

Distributed Processing of Ambit[®] BuildGates[®] Synthesis

**Product Version 4.0.8
May 2001**



© 1999-2000 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 1-800-862-4522.

All other trademarks are the property of their respective holders.

Restricted Print Permission: This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

<u>Preface</u>	6
<u>About This Manual</u>	6
<u>Other Information Sources</u>	6
<u>Syntax Conventions</u>	7
<u>Text Command Syntax</u>	7
<u>About the Graphical User Interface</u>	8
<u>Using Menus</u>	8
<u>Using Forms</u>	9
<u>1</u>	
<u>Concepts</u>	10
<u>Distributed Synthesis</u>	10
<u>Phases of Distribution</u>	12
<u>Hosts</u>	13
<u>Jobs</u>	14
<u>Job Types</u>	14
<u>Job Files</u>	15
<u>Job Manager</u>	15
<u>2</u>	
<u>Using Distributed Synthesis</u>	18
<u>Distributed Synthesis Options</u>	18
<u>Setting Up a Host List</u>	19
<u>Setting Up LSF Batch</u>	21
<u>Setting Global Variables</u>	22
<u>Setting Global Variables from the Command Line</u>	22
<u>Setting Global Variables with the GUI</u>	23
<u>Starting the Distributed Synthesis Run</u>	26
<u>Monitoring the Distributed Synthesis Run</u>	27
<u>Tracking the Run Time of Remote Jobs</u>	27

Distributed Processing of Ambit BuildGates Synthesis

<u>Monitoring the Distributed Synthesis Run with the Log File</u>	29
<u>Monitoring the Distributed Synthesis Run with the GUI</u>	29
<u>64-bit Support for Distributed Synthesis</u>	37
<u>Job Attribute Option -bits</u>	38
<u>Global Variable dist bits</u>	39
<u>Host Configuration Item -bits</u>	39
<u>Host Item kind</u>	40
<u>Host Selection</u>	41
<u>Job Attribute -rbits</u>	41
<u>Launch Mode host list vs batch</u>	42
<u>Special Command Options</u>	42
<u>Added Commands</u>	42
<u>Some Examples for Three Different Machines, After Executing set host config auto</u>	43
<u>Example of Using Global Variable dist bits and set dist bits</u>	44
<u>Renamed and Obsoleted Commands</u>	44
<u>Renamed Commands</u>	45
<u>Obsoleted Commands</u>	45

3

<u>Restarting Distributed Synthesis Jobs</u>	46
<u>Restart Feature</u>	46
<u>Global Variable dist restart signal</u>	46
<u>Global Variable dist restart embargo</u>	47
<u>Global Variable dist restart delay</u>	47
<u>Global Variable dist max restarts</u>	48
<u>Restart Script</u>	48
<u>Command check restart</u>	49
<u>Catchable Signals</u>	50
<u>Examples of check restart</u>	51
<u>Limiting Distributed Synthesis Jobs</u>	53
<u>Global Variable dist rlimit</u>	54
<u>Global Variables set dist rlimit and reset dist rlimit</u>	54
<u>Command check rlimit</u>	55

Distributed Processing of Ambit BuildGates Synthesis

A

<u>Testing the Distributed Synthesis Setup</u>	57
<u>Testing Host Setup with rac_shell</u>	57
<u>Syntax</u>	57
<u>Arguments</u>	58
<u>Return Values</u>	58
<u>The rac_shell Script Tests</u>	58
<u>Debugging Test Failures</u>	59
<u>Testing LSF Batch Setup with rac_shell</u>	60

Preface

This preface contains the following sections:

- [About This Manual](#) on page 6
- [Other Information Sources](#) on page 6
- [Syntax Conventions](#) on page 7
- [About the Graphical User Interface](#) on page 8

About This Manual

This manual provides a conceptual overview of distributed synthesis and tells you how to use multiple computers as a single processing system. To use this manual, you should be familiar with the Ambit® BuildGates® synthesis tool.

Other Information Sources

For more information about Ambit BuildGates synthesis and other related products, you can consult the sources listed here.

- [*Command Reference for Ambit BuildGates Synthesis and Cadence PKS*](#)
- [*Ambit BuildGates User Guide*](#)
- [*Timing Analysis for Ambit BuildGates Synthesis and Cadence PKS*](#)
- [*Test Synthesis for Ambit BuildGates Synthesis and Cadence PKS*](#)
- [*HDL Modeling for Ambit BuildGates Synthesis*](#)
- [*Constraint Translator for Ambit BuildGates Synthesis and Cadence PKS*](#)

Depending on the product licenses your site has purchased, you could also have these documents.

- [*PKS User Guide*](#)
- [*Datapath Option of Ambit BuildGates Synthesis and Cadence PKS*](#)

Distributed Processing of Ambit BuildGates Synthesis

Preface

■ *Low Power Option of Ambit BuildGates Synthesis and Cadence PKS*

BuildGates synthesis is often used with other Cadence® tools during various design flows. The following documents provide information about these tools and flows. Availability of these documents depends on the product licenses your site has purchased.

- *Cadence Timing Library Format Reference*
- *Cadence Pearl Timing Analyzer User Guide*
- *Cadence General Constraint Format Reference*

The following books are helpful references.

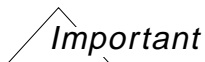
- IEEE 1364 Verilog HDL LRM
- TCL Reference, *Tcl and the Tk Toolkit*, John K. Ousterhout, Addison-Wesley Publishing Company

Syntax Conventions

This section provides the Text Command Syntax used in this document.

Text Command Syntax

The list below describes the syntax conventions used for the Ambit BuildGates synthesis text interface commands.



Command names and arguments are case sensitive. User-defined information is case sensitive for Verilog designs and, depending on the value specified for the global variable `hdl_vhdl_case`, may be case sensitive as well.

<code>literal</code>	Nonitalic words indicate keywords that you must enter literally. These keywords represent command or option names.
<code>argument</code>	Words in italics indicate user-defined arguments or information for which you must substitute a name or a value.
	Vertical bars (OR-bars) separate possible choices for a single argument.

Distributed Processing of Ambit BuildGates Synthesis

Preface

[]	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices from which you can choose one.
{ }	Braces are used to indicate that a choice is required from the list of arguments separated by OR-bars. You must choose one from the list. <code>{ argument1 argument2 argument3 }</code>
{ }	Bold braces are used in Tcl commands to indicate that the braces must be typed in literally.
...	Three dots (...) indicate that you can repeat the previous argument. If the three dots are used with brackets (that is, <code>[argument]. . .</code>), you can specify zero or more arguments. If the three dots are used without brackets (<code>argument . . .</code>), you must specify at least one argument, but can specify more.
#	The pound sign precedes comments in command files.

About the Graphical User Interface

This section describes the conventions used for the BuildGates synthesis graphical user interface (GUI) commands and describes how to use the menus and forms in the BuildGates synthesis software.

Using Menus

The GUI commands are located on menus at the top of the window. They can take one of three forms.

<i>CommandName</i>	A command name with no dots or arrow executes immediately.
<i>CommandName...</i>	A command name with three dots displays a form for choosing options.
<i>CommandName -></i>	A command name with a right arrow displays an additional menu with more commands. Multiple layers of menus and commands are presented in what are called command sequences, for example: <i>File – Import – LEF</i> . In this example, you go to the File menu, then the Import submenu, and, finally, the LEF command.

