, in the type_case scenario, if the scenario scripting omitted the `BUFFX2` array entry rather than setting it to `null`, it would be impossible to detect any missing data after applying the merging process.

Merged Reporting

Executing reporting commands in multi-scenario analysis can generate large amounts of data and detailed information about each scenario. The sheer quantity of data can make it difficult to identify the critical information. To manage these large data sets, and to help you identify critical issues across the design, PrimeTime provides the merged reporting capability.

Merged reporting is a mechanism that works with all the scenarios in command focus to automatically eliminate redundant data across scenarios and sort the data in order of criticality. This allows you to treat all the scenarios in command focus as if they are a single virtual PrimeTime instance. The merge reporting capability applies to the `get_timing_paths`, `report_analysis_coverage`, `report_clock_timing`, `report_constraint`, `report_si_bottleneck`, and `report_timing` commands.

The following sections provide general information on the reporting methods as well as specific information for each merge reporting command.

Using Merged Reporting

To use merged reporting for a timing report, issue the `report_timing` command at the master as you would in the single-core analysis PrimeTime, as in the following example:

```
pt_shell> report_timing
```

Executing the command in this way is called executing the command in the master context. This results in the master and slave processes working together to execute the `report_timing` command in all the scenarios in command focus, to produce a final merged report that is displayed at the master console. Each of the reports include the scenario name from which it came.

Generating Merged Reports

When issuing commands in a master context, the commands are operating in a distributed manner, which necessitates specifying some of their options in a slightly different way compared to the single-core analysis mode.

Using commands in distributed multi-scenario analysis is similar to using these same commands in a single-core analysis session of PrimeTime. All of the options exist in the merged mode. However, you must first invoke the `-multi_scenario` option to use the `-pre_commands` and `-post_commands` options. These options allow you to specify a string of commands to execute in the slave context before and after executing the merged report command. In situations where there are more scenarios than slave processes, using these options can reduce the amount of swapping out of scenarios by performing the `-pre_commands` or `-post_commands` options and the merged reporting command in series, as opposed to multiple swapping out and restoring the same scenarios for each command.

The following sections detail the commands:

- get_timing_path
- report_analysis_coverage
- report_clock_timing
- report_constraint
- report_si_bottleneck
- report_timing

get_timing_path

Using the `get_timing_paths` command from the master process returns a collection of timing path objects from each scenario, based on the parameters you set in the command. The timing path collection is condensed to meet the requirements you specify (such as, using `-nworst` or `-max paths`). It returns a merged output collection of the worst timing paths across all scenarios according to the reporting criteria set.

To find the scenario a particular timing path object belongs to, use the `scenario` attribute on the timing path object itself. To generate a list of attributes you are interested in, use the `-attributes` option with the `get_timing_path` command.

report_analysis_coverage

The `report_analysis_coverage` command reports the coverage of timing checks over all active scenarios in the current session. The timing check is defined by the constrained pin, the related pin, and the type of the constraint. The timing check status is reported

- As violated if the calculated signal propagation time along the particular data path does not meet timing requirements (constraints)
- As met if timing requirements are satisfied
- As untested if the timing check was skipped

Timing checks are included in the Untested category only if they are untested in all scenarios. If a timing check is tested in at least one scenario, the timing check is reported as tested (showing in either the Violated or Met category) because it has been successfully exercised in some scenario. This allows meaningful reporting to be performed across scenarios with significantly differing behavior (for example, functional versus test modes).

The output for the merged reports is similar to the single-core analysis coverage reports; however, the merged reports contain a list of scenarios. Example 1-1 shows the summary merged analysis coverage report.

Example 1-1 Summary Merged Analysis Coverage Report

```
pt_shell> report_analysis_coverage

****************************************
Report : analysis_coverage
Design : multi-scenario design
Version: B-2008.06
Date   : Mon Mar 31 07:17:11 2007
****************************************

Scenarios: SLOW_OPER, NOM_OPER, FAST_TEST

Type of Check            Total         Met            Violated         Untested
-------------------------------------------------------------------------------
setup                      8        3 (37.5%)        3 (37.5%)         2 ( 25%)
hold                       8        3 (37.5%)        3 (37.5%)         2 ( 25%)
-------------------------------------------------------------------------------
All Checks                16        6 (37.5%)        6 (37.5%)         4 ( 25%)
```

If a timing check is untested in all scenarios in the command focus, instead of a list of scenarios, the `all` tag is listed in the scenario column; however, if a timing check is tested in at least one scenario, the check is reported as tested, either violated or met, because it has been successfully exercised in some scenarios. Example 1-2 shows a report where some scenarios are met and others are untested.

Example 1-2 Detailed Merged Analysis Coverage Report

```
pt_shell> report_analysis_coverage -status_details {untested met} -sort_by slack

****************************************
Report : analysis_coverage
        -status_details {untested met}
        -sort_by slack
Design : multi-scenario design
Version: B-2008.06
Date   : Tue Mar 11 11:42:06 2008
****************************************
```

```
Scenarios: scen1, scen2, scen3

Type of Check          Total          Met           Violated         Untested
-------------------------------------------------------------------------------
setup                    15         2 ( 13%)        12 ( 80%)         1 (  7%)
hold                     15        12 ( 80%)         2 ( 13%)         1 (  7%)
-------------------------------------------------------------------------------
All Checks               30        14 ( 47%)        14 ( 47%)         2 (  7%)


Constrained       Related             Check
Pin               Pin         Clock   Type             Scenario    Slack  Reason
-------------------------------------------------------------------------------
ffd/CR            CP(rise)            hold              scen2         untested
                                                                    constant_disabled
                                                       scen1
ffd/CR            CP(rise)            setup             scen2         untested
                                                                    constant_disabled
                                                       scen1
ffa/D             CP(rise)    clk2    hold              scen2        0.14
ffa/D             CP(rise)    CLK     hold              scen2        0.14
ffb/D             CP(rise)    CLK     hold              scen2        0.14
ffb/D             CP(rise)    clk2    hold              scen2        0.14
ffd/D             CP(rise)    CLK     hold              scen2        0.14
ffd/D             CP(rise)    clk2    hold              scen2        0.14
ffa/D             CP(rise)    clk3    hold              scen1        0.15
ffb/D             CP(rise)    clk3    hold              scen1        0.15
ffc/D             CP(rise)    clk3    hold              scen1        0.15
ffc/D             CP(rise)    CLK     hold              scen1        0.15
ffd/D             CP(rise)    clk3    hold              scen1        0.15
ffd/CR            CP(rise)    CLK     setup             scen3        9.10
ffd/CR            CP(rise)    clk2    setup             scen3        9.15
ffc/D             CP(rise)    clk2    hold              scen2        9.40
1
```

report_clock_timing

The `report_clock_timing` command executed by the master process returns a merged report that displays the timing attributes of clock networks in a design. The output for the merged `report_clock_timing` report is similar to the single-core analysis `report_clock_timing` report; however, the merged report contains a list of scenarios, which are displayed under the Scen column. Example 1-3 shows an example of the output of the merged `report_clock_timing` command.