Using Forms

...	A menu button that contains only three dots provides browsing capability. When you select the browse button, a list of choices appears.
Ok	The <i>Ok</i> button executes the command and closes the form.
Cancel	The <i>Cancel</i> button cancels the command and closes the form.
Defaults	The <i>Defaults</i> button displays default values for options on the form.
Apply	The <i>Apply</i> button executes the command but does not close the form.
Help	The <i>Help</i> button provides information about the command.

Concepts

This chapter discusses the following topics:

- [Distributed Synthesis](#) on page 10
- [Hosts](#) on page 13
- [Jobs](#) on page 14

Distributed Synthesis

Envisia™ distributed synthesis lets you use a network of heterogeneous computers as a single processing system for synthesis. This increases the processing power available for a synthesis run. It reduces compute time and provides you with results more quickly.

The distributed synthesis tool performs parallel optimization of design modules. It achieves parallel optimization by executing remote `ac_shell` processes on discrete CPUs, or hosts. The tool partitions a design into jobs that can execute in parallel, and it distributes the jobs to hosts based on the current load conditions caused by other applications that are running on each host. Remote jobs behave in the same manner as jobs that run locally.

The amount of run-time speedup that you can achieve depends upon several factors:

- Width and height of the design hierarchy
- Critical path interaction when the design is “sliced” for distributed synthesis
- Amount of module reuse
- Availability of hosts and `Ambit_BuildGates` or `Envisia_PKS` licenses

You can use the tool with a synthesis database that contains RTL, gates, or a mixture of generic and gate-level modules. You cannot use the tool for postlayout optimization, because interconnect arcs across module boundaries may not be preserved in the process of creating Ambit synthesis database (ADB) files for the remote jobs.

Distributed Processing of Ambit BuildGates Synthesis Concepts

Before you start distributed synthesis, refer to [Setting Up a Host List](#) on page 19 or [Setting Up LSF Batch](#) on page 21. For a detailed description of how distributed finds out if a host is set up properly to run distributed synthesis, refer to [Testing the Distributed Synthesis Setup](#) on page 57.

The distributed synthesis tool performs the following steps when you call a synthesis command in distributed mode:

1. It walks through the design hierarchy and creates a dependency graph based on the design hierarchy. The dependency graph shows which jobs must be executed first, and which jobs depend on the results of lower-level jobs.

You can control how all or part of the design is partitioned for distribution rather than letting the distributed synthesis tool partition the design, by calling the `set_dist_point` command, as follows:

```
set_dist_point on -hier module
```

This optional command specifies that *module* and its child modules should be treated as a single distribution point. The distributed synthesis tool defines the distribution points for the rest of the design.

2. It creates a list of jobs, called a run list, and it sorts the list based on job size. Large jobs run longer, and therefore, start earlier. If two jobs have the same size, the job further down in the design hierarchy starts first.

The distributed synthesis tool creates a new run list for each distribution phase (see [Phases of Distribution](#) on page 12). Because the criteria used during each phase is different, the number of jobs in the run list can differ from phase to phase.

3. It selects and runs jobs from the run list, beginning with dependent jobs. It writes the design database for the job and the command sequence to operate on that job, and it locates an available host. The tool assigns jobs to a host based on the maximum load of the host, specified at the beginning of a run, and the weight of the job and host.

A weight is a relative size that you assign to a module or host. For modules, it indicates the relative size of the module, such as the amount of memory that the module needs during synthesis. The weight of a host indicates the relative power of the machine. The distributed synthesis tool uses the weight values to determine which hosts to use when launching a job. A host must have the same or higher weight as the jobs that it runs. A weight of 0 for a host indicates that it can run any job; a weight of 0 for a job or module indicates that it can run on any host.

When the job completes, the master job checks the run list for other jobs that are ready to run, and it repeats the process.

Distributed Processing of Ambit BuildGates Synthesis Concepts

You can set the maximum job limit by using the `dist_max_jobs` global variable. This variable lets you limit the number of jobs that run in parallel and the number of licenses that are used at any given time.

Phases of Distribution

Distribution of the synthesis process occurs whenever possible and practical within the default logic optimization flow. There are, at most, five phases of distribution: structuring, mapping, and up to three phases of timing optimization. The actual number of distribution phases is dynamic and is influenced by the following:

- The state of the input database (mapped or not mapped)
- Whether you want to stop synthesis after mapping (see the `dist_stop_after_mapping` global variable)
- Whether you have specified `-no_design_rule` on the `do_optimize -distributed` command line
- Whether timing has already been met before the current distribution phase

Each distribution phase must be completed in its entirety before another distribution phase begins. During distribution, the `status` attribute of the job changes, as follows:

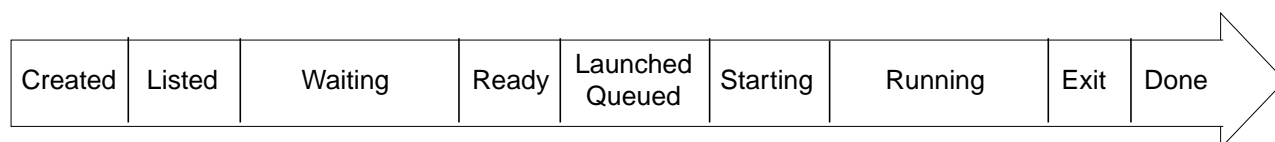
<code>created</code>	The job has been created.
<code>listed</code>	The job has been added to the run list.
<code>waiting</code>	The job is waiting for its dependent jobs to finish.
<code>ready</code>	The job is ready to run, as soon as a suitable host is available.
<code>launched or queued</code>	The job has been sent to a host machine. The <code>launched</code> status value applies to jobs started in hostlist mode; the <code>queued</code> status value applies to jobs started in LSF Batch mode.
<code>starting</code>	The job has started on the remote host. This value does not apply to jobs that you run with LSF Batch.
<code>running</code>	The job is executing on the host.
<code>exit</code>	The job has finished running on the host.

Distributed Processing of Ambit BuildGates Synthesis Concepts

done The job data has been merged back into the main synthesis database.

Figure 1-1 on page 13 shows a timeline of a job and the value of the `status` attribute for that job during a normal distributed run.

Figure 1-1 Job Status Values During a Normal Distributed Run



If the job does not complete normally, the `status` attribute is set to one of the following exception conditions:

`interrupted` You interrupted the distributed run by pressing `Control-C`.

`killed` A timeout occurred, or you invoked the `kill_job` command.

`failure` An error occurred, but it may be recoverable. The distributed synthesis tool retries the operation and changes the `status` value to reflect the outcome.

`error` An unrecoverable error occurred during the distributed run.

`deleted` You issued a `remove_job` command. After the job has been deleted, you can no longer access information about the job.

At any time after a distributed run, you can call the `get_job_info` command to return the current value of the `status` attribute for a particular job, or for all jobs. For example:

```
get_job_info -last_top status
```

This command returns the `status` value of the last top job in the distributed run. The last top job merges the information returned by all of the jobs in that phase.

Hosts

You can run distributed synthesis by using built-in load balancing, in which you specify a list of hosts, or by using LSF batch (from Platform Computing, Inc.). You need to set some host information, identify the best resources to use on the network, and specify the resources that you want to use for a distributed run.

Distributed Processing of Ambit BuildGates Synthesis Concepts

You can configure your hosts manually, or allow the tool to automatically gather and return an accurate configuration for each host. You also need to set a number of global variables that control the distributed run.

During automatic configuration, if a host fails while running a job, the tool disables the host by setting an embargo on it. Only non-embargoed hosts are used. You can remove the embargo from a host by calling the `set_host_config` command.

Important

Cadence recommends that you do not run more than one job per CPU. The job manager prevents this from happening if the host is configured correctly.

After the job manager launches a job on a host, it waits for a period of time before checking that host again to launch another job. The delay period is called the embargo time. A host can be embargoed permanently or temporarily to prevent overuse of the host machine. The embargo time is a configured time. When a new job starts, it is not immediately reflected in the load for that host. A wait time is specified after each launch so that the load information can be updated with the new information.

Jobs

A job is made up of a distributed command and one or more design modules on which the command operates for one remote `ac_shell` process. The distributed synthesis tool partitions the synthesis task into jobs that can be executed in parallel while retaining the logical hierarchy of the design. Automatic job partitioning is based on hierarchy and size of the modules in a design. The optimal size of a partition depends upon the synthesis step to be performed. You can manually specify how your design is partitioned by using the `set_dist_point` command.

Job Types

During the partitioning process, the distributed synthesis tool assigns a job number to each module in the logical hierarchy of the design. Next, it groups jobs together according to the logical hierarchy and the job size limits specified by the `dist_granularity` global. This grouping is a hierarchical design tree, with the root of the design tree called a remote job. Child modules included in the grouping are called lumped jobs, because these jobs do not go out for remote processing on their own, but are lumped with another job.

Following are the types of job that the tool creates as it partitions your design:

master The `ac_shell` that controls the distributed run.

Distributed Processing of Ambit BuildGates Synthesis Concepts

remote	A job running on another host in a separate <code>ac_shell</code> . The tools starts a new <code>ac_shell</code> for the job. A batch job is a type of remote job.
lumped	A place holder for a small job that gets lumped with the parent job to correctly reflect the dependency of the design in the hierarchy. Lumped jobs have separate job attributes, but the tool does not need to invoke a separate <code>ac_shell</code> .
top	A place holder for the results of an entire distributed run. It is the last job to run as it waits for the other jobs to complete. A top job is always a lumped job.

Job Files

During distributed synthesis, each remote distributed job operates on or creates the following files:

Design database input	Created by the master <code>ac_shell</code> and read by the remote <code>ac_shell</code> . Design database input files have the <code>.adbi</code> extension.
Input command file	Created by the master <code>ac_shell</code> and read by the remote <code>ac_shell</code> . Input command files have the <code>.tcl</code> extension.
Design database output	Created by the remote <code>ac_shell</code> and read by the master <code>ac_shell</code> upon successful completion of the remote job. Design database output files have the <code>.adbo</code> extension.
Log file	Created by the remote <code>ac_shell</code> run. This file contains the concatenation of the remote shell execution and the <code>ac_shell</code> output. Log files have the <code>.std</code> extension.

These files are stored in the directory pointed to by the `AMBIT_DIST_DIR` environment variable. During normal execution, the tool deletes all job files except the `.std` output if the distribution phase completes successfully. (See [“Setting Up a Host List”](#) on page 19.)

Job Manager

A job consists of the job itself and a list of dependent jobs. Each job has a size, and the size represents the number of instances in the design or module. The job manager tracks job

Distributed Processing of Ambit BuildGates Synthesis Concepts

dependencies to prevent high-level jobs from starting before the lower-level, dependent jobs complete.

The tool recognizes and reports the following exception conditions:

All hosts are busy All hosts are too busy to accept new jobs. The master `ac_shell` periodically checks the load of all hosts on the host list. This is done with an increasing interval delay, until a maximum time limit is met. This time limit is called the launch delay. It is the maximum delay time allowed before the master `ac_shell` checks on launching more jobs. You can specify a new launch delay by setting the `dist_launch_delay` global variable. The default is 5:00 (5 minutes.)

Failure to launch remote jobs

The master `ac_shell` could not find a lightly loaded host (a host loaded below the maximum host value) on which to launch a remote `ac_shell`. The master performs a final check on the host to make sure that it is not loaded and that it is accessible. If the launch has to be cancelled, the master makes one more attempt to relaunch the job. It repeats the process to find a lightly loaded host and attempts to launch the job again.

Job crashes

The remote job completed with an error. The job is rescheduled for another run. By default, up to three retries are allowed. You can specify the number of retries by setting the `dist_retries` global variable. The default is 2.

Job run time too long

The job has run over the run time allowed for the job. The job is killed and restarted. You can specify the timeout period for a job by setting the `dist_timeout` global variable.

Recoverable Errors

The distributed synthesis tool has tried to perform an operation, such as launching a job, and it has encountered an error. It retries the operation a specified number of times, in case the error is recoverable. The `dist_max_failures` global variable specifies the limit for the number of times the tool can retry an operation during a job run. The default value for this global variable is 3.

Stalemate

The master has timed out because all hosts on the host list are busy, or because the master could not launch a remote job for an extended period of time. When a stalemate occurs, the current jobs cannot be processed, and no new jobs can be launched.

Distributed Processing of Ambit BuildGates Synthesis Concepts

The default timeout period is 30 minutes. You can specify a timeout period by setting the `dist_launch_timeout` global variable.

User interrupts

You have interrupted the master `ac_shell`. When an interrupt occurs, all remote `ac_shells` are interrupted, and any data collected during the run is stored in the master synthesis database.

The command stops, and control returns to the command line. At this point, all the jobs have been interrupted; however the design in memory is in a partial, incomplete state. You can rerun the session, or you can delete the design. You cannot resume from the moment the master `ac_shell` was interrupted.

Killing jobs

Jobs have been killed, in one of the following ways:

The master `ac_shell` kills a job after an error is detected or a timeout period expires, and then restarts the job.

You can kill all jobs or specific jobs by calling the `kill_job` command. You can specify the jobs by name, number, or by specifying a list of jobs.

You can force a job to be killed. For example, you can set a start-up timeout for a job. The master launches an `ac_shell` and expects to hear the first heartbeat in that time period. If the remote job does not send the heartbeat during that timeout period, the remote job is killed.

Using Distributed Synthesis

This chapter discusses the following topics:

- [Distributed Synthesis Options](#) on page 18
- [Setting Up a Host List](#) on page 19
- [Setting Up LSF Batch](#) on page 21
- [Setting Global Variables](#) on page 22
- [Starting the Distributed Synthesis Run](#) on page 26
- [Monitoring the Distributed Synthesis Run](#) on page 27
- [64-bit Support for Distributed Synthesis](#) on page 37
- [Renamed and Obsoleted Commands](#) on page 44

Distributed Synthesis Options

You can choose to run distributed synthesis from a host list using built-in load balancing or with LSF batch (from Platform Computing Corporation). The method you choose determines how you configure your distribution network. When you use a host list, you specify and configure the hosts. When you use LSF batch, you can use one or several batch queues that have been set up for you by LSF Batch.