Example 1-3 Merged report_clock_timing Report

```
pt_shell> report_clock_timing -type latency -nworst 3

****************************************
Report : clock timing
         -type latency
         -launch
         -nworst 3
         -setup
```

```
Design  : multi-scenario design
Version : C-2009.06
Date    : Tue Jun 30 02:44:25 2009
****************************************

Scenarios: scen1, scen2

Clock: CLKA
                                                    --- Latency ---
Clock Pin                 Scen    Trans    Source   Network   Total
-------------------------------------------------------------------------
 1_1/CP                   scen1   0.00     0.11     25.36     25.47   rp-+
 f1_3/CP                  scen2   0.00     0.11     23.96     24.07   rp-+
 gclk_ff/CP               scen1   0.00     0.10     22.96     23.06   rpi-+
-------------------------------------------------------------------------
1
pt_shell>
```

report_constraint

The `report_constraint` command executed by the master process returns a merged report that displays constraint-based information for all scenarios in command focus. It reports whether or not a constraint is violated or met, by how much that constraint is violated, and which design object is the worst violator.

The merging process for the `report_constraint` command for distributed multi-scenario analysis treats all the scenarios in command focus as if they are a single virtual PrimeTime instance. When an object is constrained in more than one scenario, these constraints are considered duplicates. PrimeTime reports the worst violating instance of each relevant object in the design. For example, for a `max_capacitance` report, PrimeTime reports on the most critical instance for each pin in the design. When there are multiple instances of the same pin across multiple scenarios, PrimeTime retains the worst violator and uses the scenario name as a tie-breaker to ensure determinism. When you specify the `-verbose` or `-all_violators` option, PrimeTime reports the scenario name for the constraint violators.

Collection handling using the `report_constraint` command occurs in the same way as the `report_timing` command. For an example of the `report_timing` command collection handling, see “Explicit and Implicit Collection Example” on page 1-39.

Note:

There are many different types of constraints and reporting styles available when using the `report_constraint` command. For more information about the differences between single-core and multi-scenario reports, see Appendix C, “Differences Between Single-Core and Multi-Scenario Constraint Reports.

Example 1-4 shows the constraint report output for the merged distributed multi-scenario analysis constraint report.

Example 1-4 PrimeTime Distributed Multi-Scenario Analysis Merged Constraint Report

```
pt_shell> report_constraint -all_violators -path slack_only

****************************************
Report : constraint
        -all_violators
        -path slack_only
Design : multi-scenario design
Version: B-2008.06
Date   : Tue Apr 1 04:37:42 2007
****************************************

max_delay/setup ('default' group)

Endpoint                        Scenario      Slack
-------------------------------------------------------------------------
QA                               scen1        -1.80  (VIOLATED)
QB                               scen1        -1.79  (VIOLATED)

max_delay/setup ('CLK' group)

Endpoint                        Scenario      Slack
-------------------------------------------------------------------------
ffd/D                            scen1        -4.12  (VIOLATED)
ffc/D                            scen2        -4.01  (VIOLATED)
ffb/D                            scen1        -3.73  (VIOLATED)
ffa/D                            scen2        -3.60  (VIOLATED)
min_delay/hold ('CLK' group)

Endpoint                        Scenario      Slack
-------------------------------------------------------------------------
ffd/CR                          scen1        -0.40  (VIOLATED)
```

If you specify the `report_constraint` command in summary mode for distributed multi-scenario analysis when handling setup checks, the report shows the worst setup constraint for all scenarios per group. The hold information displays the sum of the worst total endpoint cost per clock group over all scenarios. Example 1-5 shows the summary output for the merged distributed multi-scenario analysis constraint report shown in Example 1-4.

Example 1-5 PrimeTime Distributed Multi-Scenario Analysis Merged Summary Report

```
pt_shell> report_constraint

****************************************
Report : constraint
Design : multi_scenario design
Version: B-2008.06
Date   : Tue Apr 1 04:37:42 2007
****************************************

                                                  Weighted
    Group (max_delay/setup)     Cost       Weight   Cost
    ---------------------------------------------------------
    default                      1.80        -      1.80
    CLK                          4.12        -      4.12
    ---------------------------------------------------------
    max_delay/setup                                 5.92
```

```
                                                    Weighted
Group (min_delay/hold)     Cost        Weight       Cost
-----------------------------------------------------------
CLK                         0.40         -          0.40
-----------------------------------------------------------
min_delay/hold                                     0.40

Constraint                                         Cost
-----------------------------------------------------------
max_delay/setup                                    5.92  (VIOLATED)
min_delay/hold                                     0.40  (VIOLATED)
```

Example 1-6 shows the output of an all-violators constraint report with the `max_capacitance` option for a multi-scenario analysis:

Example 1-6 PrimeTime Distributed Multi-Scenario Analysis Merged Constraint Report

```
pt_shell> report_constraint -all_violators -path slack_only -max_capacitance

****************************************
Report : constraint
         -all_violators
         -path slack_only
         -max_capacitance
Design : multi_scenario design
Version: B-2008.06-Beta3-DEV
Date   : Thu May 15 09:30:53 2008
****************************************
Scenarios: scen1 scen2
max_capacitance
                                                 Required        Actual
Pin                 Scenario     Capacitance     Capacitance     Slack
-------------------------------------------------------------------------------
RESET               scen2         0.40            7.00          -6.60 (VIOLATED)
ffa/QN              scen2         0.40            6.00          -5.60 (VIOLATED)
ffb/QN              scen1         0.50            5.00          -4.50 (VIOLATED)
ffc/QN              scen1         0.50            4.00          -3.50 (VIOLATED)
o/Z                 scen1         0.50            4.00          -3.50 (VIOLATED)
p/Z                 scen1         0.50            4.00          -3.50 (VIOLATED)
```

report_si_bottleneck

The `report_si_bottleneck` command allows you to locate nets that are most critical in crosstalk delay. With it, you need minimal net repair effort to identify and fix most problems. Across all scenarios, sorting and printing is performed with duplicates pruned to leave a unique set of nets across the scenarios.

The following example executes the `report_si_bottleneck` command at the master.

```
pt_shell> report_si_bottleneck \
          -cost_type total_victim_delay_bump -max \
          -slack_lesser_than 2 -significant_digits 6

****************************************
Report : si_bottleneck
```

```
        -cost_type total_victim_delay_bump
        -slack_lesser_than 2
        -max_nets 20
        -significant_digits 6
        -minimum_active_aggressors 1
        -max
Design : TestBH
****************************************

 Bottleneck Cost:  total_victim_delay_bump

 Net                          Scenario Name                    Cost
 ---------------------------------------------------------------------------
 OutFirstReg                       s1                        0.919074
 Reg                               s2                        0.639971
 PreNet                            s1                        0.367737
 Comb                              s4                        0.021904
 InReg                             s3                        0.000039
 InComb                            s2                        0.000033
```

report_timing

The `report_timing` command executed by the master process returns a merged report that eliminates redundant data across scenarios and sorts the data in order of slack, effectively treating the scenarios in the command focus as a single analysis. The merging process allows you to treat all the scenarios in command focus as if they are a single virtual PrimeTime instance. To do this, PrimeTime reports the worst paths from all scenarios while maintaining the limits imposed by the `-nworst`, `-max_paths`, and other options of the `report_timing` command. When the same path is reported from multiple scenarios, PrimeTime keeps only the most critical instance of that path in the merged report and shows the scenario in which that instance of the path was the most critical. This way, the resulting report is more evenly spread across the design instead of being focused on the one portion of the design that is critical in all scenarios.

When the same path is reported from multiple scenarios, PrimeTime keeps only the most critical instance of that path in the merged report and shows the scenario in which that instance of the path was the most critical. This way, the resulting report is more evenly spread across the design instead of being focused on one portion of the design that is critical in all scenarios.

Note:

You can disable this capability with the `-dont_remove_duplicates` option of the `report_timing` command. For more information, see “Handling Duplicate Paths” on page 1-40.

PrimeTime considers two paths from two scenarios to be instances of the same path if the two paths meet all of the following criteria:

- path group

- sequence of pins along the data portion of the path
- transitions at every pin along the data portion of the path
- launch clock
- capture clock
- constraint type

In the following example, a multi-scenario analysis was run with two scenarios, func_bc and func_wc. The following merged `report_timing` command was issued at the master:

```
pt_shell> report_timing -delay_type min -nworst 2 \
   -max_paths 2 -group mrx_clk_pad_I

Start of Master/Slave Task Processing
-------------------------------------
Started   : Command execution in scenario 'func_wc'
Started   : Command execution in scenario 'func_bc'
Succeeded : Command execution in scenario 'func_bc'
Succeeded : Command execution in scenario 'func_wc'
---------------------------------
End of Master/Slave Task Processing

  Startpoint: txethmac1/WillTransmit_reg
       (rising edge-triggered flip-flop clocked by mtx_clk_pad_i)
  Endpoint: WillTransmit_q_reg
       (rising edge-triggered flip-flop clocked by mrx_clk_pad_i)
  Path Group: mrx_clk_pad_i
  Path Type: min
  Scenario: func_bc
  Min Data Paths Derating Factor  : 1.00
  Min Clock Paths Derating Factor : 1.00
  Max Clock Paths Derating Factor : 1.05

  Point  IncrPath
  ------------------------------------------------------------------
  clock mtx_clk_pad_i (rise edge)              0.00         0.00
 library hold time -                           0.06         0.67
  data required time                                      0.67
  ------------------------------------------------------------------
  data required time                                      0.67
  data arrival time                                      -0.76
  ------------------------------------------------------------------
  slack (MET)                                             0.10

  Startpoint: txethmac1/WillTransmit_reg
       (rising edge-triggered flip-flop clocked by mtx_clk_pad_i)
  Endpoint: WillTransmit_q_reg
       (rising edge-triggered flip-flop clocked by mrx_clk_pad_i)
  Path Group: mrx_clk_pad_i
  Path Type: min
```

```
Scenario: func_wc
Min Data Paths Derating Factor  : 0.95
Min Clock Paths Derating Factor : 0.95
Max Clock Paths Derating Factor : 1.00

Point IncrPath
-----------------------------------------------------------------
clock mtx_clk_pad_i (rise edge)                    0.00      0.00
clock network delay (propagated)                   1.0      51.05
txethmac1/WillTransmit_reg/CP (_SDFCNQD1)          0.00       1.05 r
txethmac1/WillTransmit_reg/Q (_SDFCNQD1)           0.44       1.49 f
txethmac1/WillTransmit (eth_txethmac_test_1)       0.00       1.49 f
WillTransmit_q_reg/D (_SDFQD1                      0.00       1.49 f
data arrival time1.49

clock mrx_clk_pad_i (rise edge)                    0.00       0.00
clock network delay (propagated                    1.31       1.31
clock reconvergence pessimism                      0.00       1.31
WillTransmit_q_reg/CP (_SDFQD1)                    1.31 r
library hold time                                  0.06       1.37
data required time                                 1.37
-----------------------------------------------------------------
data required time                                 1.37
data arrival time                                 -1.49
-----------------------------------------------------------------
slack (MET)                                        0.12
```

For specific merge process examples, see the individual merging command man pages.

The following sections provide information on how to use the `report_timing` options to control the output reports.

Standard Option Handling

All of the options of the `report_timing` command are available for merged reporting, with a few exceptions (see "Limitations" on page 1-50). Provided collections are not being passed to the options, they can be specified at the master just as in normal PrimeTime as in the following examples:

```
pt_shell> report_timing -nworst 10
pt_shell> report_timing -nworst 10 -max_paths 15
pt_shell> report_timing -delay_type max_rise -path_type
full_clock -nosplit
pt_shell> report_timing -from UI/CP -rise_to U2/D
-significant_digits 5
```

As in the `remote_execute` command, to evaluate sub-expressions and variables remotely, generate the options using curly braces. For example,

```
report_timing -from {$all_in} -to {[all_outputs]}
```

In this example, the `report_timing` command is a merged reporting command which collates data from the slave and generates a merged report at the master. The `all_in` variable and the `all_outputs` expression are evaluated in a slave context.

To evaluate expressions and variables locally at the master, enclose the command string in quotes. All master evaluations must return a string. For example,

```
report_timing -to "$all_out"
```

Explicit and Implicit Collection Example

Use the `-pre_commands` option so the collection is generated in the same task as the `report_timing` command is executed. The merged `report_timing` command at the master then refers to the explicit collection using the name you specified.

Example 1-7 Explicit collection

```
pt_shell> report_timing
           -pre_commands {set start_pins [get_pins U1/*]}
           -from {$start_pins}
```

The implicit collection is referred to in the merged `report_timing` command at the master using curly braces around the slave expression. At the slave, the implicit collection is generated and passed to the `-from` option of the `report_timing` command.

Example 1-8 Implicit Collection

```
pt_shell> report_timing -from {[get_pins U1/A]}
```

Complex Option Handling

To allow for evaluating master and slaves variables and expressions from the multi-scenario master, the following options have been altered at the multi-scenario master:

```
-from        -rise_from        -fall_from
-through     -rise_through     -fall_through
-to          -rise_to          -fall_to
-exclude     -rise_exclude     -fall_exclude
```

Unlike in normal PrimeTime, where these variables accept a list as an argument, in multi-scenario merged reporting these variables accept a string as an argument. For example,

```
# In normal PrimeTime you would have:
report_timing -from {ffa ffb} # in this case, the argument
                              # to -from is a list.

# In multi-scenario merged reporting you would have:
report_timing -from {ffa ffb} # in this case, the argument
                              # to -from is treated as a
string.
```

Handling Duplicate Paths

The `-dont_merge_duplicates` option is available with the `report_timing` command only if you invoke PrimeTime with the `-multi_scenario` option. This option impacts the way in which paths from multiple scenarios are merged. By default, when the same path is reported from more than one scenario, PrimeTime reports only the single most critical instance of that path in the merged report and shows its associated scenario. When you use this option, PrimeTime does not merge duplicate instances of the same path into a single instance; however, it shows all critical instances of the path from all scenarios.

Note:

Since the number of paths reported is limited by the `-nworst`, `-max_paths`, and other options of this command, when you use the `-dont_merge_duplicates` option, the resulting merged report may not be evenly spread out across the design. The report may be focused on the portion of the design that is critical in each scenario.

Controlling Fault Handling

In distributed multi-scenario analysis, PrimeTime supports fault handling, which you can control by setting the `multi_scenario_fault_handling` variable to either `ignore` or `exit`. When the variable is set to `ignore` (the default) and a critical fault occurs in a slave process, the abnormally terminated scenarios and the scenarios that depend on those that terminated are excluded from further analysis within the context of the current session. A warning message is displayed showing which scenarios abnormally terminated. The session then proceeds to analyze the remaining scenarios that are in command focus. The following situations might occur:

- If all scenarios in command focus of the current session abnormally terminate, any subsequent distributed command issued at the master fails and you receive an error message explaining that there are no scenarios in command focus. You should change the command focus within the context of the current session and analyze the selected scenarios.
- If all scenarios in the current session abnormally terminate, the current session terminates and you receive a warning saying the session is removed. Any subsequent distributed commands issued at the master causes an error message explaining that there is no current session.
- Critical faults cause resources to go offline. If the resources available are reduced to the point where none are online, the session must terminate and you receive an error message explaining that the current session has terminated.