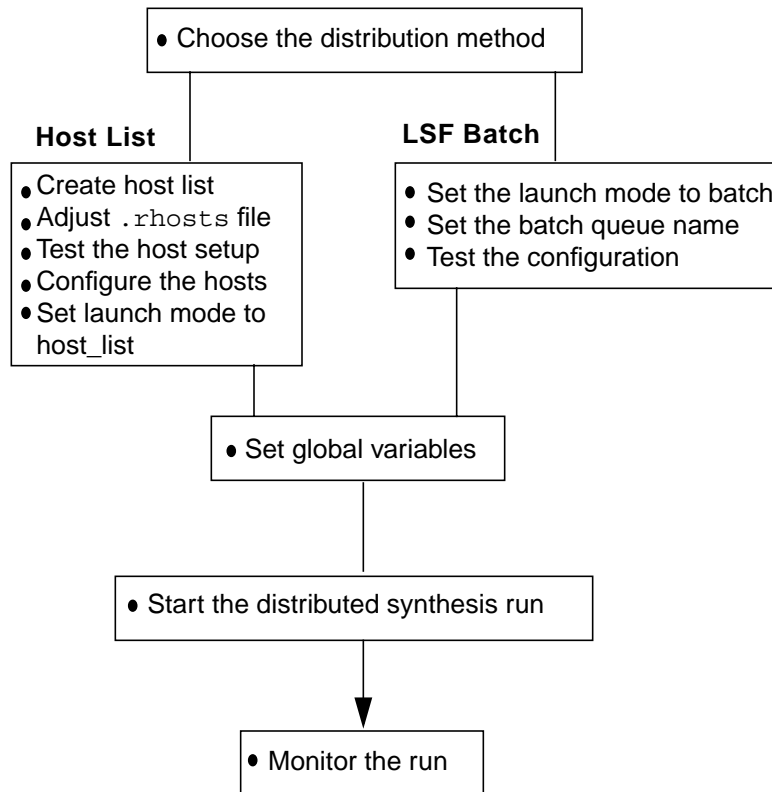
Regardless of the method you use, you set global variables to control the distributed run. For example, you can specify the maximum number of parallel jobs or the number of times the distributed synthesis tool can retry a job. After you start the distributed run, you can monitor its progress. The GUI provides many features for monitoring jobs.

[Figure 2-1](#) on page 19 shows a flowchart for setting up your distribution network, starting a distributed synthesis run, and monitoring the progress of the run.

Distributed Processing of Ambit BuildGates Synthesis

Using Distributed Synthesis

Figure 2-1 Distributed Synthesis Flowchart



Setting Up a Host List

Before using distributed synthesis, you must specify, test, and configure your remote hosts. The distributed synthesis tool keeps track of and distributes all design-specific information such as libraries, global variable definitions, and design constraints.

You can run distributed synthesis from either the command line or the GUI.

Important

You must have access to one `Ambit_BuildGates` or `Envisia_PKS` license per CPU doing distributed synthesis, plus one for the master `ac_shell`. For example, if you use one 4-CPU machine and one 2-CPU machine during distributed synthesis, you need a total of seven licenses.

1. From the UNIX command prompt and before starting the master `ac_shell`, specify the working directory for job files (optional). For example:

```
setenv AMBIT_DIST_DIR $HOME/job_files
```

Distributed Processing of Ambit BuildGates Synthesis

Using Distributed Synthesis

The default area for job-specific files is the directory from which you start the synthesis tool. To have job-specific files placed automatically in another directory, set the `AMBIT_DIST_DIR` environment variable before you invoke `ac_shell`.

2. Adjust the `.rhosts` file:

```
+ user_login
```

You may need to adjust the `.rhosts` file in your home directory so that the master machine can open shells on remote hosts without needing a password. Adding the preceding line to the `.rhosts` file on each machine in your host list should allow you to `rlogin` from your machine to any other machine.

Note: Be sure to include a space between the plus sign and `user_login`.

3. Start `ac_shell` and you may optionally remove all previously defined hosts from the host list:

```
ac_shell
remove_host -all
```

4. Define the new list of hosts:

```
set_host_list host_name_1 ... host_name_n
```

5. Test the host setup:

```
check_host -all -all -verbose -force
```

You include two `-all` options. The first specifies all hosts; the second specifies all tests.

The `check_host` command calls a Tcl script, `rac_shell`. The script runs six tests on the specified hosts, checking that the following are true:

- The remote host is valid and running.
- It is possible to `rlogin` to the remote host, without entering a password.
- The `csch` is executable on the host.
- The `csch` on the remote host returns its exit status to the local host.
- It is possible to invoke a remote `ac_shell` using the `remsh` or `rsh` command.
- The remote `ac_shell` can get a license.

The script returns the outcome of the tests as it checks each host. For more information on troubleshooting hosts using the `rac_shell` script, see [Appendix A, "Testing the Distributed Synthesis Setup."](#)

6. Configure the hosts:

```
set_host_config -all -auto -force
```

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

The `set_host_config` command configures the host machine by setting machine attributes such as the number of CPUs, the amount of memory, and the CPU speed.

Important

You must make sure that number of CPUs is configured correctly.

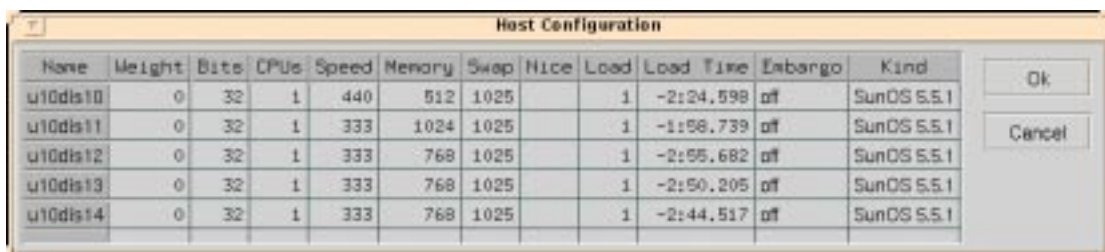
You can configure hosts manually or use the `-auto` option to configure automatically. Use the `-all` option to configure all hosts in the host list, or configure an individual host by specifying the host name.

You can view and adjust host attributes in the *Host Configuration* window of the GUI, as follows:

- ▶ In the *Distributed* panel, click the *Configure Hosts* icon (the top rightmost icon on the toolbar).

The Host Configuration Window displays the names of the hosts in the current host list, as shown in [Figure 2-2](#) on page 21.

Figure 2-2 Host Configuration Window



Name	Weight	Bits	CPUs	Speed	Memory	Swap	Nice	Load	Load Time	Embargo	Kind
u10dis10	0	32	1	440	512	1025		1	-2:24,598	off	SunOS 5.5.1
u10dis11	0	32	1	333	1024	1025		1	-1:58,739	off	SunOS 5.5.1
u10dis12	0	32	1	333	768	1025		1	-2:55,682	off	SunOS 5.5.1
u10dis13	0	32	1	333	768	1025		1	-2:50,205	off	SunOS 5.5.1
u10dis14	0	32	1	333	768	1025		1	-2:44,517	off	SunOS 5.5.1

If the embargo status of a host is *on*, the host cannot be used because it did not respond or there are problems with remotely logging into the host machine.