The `exit` option of the `multi_scenario_fault_handling` variable is available for backwards compatibility. When you select this option, if a critical fault occurs in a slave process, the active command completes its task and then the master process terminates the entire analysis. The process exits gracefully and an informational message is displayed explaining what occurred.

The following sections explain the other improvements to fault handling in distributed multi-scenario analysis.

- Merged Reporting Commands
- Netlist Editing Commands
- Using the remote_execute Command
- Other Commands

Merged Reporting Commands

Merged reporting is a mechanism that works with all the scenarios in command focus to eliminate redundant data automatically across scenarios and sort the data in order of criticality. This allows you to treat all the scenarios in command focus as if they are a single virtual PrimeTime instance. Due to the distributed feature, if one or more slave processes terminate abnormally, the relevant information for the reporting command is not available. Therefore, when a scenario terminates abnormally, you receive a warning message, and PrimeTime removes this scenario from the current session. PrimeTime then automatically reissues the reporting command to the remaining scenarios in command focus. Any reporting command issued afterwards is applied only to the remaining scenarios in command focus and any abnormally terminated scenarios are excluded from further analysis.

Netlist Editing Commands

Before a netlist change is committed to any scenario from the master, it must be possible to commit the change to all scenarios. Thus, PrimeTime first checks that the netlist change could be successful in all scenarios and next it commits the change. If a scenario terminates abnormally while PrimeTime is checking the possibility of successfully executing the netlist change, this scenario is removed from the command focus, and the netlist editing command is automatically reissued to the remaining scenarios in command focus. If all of the checking processes succeeded in all scenarios, but there was a scenario that terminated abnormally when PrimeTime went to commit the change, a warning message is issued, and all the remaining scenarios in command focus apply the netlist change. Subsequent netlist editing commands are applied only to the remaining scenarios in command focus and any abnormally terminated scenarios are excluded from further analysis.

Using the remote_execute Command

Distributed multi-scenario analysis supports execution of any number of commands on all scenarios in command focus by using the `remote_execute` command. If the `remote_execute` command encounters a scenario that terminated abnormally, a warning message is issued, the scenario is removed from the current session, and subsequent calls to the command apply only to the remaining scenarios in command focus. Abnormal termination of one or more scenarios has no impact on command execution in other scenarios.

Other Commands

The following commands do not come under the categories that have been listed, but they are handled in a similar way when an abnormal termination occurs.

- `get_distributed_variable` command

  When a scenario terminates abnormally, you receive a warning message, and the scenario is removed from the current session. PrimeTime then automatically reissues the command to the remaining scenarios in command focus.

- `get_timing_path` command

  When a scenario terminates abnormally, you receive a warning message, and the scenario is removed from the current session. PrimeTime then automatically reissues the command to the remaining scenarios in command focus.

- `save_session` command

  When a scenario terminates abnormally, you receive a warning message, and the scenario is removed from the current session. PrimeTime then automatically reissues the command to the remaining scenarios in command focus.

- `set_distributed_variable` command

  When a scenario terminates abnormally, you receive a warning message, and the scenario is removed from the current session. The command executes as normal for the remaining scenarios in command focus.

Messages and Log Files

Multi-scenario analysis has the potential to generate large amounts of data. To avoid overloading you with data at the master, all data is written to a set of log files. A limited set of interactive messages are displayed at the master.

The `remote_execute` command sends all data to the master terminal if the `-verbose` switch is used.

Interactive Messages

While a task is executing on the slaves, the slaves send messages back to the master to indicate their progress.

Progress Messages

As the slave progresses through each stage of the analysis, the slaves send confidence messages to update you on the progress of the analysis. For example,

```
pt_shell> current_session {scen1 scen2}
pt_shell> remote_execute {cmd1}

Start of Master/Slave Task Processing
-------------------------------------
Started   : Netlist image generation for session 'session'
Succeeded : Netlist image generation for session 'session'
Started   : Baseline image generation for scenario 'scen1'
Succeeded : Baseline image generation for scenario 'scen1'
```

User Control of Task Execution Status Messages

You can control the verbosity of distributed multi-scenario analysis task execution status messages. You can set the `multi_scenario_message_verbosity_level` variable to one of two values, default and low. When you set to the default value, all task execution status messages are displayed:

```
Start of Master/Slave Task Processing
-------------------------------------
Started    : Command execution in scenario 'wc'
Started    : Command execution in scenario 'bc'
Succeeded  : Command execution in scenario 'bc'
Succeeded  : Command execution in scenario 'wc'
---------------------------------
End of Master/Slave Task Processing
```

When you set the variable to low, only error, fatal, and failure messages are displayed. You might find this useful for Tcl scripts where more control over the output is desired. If this variable is changed by a Tcl script, it is desirable to change the variable back to its original value after completing the script or procedure to avoid confusion.

Error Messages

When an error occurs on a slave for a given scenario, the error is reported in full at the master together with the name of that scenario. For example,

```
pt_shell> current_session {scen1 scen2}
pt_shell> remote_execute {cmd1}

Start of Master/Slave Task Processing
-------------------------------------
Started   : Netlist image generation for session 'session'
Error    : Problem in read_verilog. No designs were read
(DBR-011) (default_session)
```

Warning Messages

When a warning occurs on a slave for a given scenario, the warning is reported at the master only if it is the first time that this warning has occurred in that scenario. For example,

```
pt_shell> current_session {scen1 scen2}
pt_shell> remote_execute {cmd1}

Start of Master/Slave Task Processing
-------------------------------------
Started   : Netlist image generation for session 'session'
Succeeded : Netlist image generation for session ‘session’
Started   : Baseline image generation for scenario 'scen1'
Warning   : RC-009 has occured on scenario 'scen1'
```

Log Files

You can examine all information generated by the slaves in the log, output log, command log, or merged error log file. The root working directory for all multi-scenario analysis data, including log files, is determined by the setting of the `multi_scenario_working_directory` variable. If you do not explicitly set this variable, it is set to the current directory (from which the PrimeTime master was invoked). You must have write permission to the working directory, and the directory must be accessible by the master and to all slaves.

Output Log

Each scenario has its own output log file which can be found in the scenario working directory. For example, for a scenario named s1, the log would be as follows:

```
multi_scenario_working_directory/s1/out.log
```

You cannot set or change the name of the output log. The output log for a particular scenario contains all errors, warnings and informational messages (all information that you would normally see in the terminal window of a conventional single analysis PrimeTime run).

Command Log

Each remote process has its own command log file that you can find in the `command_log` directory under the working directory. For example, for a process launched on a host named srv1 which is the sixth process to be launched, the command log would be in the following file:

```
multi_scenario_working_directory/command_log/srv1_6_pt_shell_command.log
```

You cannot set or change the name of the command log of remote processes. The command log for a particular process contains all commands executed on that process. You can locate the masters command log in the following file:

```
$sh_launch_dir/pt_shell_command.log
```

The `sh_source_logging` variable, when set to `true`, causes individual commands from sourced scripts to be written to the command log file.

Merged Error Log

Large numbers of errors, warnings, and informational messages can occur during a multi-scenario analysis. To help you debug these messages, you can set the merged error log variable at the master. For example, the following command creates a merged error file in `multi_scenario_working_directory/error.log`:

```
pt_shell> set multi_scenario_merged_error_log "error.log"
```

When this variable is set, all error messages from the start of a particular task to the end of the task are merged together. If the same error message occurs on a number of scenarios, the message is printed to the merged error file only once, with a list of the scenarios in which the message occurred.

Note:

You must set the `multi_scenario_working_directory` and `multi_scenario_merged_error_log` variables before issuing the `create_distributed_farm` command.

Command Output Redirection

Command output redirection is supported in multi-scenario analysis. Be sure to redirect output to different destination files for each process. Otherwise, an error occurs when two different processes attempt to control the same file at the same time.

Note:

When using output redirection, be sure use a relative path. Do not use an absolute path; if you do, all slave processes attempt to write to the same file, which could lead to read/write conflicts and cause the slave processes to fail.

As with the previous log file example, the value of the `multi_scenario_working_directory` variable is used along with the scenario name to determine the output destination file name. The following examples demonstrate the file naming conventions:

Example 1: Interactive redirection

```
pt_shell> set x myfile
pt_shell> remote_execute {report_timing} > $x.out
```

The output of the `report_execute` command is directed to a file named myfile.out in the current working directory. The $x variable is evaluated only at the master process.

Example 2: Script-based redirection

The `multi_scenario_working_directory` variable has been set to /rpt/new, and there are two scenarios in command focus, s3 and s4. The master issues a `remote_execute {"source myscript.tcl"}` command, which sources the Tcl script “myscript.tcl” in the scenarios in command focus. The script “myscript.tcl” contains the following line:

```
report_timing > rt.log
```

The output of the `report_timing` command goes into the following files:

```
/rpt/new/s3/rt.log
/rpt/new/s4/rt.log
```

Session Example

The following script is a typical multi-scenario master script.

Note:

You must set the `multi_scenario_working_directory` and `multi_scenario_merged_error_log` variables before issuing the `create_distributed_farm` command.

The .synopsys_pt_setup file in the directory where the master was launched (/rpt/new) contains the following settings:

```
set multi_scenario_working_directory "./ms_session_1"
set multi_scenario_merged_error_log  "merged_errors.log"
```

```
########################################
# Start of specifying the set-up
# Start of specifying the first task
########################################
set search_path "./scripts"

# Add 2 32 bit hosts to the pool of hosts for
# the distributed farm from hosts platinum1 and platiunm2
add_distributed_hosts -32bit -farm now platinum1
add_distributed_hosts -32bit -farm now platinum2

# Create the distributed farm for processing the scenarios
create_distributed_farm

# set the upper limit on the license usage
set_multi_scenario_license_limit -feature PrimeTime 2

create_scenario -name s1 \
  -common_data common_data_scenarios_s1_s3.tcl \
  -specific_data specific_data_scenario_s1.tcl

create_scenario -name s2 \
  -common_data  common_data_scenarios_s2_s4.tcl \
  -specific_data  specific_data_scenario_s2.tcl

create_scenario -name s3 \
  -common_data  common_data_scenarios_s1_s3.tcl \
  -specific_data  specific_data_scenario_s3.tcl

create_scenario -name s4 \
  -common_data  common_data_scenarios_s2_s4.tcl \
  -specific_data  specific_data_scenario_s4.tcl

##############################
# End of specifying the setup
##############################

current_session {s3 s4}

remote_execute -pre_commands {source contraint_group1.tcl\
   source constrint_group2.tcl} {report_timing}

quit
# analysis is finished
# all slaves are inactivated and licenses are returned
```

This is what the script does:

1. The initial `set` command sets the search path at the master (used for finding scripts).

2. The `add_distributed_hosts` command specifies that one host from each host, platinum1 and platinum2, is to be added to the distributed farm for running 32 bit processes.
3. The `create_distributed_farm` command starts the slave processes on the hosts specified by the `add_distributed_hosts` command.
4. The `set_multi_scenario_license_limit` command limits to two the number of PrimeTime licenses that the master uses for slave processes.
5. The `create_scenario` commands create four scenarios named s1 through s4. Each command specifies the common-data and specific-data scripts for the scenario.
6. The `current_session` command selects s3 and s4 for the current session, which is also the command focus by default. (You can use the `current_scenario` command to narrow the focus further.)
7. The `remote_execute -pre_commands` command begins the task generation process and execution of those tasks in scenarios s3 and s4. The resulting logs contain all the output from the commands that were executed.

   Because the `-pre_commands` option is specified, the two `source` commands are executed before the `report_timing` command. The command list can contain any valid PrimeTime commands or sourcing of scripts that contain valid PrimeTime commands (excluding all commands that alter the netlist or save or restore).
8. The quit command terminates the slave processes and master processes in an orderly manner and returns all the checked-out PrimeTime licences back to the central license pool.

At the end of this example session, you would find the report files and log files in the full file paths listed below (assuming the master pt_shell was launched from the directory `/rpt/new`).

- Master command log:

  ```
  /rpt/new/pt_shell_command.log
  ```

- Errors generated by the slaves and returned to the master and merged for reporting purposes:

  ```
  /rpt/new/ms_session_1/merged_errors.log
  ```

- Slave output log for work done for the session, that is, netlist image generation:

  ```
  /rpt/new/ms_session_1/default_session/out.log
  ```

- Slave output logs generated for s1 and s2:

  ```
  /rpt/new/ms_session_1/s3/out.log
  /rpt/new/ms_session_1/s4/out.log
  ```

- Slave command logs:

```
/rpt/new/ms_session_1/pt_shell_command_log/
   platinum1_pt_shell_command.log
/rpt/new/ms_session_1/pt_shell_command_log/
   platinum2_pt_shell_command.log
```

Modeling Support

Distributed multi-scenario analysis supports interface logic models and extracted timing models without your having to perform any special work for setup. Additionally, you should not have to change any modeling arrangements made to your design. For more information about modeling support in PrimeTime, see the *PrimeTime Modeling User Guide*.

Activating CCS Timing Voltage Scaling

A scenario is a specific combination of operating condition and operating mode for a given configuration. You create a scenario with the `create_scenario` command, as in the following example:

```
pt_shell> create_scenario -name scen1 \
            -common_data {common_s.pt} \
            -specific_data {specific1.pt}
```

This command specifies a scenario name, the common-data script files, and the specific-data script files.

Using Common or Specific Script Scenarios

When you set up scenarios using common or specific scripts (`-common_scripts` or `-specific_scripts` options, respectively), you must create voltage scaling groups just before running the `link_design` in one of the scripts used to set up a scenario.

If voltage scaling groups are created in a script that is part of a scenario's `-specific_scripts` specification, only that scenario sees those scaling groups. If voltage scaling groups are created in a script that is part of a scenario's `-common_scripts` specification and another scenario shares the same common specification, both of those scenarios see the same voltage scaling groups.

Creating a New Scenario With a Previously Saved Scenario

If you specified voltage scaling groups in a single scenario analysis and ran the `save_session` command, those voltage scaling groups are available to any scenario using that saved session. To access the saved scenario, use the `create_scenario -image` command.

Limitations

The distributed multi-scenario feature has the following known functional limitations and issues:

- Maximum number of slave processes

  Currently, the maximum number of slave processes that the master process can control 256 concurrent hosts.

- Number of allocated licenses

  You cannot reduce the number of allocated licenses after they have been allocated.