Setting Up LSF Batch

If you use LSF Batch, you do not have to specify and configure hosts. Instead, you perform the following steps:

1. Set the global variable `dist_launch_mode` to `batch`:

```
set_global dist_launch_mode batch
```

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

2. Specify the queue name that you want to use for batch processing of your synthesis run:

```
set_global dist_batch_queue queue_name
```

3. Test your configuration:

```
check_batch -all
```

The `check_batch` command calls a Tcl script, `rac_shell.tcl`. The script runs all tests using LSF Batch, checking that the following are true:

- LSF Batch jobs can be submitted.
- It is possible to invoke a remote `ac_shell` using the `remsh` or `rsh` command.
- The remote `ac_shell` can acquire a license.

The script returns the outcome of the tests as it checks each host.

For more information about testing your configuration, see [Appendix A, “Testing the Distributed Synthesis Setup.”](#)

Setting Global Variables

The distributed synthesis tool lets you set global variables to control the distributed run. You can set these global variables from the command line by calling the `set_global` command. The GUI lets you set these global variables from the *General Preferences* dialog box.

Setting Global Variables from the Command Line

A number of global variables lets you control the optimization flow. In addition, a number of global variables lets you control the distributed synthesis run. You can set these global variables as follows:

1. Specify the maximum number of jobs that may be run in parallel:

```
set_global dist_max_jobs 5
```

You must set `dist_max_jobs` to a value of 2 or more. The default value is 2.

2. Set any other distributed global variables that you want for the distributed synthesis run. For example:

```
set_global dist_retries 3
set_global dist_uniquify late
set_global dist_enable_final_top_down true
```

The `dist_retries` global variable specifies the maximum number of retry attempts made to run a job after the initial job run has failed. The default is 2.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

The `dist_uniquify` global variable specifies the point at which uniquification occurs during optimization. When the value is set to `late`, uniquification happens after mapping. The default is `early`.

When you set the `dist_enable_final_top_down` global variable to `true`, a final top-down pass of timing optimization is performed by calling the `do_xform_timing_correction` command.

3. To run in distributed mode by default (without using the `-distributed` option), set the global variable `dist_default` to `on`.

```
set_global dist_default on
```

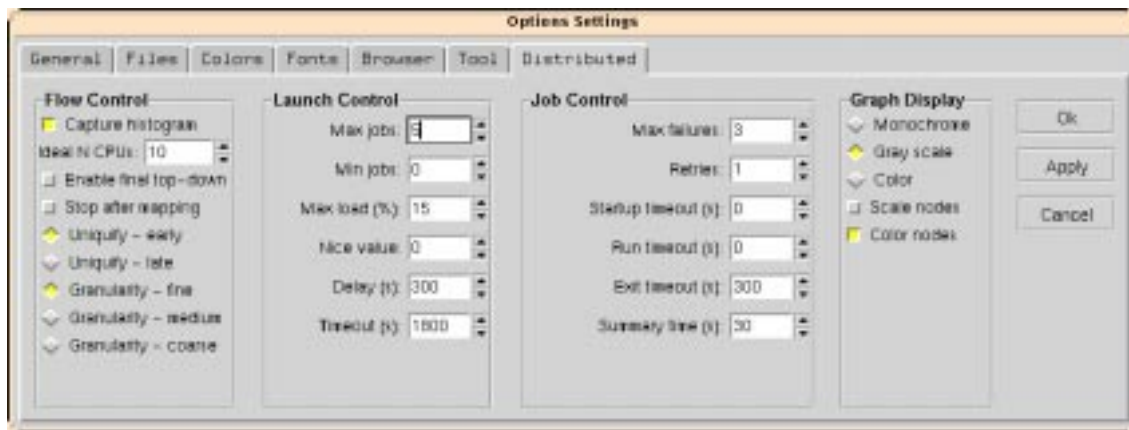
For more information about the global variables for distributed synthesis, see the `set_global` command in the Command Reference for Ambit BuildGates Synthesis and Cadence PKS .

Setting Global Variables with the GUI

You can control design optimization, how jobs launch and run, and the appearance of graphs in the *Distributed* panel of the *General Preferences* dialog box.

1. From the *View* menu, choose *General Preferences*. This opens the *Options Settings* dialog box.
2. Click the *Distributed* tab. This opens the *Distributed* panel of the dialog box shown in [Figure 2-3](#) on page 23.

Figure 2-3 Distributed Panel of the Options Settings Dialog Box



The *Distributed* panel is divided into sections, as follows:

Distributed Processing of Ambit BuildGates Synthesis

Using Distributed Synthesis

Note: The global variable set by the GUI are indicated at the end of each option description.

■ Flow Control

In the *Flow Control* section of the *Distributed* panel, you can set the following options:

- Select *Capture histogram* and set the number of ideal CPUs to view a histogram of ideal run times for a synthesis run. The global variable set by the GUI is dist_capture_job_histogram.
- Select *Enable final top-down* for a final, top-down timing optimization (useful in improving slack for designs that have difficulty in meeting timing constraints). This top-down optimization is performed using the do_xform_timing_correction command. The global variable set by the GUI is dist_enable_final_top_down. The global variable set by the GUI is
- Select *Stop after mapping* to stop optimization after mapping has completed, but before running any timing optimization. The global variable set by the GUI is dist_stop_after_mapping.
- Select *Uniquify - early* to perform uniquification during structuring; select *Uniquify - late* to perform uniquification during mapping. The global variable set by the GUI is dist_uniquify.
- Select *Granularity - fine* to create the smallest jobs for remote distribution. This setting creates the greatest number of remote jobs; it is best for small jobs. Select *Granularity - coarse* for very large designs. The default setting is *Granularity - medium*. The global variable set by the GUI is dist_granularity.

■ Launch Control

In the *Launch Control* section of the *Distributed* panel, you can set the following options:

- Increase or decrease the maximum number of jobs that can be run at one time by a remote `ac_shell` by setting the *Max jobs* text box. The global variable set by the GUI is dist_max_jobs.
- Increase or decrease the minimum number of jobs that can be run in distributed mode by setting the *Min jobs* text box. If the number of jobs in a design hierarchy is less than this value, the optimization command is not run in distributed mode. If this number is less than 2, the tool issues a warning and prevents the command from running in distributed mode. The global variable set by the GUI is dist_min_jobs.

Distributed Processing of Ambit BuildGates Synthesis

Using Distributed Synthesis

- ❑ Increase or decrease the CPU load by setting the *Max load* text box. Jobs are not launched on hosts with a CPU utilization that exceeds this percentage. The global variable set by the GUI is dist_max_load.
- ❑ Increase or decrease the UNIX `nice` value used when running remote jobs by setting the *Nice value* text box. The global variable set by the GUI is dist_nice.
- ❑ Increase or decrease the maximum delay time, in seconds, between checking for ready jobs by setting the *Delay* text box. The global variable set by the GUI is dist_launch_delay.
- ❑ Increase or decrease the launch idle timeout period, in seconds, by setting the *Timeout* text box. If the master `ac_shell` is unable to launch any remote jobs within this time period (since the last job was launched) the distributed command stops. The global variable set by the GUI is dist_launch_timeout.