- No support for the `-true` or `-justify` options of the merged `report_timing` command

2

Distributed Multi-Scenario Analysis Netlist Editing Support

PrimeTime provides commands for editing the netlist. Editing the netlist is performed to address timing issues without having to iterate through logic synthesis.

The netlist editing capability is described in the following sections:

Introduction

PrimeTime distributed multi-scenario analysis feature provides the ability to make incremental changes to the netlist, such as `size_cell` and `insert_buffer`. It also enables you to analyze the circuit timing in the presence of these changes. This feature is important in the work flow for creating, evaluating, and implementing engineering change orders (ECOs). The distributed multi-scenario analysis flow supports the use of all netlist editing commands (except for `swap_cell`). Using this feature you can quickly evaluate the effects of different netlist edits and obtain merged reporting on the tested scenarios.

You can execute any number of netlist editing commands using a varying command focus. Therefore, distributed multi-scenario analysis supports execution of netlist editing commands on a per-scenario basis using the `remote_execute` command. The execution speed of these operations is highly optimized. Netlist editing command execution is also supported directly at the master; however, complex nested command structures are not supported in this mode of operation.

Editing the Netlist

The PrimeTime distributed multi-scenario analysis supports the use of the following netlist editing commands.

Table 2-1 Netlist Editing Commands

Object	Task	Command
Buffer	Inserting a buffer	`insert_buffer`
	Removing a buffer	`remove_buffer`
Cell	Changing the size (or drive strength) of a cell	`size_cell`
	Creating a cell	`create_cell`
	Renaming a cell	`rename_cell`
	Removing a cell	`remove_cell`
Net	Adding a new net to the design	`create_net`
	Connecting nets to pins in the design	`connect_net`
	Disconnecting nets from pins in the design	`disconnect_net`
	Renaming a net	`rename_net`
	Removing a net	`remove_net`

The `swap_cell` netlist editing command is not supported at the master in distributed multi-scenario analysis, but can execute through the`remote_execute` command. For more information about netlist editing in PrimeTime, see *PrimeTime Advanced Timing Analysis User Guide*.

Replacement Cell Lookup

When replacing or sizing a cell in the current design, the PrimeTime applies the change to all scenarios in the current focus. Therefore, it is most useful to determine the alternative library cells you can use to replace a cell on all those scenarios, excluding library cells that you can only use as a replacement cell on a subset of the scenarios in the current design. By executing the `get_alternative_lib_cells -base_names` command in a master context, PrimeTime returns a list of library cell base names. You can use any library cell base name with the `size_cell` command to replace the existing cell.

```
pt_shell> get_alternative_lib_cells -base_names u1 \
           {"AN2", "AN2i"}
```

PrimeTime returns library cells in the order in which they are found according to the same search order that exists for the `size_cell`, `insert_buffer`, and `create_cell` commands. For more information about the order in which libraries are searched, see *PrimeTime Advanced Timing Analysis User Guide*.

Supporting ECO Commands

You can execute the following commands in a slave process using the `remote_execute` command and apply them to all the scenarios in the command focus:

- `set_coupling_separation`
- `remove_coupling_separation`
- `read_parasitics -eco`

The `set_coupling_separation` command in distributed multi-scenario analysis allows you to create a separation constraint on nets in all the scenarios in command focus. Additionally, you can remove the coupling separation from all scenarios in command focus using the `remove_coupling_separation` command.

The `-eco` option to the `read_parasitics` command indicates the files you are annotating are ECO parasitics from only StarRC. In distributed multi-scenario analysis, you can use this option with the `read_parasitics` command to apply the feature to all scenarios in command focus.

Note:

In the slave context, the `get_alternative_lib_cells` and `report_alternative_lib_cells` operate the same as these commands would when in the single-core analysis mode of PrimeTime.

A

Master Command List

This appendix lists all of the commands that you use during a distributed multi-scenario analysis.

Note:
All regular Tcl commands are supported.

These commands are broken down into several sections within this appendix.

Configuration and Setup Commands

Table A-1 lists the commands to use to create and remove your multi-scenario configurations and scenarios.

Table A-1 Configuration and Setup Commands

Command	Description
`create_scenario`	Creates a single scenario. This command is required.
`-name [scenario_name]`	A unique string used to identify and refer to a single scenario.
`-common_data {file_list}`	A list of files that contains commands and data that is common across scenarios. These files and the netlist image are used to generate the baseline image for the scenario.
`-specific_data {file_list}`	A list of files that contains commands and data that is specific to the scenario being created.
`-common_variables {list}`	A list of variables that is common across scenarios.
`-specific_variables {list}`	A list of variables that is specific to the scenario being created.
`-image`	Restores a PrimeTime session.
`remove_scenario {scenario_list}`	Removes one or more scenarios which are given as a list to the command.
`remove_multi_scenario_design`	Removes all multi-scenario data specified, including the session, all the scenarios, and the configuration.
`report_multi_scenario_design`	Reports on the current user-defined multi-scenario analysis.
`-configuration`	Reports on the current configuration.
`-scenario`	Reports on the current scenario.
`-session`	Reports on the current session.
`-license`	Reports on the current license.

Process Setup Commands

Table A-2 lists the commands to use to add and remove distributed processes.

Table A-2 Process Setup Commands

Command	Description
`add_distributed_hosts`	Adds distributed hosts to the farm's pool. This command is required.
`[-32bit]`	Specifies the type of executable to be used.
`-farm`	Specifies the type of farm that is the source of the distributed hosts. The types must be one of `lsf`, `grd`, `generic`, or `now`.
`[-setup_path setup path]`	Specifies the path where submission scripts for grd, lsf or generic reside. LSF—the setup path should contain "bsub", "bkill" GRD— the setup path should contain "qsub", "qdel" Generic—the setup path should contain "gsub", "gdel"
`[-num_of_hosts` *`count`*`]`	Specifies the number of distributed hosts to use. If this is not specified one is assumed.
`[-submission_script]`	You can explicitly specify the script to call when submitting jobs to farm types grd, lsf or generic using this option. When both the *-setup_path* and *-submission_script* options are used simultaneously, the path and submission Sept. specification are combined to define the script used when submit jobs to the specified computing resource. The *-submission_script* option is not valid with *-farm now*.
`[-options]` *`string`*	Specifies the options to be provided to the *submission_script*.
`[workstation_name]`	Specifies the name of a work station (in now).
`remove_distributed_hosts`	Removes all distributed hosts added to the farm's pool.
`create_distributed_farm`	Starts the slave processes on the hosts specified by the `add_distributed_hosts` command.

License Setup Command

Table A-3 shows the command to use to set the number of licenses the master can use to control the slave sessions.

Table A-3 License Setup Command

Command	Description
`set_multi_scenario_license_limit`	Specifies the number of licenses the master can use for slave processes. This command is required.
`number`	An integer greater than zero (0). This number should be equal to the number of slave processes you wish to use.
`-feature` *`feature`*	String consisting of the feature name. Values are `PrimeTime`, `PrimeTime-SI`. For example, `-feature PrimeTime 4` checks out four PrimeTime licenses from the global license pool for slave usage.
`-force_checkout`	Forces licenses to be checked out immediately.

Analysis Focus Commands

Table A-4 lists the commands to use to set and remove sessions and scenarios that interpret commands from the master process.

Table A-4 Analysis Focus Commands

Command	Description
`current_session`	Puts a particular group of scenarios in focus for analysis. The scenarios in focus receive and interpret commands from the master. Returns a list of all scenarios in the current session. This command is required.
`-all`	Adds all previously created scenarios to the current scenario.