■ Job Control

In the *Job Control* section of the *Distributed* panel, you can set the following options:

- ❑ Increase or decrease the limit for the number of recoverable errors allowed prior to or during a job run by setting the *Max failures* text box. The global variable set by the GUI is dist_max_failures.
- ❑ Increase or decrease the maximum number of retry attempts made to run a job after the initial job run has failed by setting the *Retries* text box. The global variable set by the GUI is dist_retries.
- ❑ Increase or decrease the startup timeout period, in seconds, for a remote job by setting the *Startup timeout* text box. A remote job should start to run (sending the first heartbeat to the master `ac_shell`) within the specified timeout period. If a job does not start within this period, it is considered to have failed. The startup timeout period starts when the job is launched. Setting the value to `0.00` turns off this option. The global variable set by the GUI is dist_startup.
- ❑ Increase or decrease the run time, in seconds, of a remote job by setting the *Run timeout* text box. If the run time of a job exceeds the time period specified, the master `ac_shell` kills the job and reruns it on another host. The run time of a job starts when the master `ac_shell` receives the first heartbeat. Setting the value to `0.00` turns off this option. The global variable set by the GUI is dist_timeout.
- ❑ Increase or decrease the shutoff timeout period, in seconds, for a remote job by setting the *Exit timeout* text box. If a remote job does not finish in the timeout period after the final heartbeat has been received by the master `ac_shell`, the remote job is killed. Setting the value to `0.00` turns off this option, but this is not recommended. The global variable set by the GUI is dist_shutoff.

Distributed Processing of Ambit BuildGates Synthesis

Using Distributed Synthesis

- ❑ Increase or decrease the delay time, in seconds, between printing summary reports on currently running jobs by the master `ac_shell` by setting the *Summary time* text box. The global variable set by the GUI is `dist_summary_delay`.

■ Graph Display

In the *Graph Display* section of the *Distributed* panel, you can set the following options:

- ❑ Select one of the following display modes:
 - Monochrome*—Displays the dependency graph in black and white.
 - Gray scale*—Displays the dependency graph in shades of gray.
 - Color*—Displays the dependency graph in color, where the master job is red, remote jobs are purple, and lumped jobs are beige.
- ❑ When *Scale nodes* is enabled, the size of the node is relative to the size of the job. That is, large jobs are represented by large nodes; small jobs are represented by small nodes. When *Scale nodes* is disabled, all nodes are the same size.
- ❑ When *Color nodes* is enabled, the nodes are colored as determined by the display mode that you have selected: *Monochrome*, *Gray scale*, or *Color*.

Starting the Distributed Synthesis Run

Once you complete the setup and configuration described in the previous sections, you run distributed synthesis. To start a synthesis run from the `ac_shell` command line:

1. Read the library and the design files into the synthesis database, and set the constraints. For more information, see the *Ambit BuildGates Synthesis User Guide*.
2. Start distributed synthesis optimization with one of the following commands:
 - ❑ `do_optimize -distributed`

This command runs all three phases of synthesis in distributed mode—structuring, mapping, and timing optimization.
 - ❑ `do_xform_map -distributed`

This command issues a warning message and does not run in distributed mode if any `extract_critical` options such as `-timing` or `-critical_ratio` are selected.
 - ❑ `do_xform_optimize_generic -distributed`

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

This command distributes only the structuring step and none of the other steps, such as mapping.

❑ `do_xform_optimize_slack -distributed`

This command distributes the timing correction or remapping phase and the final characterize phase, but not the initial characterize and buffer tree insertion phase.

Important

Only commands that support the `-distributed` command option can run in the distributed mode. All standard `do_optimize` and `transform` command line options, except the `-time_budget` and `-pks` options, support the `-distributed` option.

Monitoring the Distributed Synthesis Run

The distributed synthesis tool provides several mechanisms for monitoring the progress of a distributed run:

- When you run distributed synthesis from a script, you can use attributes of a job to return information about the time taken to run the different job phases.
- When you run from the command line, the distributed synthesis tool writes information to the synthesis log file for each job that it runs.
- When you run from the GUI, you can open several windows that are updated continuously with information about the distributed run.

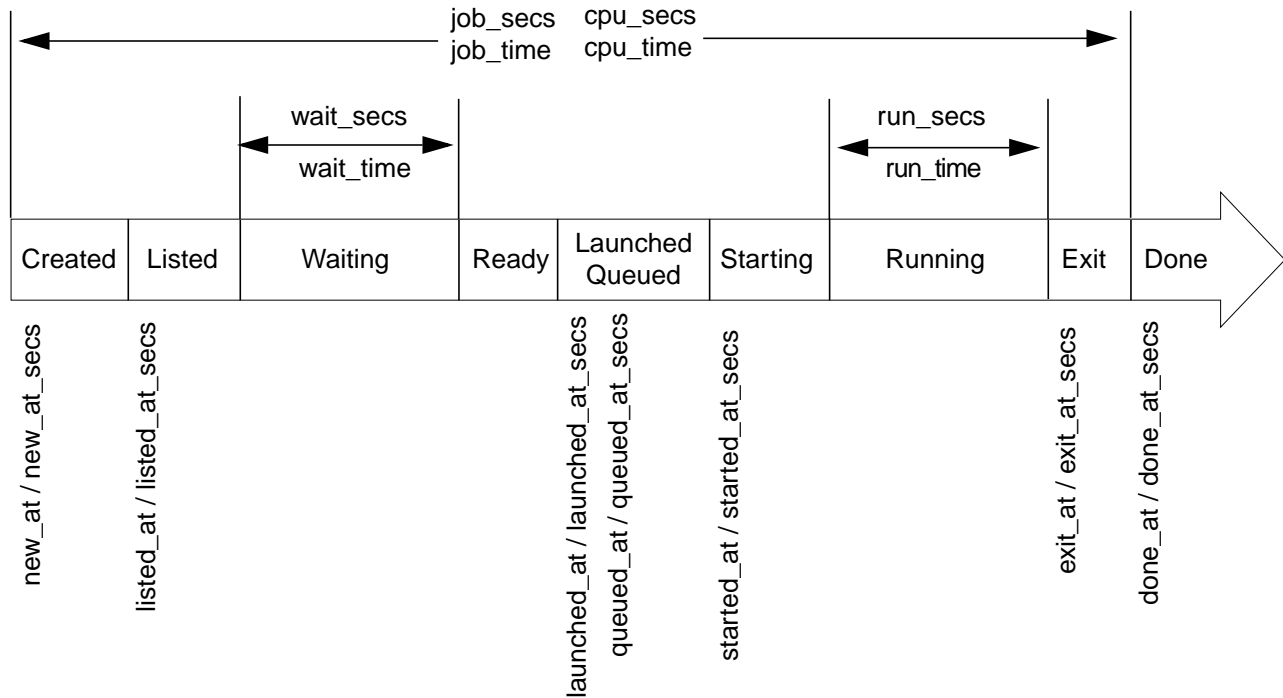
Tracking the Run Time of Remote Jobs

As the job passes through each of the distribution phases, the distributed synthesis tool keeps track of the time that the phase begins or the elapsed time of the phase. It stores this information as attributes of the job. You can access the job attributes, usually from scripts, to report the progress of the distributed synthesis run. You call the `get_job_info` command to return the values of these attributes.

Figure 2-4 on page 28 shows the timeline of a distributed job and the attributes that keep track of the run time of each phase.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Figure 2-4 Runtime Attributes of a Job



Each time attribute is available in two formats—number of seconds or *day-hh:mm:sss* format. Times that are returned in *day-hh:mm:sss* format are strings, which you can use to display wall clock times during the distributed run. For example, you can get the time at which the job with an ID of 1 was placed on the run list, as follows:

```
get_job_info 1 listed_at
```

You can get the number of seconds from the beginning of the distributed run until the job was placed on the run list, as follows:

```
get_job_info 1 listed_at_secs
```

You can also perform arithmetic operations on the times that are stored in number of seconds format. For example, to determine the amount of time that elapsed between launching the job and starting the job, you can subtract the value of the `started_at_secs` attribute from the `launched_at_secs` attribute. For example:

```
launch_time = [get_job_info 1 launched_at_secs] | [get_job_info 1 started_at_secs]
```

Job attributes are updated whenever a remote job completes. These variables contain the cumulative remote totals when the master `ac_shell` exits.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Monitoring the Distributed Synthesis Run with the Log File

The `ac_shell.log` file records all of the messages that Distributed Synthesis returns during a distributed synthesis run. At the beginning of the distributed run, the tool writes information to the log file about the top job. For example:

```
Info: Job 1 Creating -distributed jobs ... (at 44.729) AutoConfig 'all' (top)
<JMGR-101>.
Info: Job 1 Running -distributed jobs ... (at 44.945) AutoConfig 'all' (top)
<JMGR-101>.
Info: Job Mgr Total: no jobs to run (no lumped jobs) of width 0 (at 44.946)
<JMGR-101>.
```

As the tool launches each job, it displays the job number, the host name, and the time that the job was launched. The tool also displays a summary message, which includes information about the total number of jobs in the distributed run, the number of jobs remaining to run, the number of jobs that are done, and error information. For example:

```
Info: Job 2 launched on 'u10dis10' pid 10785 (at 50.765) AutoConfig 'u10dis10'
<JMGR-101>.
Info: Job Mgr Summary: 1 job running (1 most, 19 max), 19 to do, 0 done,
20 total, no errors (at 50.767) <JMGR-101>.
Info: Job 3 launched on 'u10dis12' pid 10797 (at 56.641) AutoConfig 'u10dis12'
<JMGR-101>.`
Info: Job Mgr Summary: 2 jobs running (2 most, 19 max), 18 to do, 0 done,
20 total, no errors (at 56.643) <JMGR-101>.
Info: Job 4 launched on 'u60pv80' pid 10823 (at 1:02.191) AutoConfig 'u60pv80'
<JMGR-101>.
Info: Job Mgr Summary: 3 jobs running (3 most, 19 max), 17 to do, 0 done,
20 total, no errors (at 1:02.193) <JMGR-101>.
Info: Job 2 running on 'u10dis10' rpid 12029 pid 10785 (at 1:02.196)
AutoConfig 'u10dis10' <JMGR-101>.
Info: Job 5 launched on 'u60pv81' pid 10845 (at 1:07.643) AutoConfig 'u60pv81'
<JMGR-101>.
```

As each job completes, the tool displays the run time of the job. For example:

```
Info: Job 2 done, run_time 5.452 (at 1:08.092) AutoConfig 'u10dis10'
<JMGR-101>.
```

Because the log file is a running history of every message returned during a distributed synthesis run, you may find it difficult to monitor the progress of the run by viewing this file. The GUI displays the same information, but in a form that is easier to use.

Monitoring the Distributed Synthesis Run with the GUI

The GUI gives you access to information about the job status, remote jobs, log files for remote jobs, job dependencies, actual run times, and ideal run times. This information is available from the *Distributed* panel and continuously updated throughout the distributed synthesis run.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Viewing Job Status

While optimization is taking place, the GUI updates the job status in the *Distributed* panel. The *Distributed* panel provides feedback during distributed optimization, as shown in Figure 2-5 on page 30.

Figure 2-5 Job Status

Job	Run	Status	Host	Weight	Bits	Size	Parent	Path	Kind	Name
1-T	1	done	localhost	0	64	0	none	1	AutoConfig	all
2-R	1	done	u10dis10	0	32	0	1	1-2	AutoConfig	u10dis10
3-R	1	done	u10dis11	0	32	0	1	1-3	AutoConfig	u10dis11
4-R	1	done	u10dis12	0	32	0	1	1-4	AutoConfig	u10dis12
5-R	1	done	u10dis13	0	32	0	1	1-5	AutoConfig	u10dis13
6-R	1	done	u10dis14	0	32	0	1	1-6	AutoConfig	u10dis14
7-T	1	done	localhost	0	64	0	none	7	Structuring	cpu
8-R	1	done	u10dis10	0	32	530	7	7-8	Structuring	cpu
9-L	1	done	u10dis10	0	32	49	8	7-8+9	Structuring	reg8_1
10-L	1	done	u10dis10	0	32	49	8	7-8+10	Structuring	reg8_0
11-L	1	done	u10dis10	0	32	73	8	7-8+11	Structuring	count5
12-L	1	done	u10dis10	0	32	148	8	7-8+12	Structuring	decode
13-L	1	done	u10dis10	0	32	12	12	7-8+12+13	Structuring	AWPLR_0
14-L	1	done	u10dis10	0	32	181	8	7-8+14	Structuring	alu
15-L	1	done	u10dis10	0	32	56	14	7-8+14+15	Structuring	AWACL_UN5_ADD_8_C
16-T	1	done	localhost	0	64	0	none	16	Mapping	cpu
17-R	1	done	u10dis11	0	32	633	16	16-17	Mapping	cpu
18-L	1	done	u10dis11	0	32	66	17	16-17+18	Mapping	reg8_0
19-L	1	done	u10dis11	0	32	66	17	16-17+19	Mapping	reg8_1
20-L	1	done	u10dis11	0	32	73	17	16-17+20	Mapping	count5
21-L	1	done	u10dis11	0	32	81	17	16-17+21	Mapping	decode
22-L	1	done	u10dis11	0	32	306	17	16-17+22	Mapping	alu
23-L	1	done	u10dis11	0	32	56	22	16-17+22+23	Mapping	AWACL_UN5_ADD_8_C
24-T	1	done	localhost	0	64	0	none	24	OptimizeTiming	cpu
25-H	1	done	localhost	0	0	213	24	24-25	OptimizeTiming	cpu
26-L	1	done	localhost	0	0	18	25	24-25+26	OptimizeTiming	count5
27-L	1	done	localhost	0	0	115	25	24-25+27	OptimizeTiming	alu
28-L	1	done	localhost	0	0	45	27	24-25+27+28	OptimizeTiming	AWACL_UN5_ADD_8_C
29-L	1	done	localhost	0	0	16	25	24-25+29	OptimizeTiming	reg8_0
30-L	1	done	localhost	0	0	28	25	24-25+30	OptimizeTiming	decode
31-L	1	done	localhost	0	0	16	25	24-25+31	OptimizeTiming	reg8_1

The *Distributed* panel displays the following information:

Job

The job number and the job type. The job type can be one of the following values:

T The top job is the synchronization or roll-up point for all remote jobs.