Table A-4 Analysis Focus Commands (Continued)

Command	Description
`{scenario_list}`	A list of the names of scenarios to put in the current session. All scenarios listed must have previously been created.
`remove_current_session`	Removes the previously set session. Defocuses all scenarios. Scenarios stay in memory until removed by the `remove_scenario` or quit command.
`current_scenario`	Changes the command focus to the selected scenarios within the current session. Returns a list of all scenarios in command focus.
`{scenario list}`	A list of scenario names in the current session.
`[-all]`	Brings the focus back to all of the scenarios specified with `current_session`.

Slave Context Commands

Table A-5 lists the commands to use to create a command buffer from the master to the slave processes. You can run these commands in batch or interactive mode.

Table A-5 Slave Context Commands

Command	Description
`remote_execute`	Batch or interactive mode: From the master, creates a command buffer to be executed in the slave context.
`-pre_commands {slave_cmd_string}`	Specifies string of semicolon-delimited commands to be executed in slave context before the execution of the default command string. The maximum size of a command is 1000 characters.
`-post_commands {slave_cmd_string}`	Specifies a string of semicolon-delimited commands to be executed in slave context. The maximum size of a command is 1000 characters.
`-verbose`	Returns all data produced at the slaves to the master terminal.

Distributed Multi-Scenario Analysis report_timing Options

Table A-6 lists `report_timing` options you can use for multi-scenario reporting.

Table A-6 report_timing Options

Option	Description
`-dont_merge_duplicates`	Turns off the merging of duplicate paths in the timing report. By default, when the same path is reported from more than one scenario, PrimeTime reports only the single most critical instance of that path in the merged report and shows its associated scenario. By using this option, PrimeTime does not merge duplicate instances of the same path into a single instance, but instead shows all critical instances of the path from all scenarios. Since the number of paths reported is limited by the `-nworst`, `-max_paths`, and other options of this command, the resulting merged report, when this option is used, may not be evenly spread out across the design, but instead may be focused on the portion of the design that is critical in each scenario.
`-pre_commands`	Specifies string of semicolon-delimited commands to be executed in slave context before the execution of the `merged_reporting` command. The maximum size of a command is 1000 characters.
`-post_commands`	Specifies string of semicolon-delimited commands to be executed in slave context before after the execution of the `merged_reporting` command. The maximum size of a command is 1000 characters.

Reporting Variables

Table A-7 lists variables to use for multi-scenario reporting.

Note:
You must set the `multi_scenario_working_directory` and `multi_scenario_merged_error_log` variables before creating the distributed farm.

Table A-7 Reporting Variables

Variable	Description
`multi_scenario_merged_error_log`	Default value is `""`. Note: Must be specified before issuing the `create_distributed_farm` command.
`multi_scenario_merged_error_limit`	Default value is 100. Specifies the maximum number of errors of a particular type to be written to the command log on a per-task basis.
`multi_scenario_working_directory`	Default value is `cwd`. Set the working directory to a directory which can be seen by all slaves and the master and to which you have write permissions. Note: Must be specified before issuing the `create_distributed_farm` command.

`pt_shell_mode`	Specifies the working mode: `primetime`, `primetime_master`, or `primetime_slave`.
	This variable is read-only
	You can use this variable in your scripts to determine if you are operating in a normal PrimeTime session or on the master or slave. For example, to block warnings about the working directory, you could use the following:
	`if {$pt_shell_mode == "primetime_master"} {`
	`  set_multi_scenario_working_directory "/home/wd"`
	`}`

Disallowed Commands

Some commands are not allowed at certain stages of the flow. For example, if during the analysis, you execute the `remove_design` command either in a script or at the command line, an error condition results. Table A-8 lists the commands that you cannot run on the slave processes.

Table A-8 Disallowed Commands for Slave Processes

Command	Description
`remove_design`	Removes one or more designs from memory.
`rename_design`	Renames a design.
`quit`	Quits `pt_shell`.
`exit`	Exits `pt_shell`.
`restore_session`	Restores a `pt_shell`. session.

Table A-9 lists the commands that you cannot run on the master process.

Table A-9 Disallowed Commands for the Master Process

Command	Description
`remove_design`	Removes one or more designs from memory.
`rename_design`	Renames a design.
`swap_cell`	Swaps a cell with a new design or lib cell.
`restore_session`	Restores a `pt_shell`. session.

B

Procedure for Filing STARs for Multi-Scenario Analysis

This appendix describes the procedure for filing Synopsys Technical Action Requests (STARs) for multi-scenario analysis problems or issues.

This appendix contains the following sections:

Problems and Issues

Problems and issues in multi-scenario may be difficult to reproduce reliably. It is extremely important to provide comprehensive supporting information when filing a multi-scenario STAR. If you do not provide the information requested, then problem resolution becomes very difficult, resulting in longer turn-around times for filed issues. When diagnosing and filing the issues, follow the steps described in this appendix.

Initial Problem Diagnosis

The purpose of this procedure is to determine if the issue is a single-core analysis or a multi-scenario analysis specific problem.

To diagnose the problem:

1. Try a single-core analysis PrimeTime run with your command set. Do this by running a script file that calls each of the common and specific scenario script components that make up the scenario:
   - source read_netlist.tcl
   - source libs.tcl
   - source operating_conditions.tcl
2. If the problem occurs in single-core analysis mode, follow the normal STAR filing procedure.
3. If the problem does not occur in single-core analysis mode, follow the steps for either a large design (more than 200,000 instances), or a small design (fewer than 200,000 instances).

Diagnosing Large Test Cases

To diagnose problems for large test cases (more than 200,000 instances):

1. Try to establish if this is a transient (intermittent) or reliably reproducible problem.
2. Try to reproduce the problem in multi-scenario using the provided demo design. To download the demo design, see the following SolvNet article:

   https://solvnet.synopsys.com/retrieve/014511.html
3. If the problem is transient, see “Transient Issues” on page B-3.
4. If the problem occurs every time, see “Reproducible Issues” on page B-4.

5. If you are unable to reproduce the issue at all, then document as much as you can about the problem including the hardware setup (see “Other Required Information” on page B-4), and go back to “Initial Problem Diagnosis” on page B-2 and try the same process with your original design while trying to simplify the test case as much as possible.
   - Remove all commands after the problem command, so that the last command received is the one that causes the issue.
   - Remove any other superfluous commands that do not affect the outcome you are observing.
6. Package the test case and any additional information and file a STAR using the normal procedure.

Diagnosing Small Test Cases

To diagnose problems for small test cases (fewer than 200,000 instances):

1. Try to establish if this is a transient or reproducible problem.
2. Using your test case, try to reproduce the problem in multi-scenario mode.
3. If the problem is transient, see “Transient Issues.”
4. If the problem occurs every time, see “Reproducible Issues” on page B-4.
5. If you are unable to reproduce the issue at all, document as much as you can about the problem including the hardware setup (see “Other Required Information” on page B-4).

Transient Issues

To diagnose transient issues:

1. Try to establish a pattern and record the following data:
   - Describe how often and under what conditions the problem occurred.
   - Are there any other import factors that may be affecting your network?
2. Generate the information listed in the “Other Required Information section.
3. Package the test case and all the additional information requested, then file a STAR using the normal procedure.

Reproducible Issues

If your issue is reproducible, record the conditions as follows:

- Describe under what system conditions the problem occurred.
- Generate the information listed in the "Other Required Information section.
- Package the test case and all the additional information requested and file a STAR using the normal procedure.

Other Required Information

The following sections describe the other types of information that you must gather before filing a multi-scenario STAR.

Hardware Setup and Loading Conditions

To gather the hardware setup and loading information for your STAR:

1. Define what hardware configuration you are using.
2. Specify machine use (master, slave, or both).
3. Provide data on your computer using the following commands:
   - `uname -aX > snps_ms_hw_uname.txt`
   - `sysinfo > snps_ms_hw_sysinfo.txt`
4. Collect the following required information:
   - Machine manufacturer
   - Number of hosts
   - Amount of RAM
   - Amount of swap space
5. Provide data on the loading in your hardware environment. This can be captured with the UNIX `top` command.

Multi-Scenario Environment

Provide the following information about your multi-scenario environment:

1. Your multi_scenario_merged_error_log file.
2. Your .synopsys_pt.setup file used at the master launch point or the your .synopsys_pt.setup file from your home account or both.

Your Environment

Provide the following information about your environment:

1. A text file of all the environment variable settings:
   - `printenv > snps_ms_printenv.log`
   - `set > snps_ms_set.log`
   - `setenv > snps_ms_setenv.log`
2. A copy of your .cshrc file.

C

Differences Between Single-Core and Multi-Scenario Constraint Reports

This appendix lists the differences in the report output for single-core and distributed multi-scenario analysis for constraint reports. It describes the different types of constraints and reporting styles available for the `report_constraint` command.

Constraint Merging Operations

There are four fundamental operations performed to gather constraint information for merging reports. Table C-1 describes how these operations are currently defined in single-core analysis mode and how they are supported in the multi-scenario mode of PrimeTime.

Table C-1 Constraint Merging Operation

Name	Single Scenario Analysis	Distributed Multi-Scenario Analysis
Constraint summation	Add all weighted constraint costs of a particular constraint together.	Add the worst constraint cost at each violating endpoint, pin, and net over all scenarios in command focus. When multiple scenarios have the same violating endpoints, pins, or nets, use the worst slack of these scenarios.
Worst constraint per group	Find the worst cost per path group of a particular constraint.	Find the worst cost per path group of a particular constraint across all scenarios. Groups are considered duplicate if they have the same name. If there is a tie between scenarios, the scenario name is used as a tie-breaker
Worst constraint	Find the worst cost of a particular constraint.	Find the worst cost of a particular constraint across all scenarios. If there is a tie between scenario names, group names, or both, the group name and then the scenario name is used as tie-breakers.
Sort constraints	Sort all violators of a particular constraint from best to worst.	Sort all violators of a particular constraint across all scenarios from best to worse. For each constraint type remove any duplicate constraints. A constraint is considered duplicate if the same constraint is specified on the same pin. Two pins are considered duplicate if they have the same leaf name and hierarchy. For example, a/b/c, d, a/b, c/d are different. If there is a tie between constraints, the scenario name is used as the tie-breaker

Printing Styles

There are a number of printing styles you can specify to write out an individual constraint report. Table C-2 describes how these printing styles are currently defined in the single-core analysis mode and how they are defined in the multi-scenario mode of PrimeTime.

Table C-2 Printing Styles

Printing Style	Single Scenario Analysis	Distributed Multi-Scenario Analysis
Group summary	Group (max_delay/setup) Cost Weight Weighted Cost CLK 0.00 1.00 0.00	Group (max_delay/setup) Cost Weight Weighted Cost CLK 0.00 1.00 0.00
Constraint summary	Constraint Cost max_delay/setup 2.46 (VIOLATED)	Constraint Cost max_delay/setup 2.46 (VIOLATED)
Object summary	Pin pulse width pulse width Slack nand1/Z 10.00 9.58 -0	Pin Scenario pulse width pulse width Slack nand1/Z scen1 10.00 9.58 -0

Table C-2 Printing Styles (Continued)

Printing Style	Single Scenario Analysis	Distributed Multi-Scenario Analysis
Path verbose	Startpoint: c (input port) Endpoint: z3 (output port) Path Group: default Path Type: max Point Incr Path --- input external delay 0.00 0.00 r c (in) 0.00 0.00 r U6/Z (IV) 1.34 1.34 f U2/Z (NR2) 3.35 4.69 r U15/Z (AO7) 0.87 5.56 f U24/Z (AO3) 1.02 6.57 r z3 (out) 0.00 6.57 r data arrival time 6.57 max_delay 6.50 6.50 output external delay 0.00 6.50 data required time 6.50 --- data required time 6.50 data arrival time -6.57 --- slack (VIOLATED) -0.07	Startpoint: c (input port) Endpoint: z3 (output port) Path Group: default Scenario: scen1 Path Type: max Point Incr Path --- input external delay 0.00 0.00 r c (in) 0.00 0.00 r U6/Z (IV) 1.34 1.34 f U2/Z (NR2) 3.35 4.69 r U15/Z (AO7) 0.87 5.56 f U24/Z (AO3) 1.02 6.57 r z3 (out) 0.00 6.57 r data arrival time 6.57 max_delay 6.50 6.50 output external delay 0.00 6.50 data required time 6.50 --- data required time 6.50 data arrival time -6.57 --- slack (VIOLATED) -0.07
Path slack only summary	min_delay/hold ('CLK' group) Endpoint Slack ffd/CR -0.40 (VIOLATED)	min_delay/hold ('CLK' group) Endpoint Scenario Slack ffd/CR scen1 -0.40 (VIOLATED)
Object verbose	Net: a max_fanout 5.00 - Fanout 7.00 Slack -2.00 (VIOLATED)	Net: a Scenario: scen1 max_fanout 5.00 - Fanout 7.00 Slack -2.00 (VIOLATED)

Reporting Modes

Table C-3 classifies the constraints into different constraint types. Each constraint type has several reporting modes depending on the `-all_violators`, `-path_type`, and `-verbose` options that you specify. Each reporting mode is associated with a particular merging operation (see Table C-1) and printing format (see Table C-2).

Table C-3 Reporting Modes

Constraint Type	Constraints	Report Mode
Connection class	`connection_class`	Summary • Constraint summation • Constraint summary Verbose • Worst constraint • Object verbose All violators • Sort constraints • Object summary All violators, verbose • Sort constraints • Object verbose
Design rule checking	• `max_capacitance` • `min_capacitance` • `max_transition` • `min_transition` • `max_fanout` • `min_fanout` • `max_area`	Summary • Constraint summation • Constraint summary Verbose • Worst constraint • Object verbose All violators • Sort constraints • Object summary All violators, verbose • Sort constraints • Object verbose

Table C-3 Reporting Modes (Continued)

Constraint Type	Constraints	Report Mode
Miscellaneous timing constraints	• `min_pulse_width` • `min_period` • `max_skew` • `clock_separation`	Summary • Constraint summation • Constraint summary Verbose • Worst constraint • Object verbose All violators • Sort constraints • Object summary All violators, verbose • Sort constraints • Path verbose
Timing path	• `setup` • `hold` • `clock_gating_setup` • `clock_gating_hold` • `recovery` • `removal`	Summary • Worst constraint per group (`clock_gating_setup`, `setup`, and `recovery`) • Constraint summation (`clock_gating_hold`, `hold`, and `removal`) • Group summary • Constraint summary Verbose • Worst constraint per group • Path verbose All violators • Sort constraints • Path slack only summary All violators, verbose • Sort constraints • Path verbose All violators, `path_type_end` • Sort constraints • Object summary

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V