M The master job runs on the local host and is responsible for top-level optimization during bottom-up distribution phases.

Distributed Processing of Ambit BuildGates Synthesis

Using Distributed Synthesis

- R* The remote job runs on a remote host.
- L* The lumped job is included with a remote job listed in the *Path* column. Remove lumped jobs from the display by clicking the *Hide* button at the top of the panel.

Run The number of times a remote job has been run. Numbers greater than 1 indicate that the early run had a problem and that the job has been retried on a different remote host.

Status The status of the jobs. Colors associated with the various job status values may be modified through the *View – Preferences* menu. The *Status* values are as follows:

- created* The job files have been created by the master.
- waiting* The job is waiting to run.
- launched* The job has been sent to a remote machine (host list mode only).
- queued* The job has been submitted as an LSF Batch job (batch mode only).
- starting* The job has started executing on the remote machine (host list mode only).
- running* The job is executing on a remote machine.
- interrupted* The job was terminated by the user.
- exit* The job has finished on a remote machine, but the results of the job have not yet been absorbed back into the master `ac_shell` database.
- done* The job has finished, and the job output has been successfully merged back into the master `ac_shell` database.
- failure* The remote job encountered a failure. The job is automatically restarted on another machine, as long as the number of retries is equal to or below the value of the `dist_retries` global variable (default = 2).

Distributed Processing of Ambit BuildGates Synthesis

Using Distributed Synthesis

<i>killed</i>	The remote job was killed by the master due to a failure condition in the system or due to a <code>kill_job</code> command.
<i>Host</i>	The name of the host on which the remote job is executing.
<i>Weight</i>	Indicates the weight of the host. The weight of the host must be the same as the weight of the job.
<i>Bits</i>	Indicates the bits of the host, 32- or 64-bits.
<i>Size</i>	The size of a remote job as measured by the number of instances that may be modified.
<i>Parent</i>	The parent job.
<i>Path</i>	The job hierarchy path from the top of the hierarchy down to and including the current job.
<i>Kind</i>	The type of the current distribution phase.
<i>Name</i>	The name of the module associated with the remote job. This is the module at the root of the design tree that is sent for remote processing.

Viewing Remote Job Information

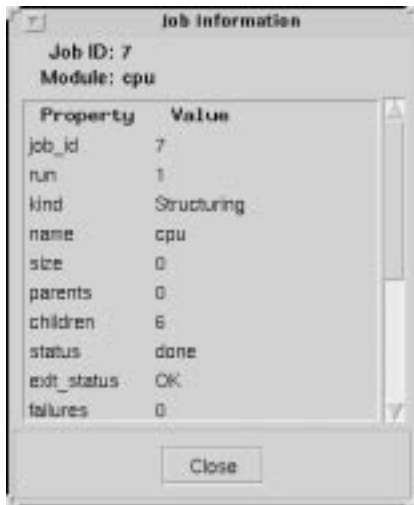
To view information on any remote job:

1. Select the job in the *Distributed* panel, click the right mouse button, and choose *Info*. The GUI displays the job information form, as shown in [Figure 2-6](#) on page 33.

The Job Information form displays the job ID, module name for the top module in the job, and the value of each job property.
2. To close the form, click *Close*.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Figure 2-6 Job Information



Viewing the Log File for a Remote Job

To display the log file for a job:

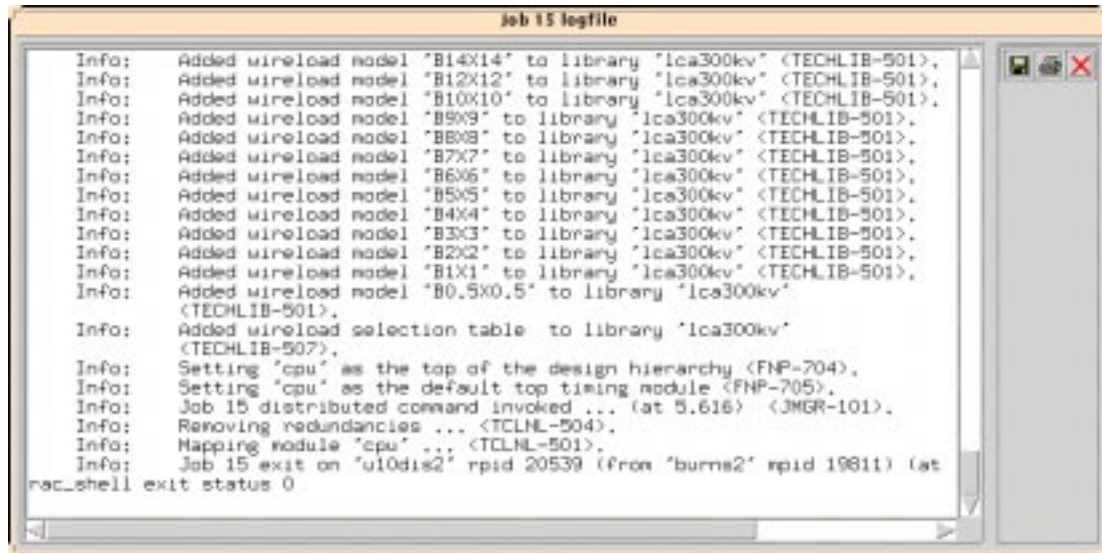
1. Select the job, right-click the mouse button, and choose *Logfile* from the pop-up menu. The GUI displays the log file for that job, as shown in [Figure 2-7](#) on page 34. This menu choice appears only if a log file exists for the selected job.

When you scroll to the end of the file, you can see that the remote shell operates on multiple modules at one time.

2. To print the graph, click on the printer (middle) icon located at the upper right-hand corner of the graph..
3. To save the graph, click on the save (left) icon located at the upper right-hand corner of the graph.
4. To close the graph, click the red X (right) icon located at the upper right-hand corner of the graph.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Figure 2-7 Log File for a Remote Job (Partial)



```
Job 15 logfile
Info: Added wireload model "B14X14" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B12X12" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B10X10" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B9X9" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B8X8" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B7X7" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B6X6" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B5X5" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B4X4" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B3X3" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B2X2" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B1X1" to library "lca300kv" <TECHLIB-501>.
Info: Added wireload model "B0.5X0.5" to library "lca300kv"
<TECHLIB-501>.
Info: Added wireload selection table to library "lca300kv"
<TECHLIB-507>.
Info: Setting "cpu" as the top of the design hierarchy <FNP-704>.
Info: Setting "cpu" as the default top timing module <FNP-705>.
Info: Job 15 distributed command invoked ... (at 5.616) <JMGR-101>.
Info: Removing redundancies ... <TCLNL-504>.
Info: Mapping module "cpu" ... <TCLNL-501>.
Info: Job 15 exit on "ul0dis2" rpid 20539 (from "burns2" rpid 19811) (at
rac_shell exit status 0
```

Viewing the Job Dependency Graph

To view the the job dependency graph for top jobs:

1. Right-click the job and select *Draw New Graph*. The GUI displays the Job Dependency Graph, as shown in [Figure 2-8](#) on page 35. The job dependency graph matches the logical hierarchy of the design.

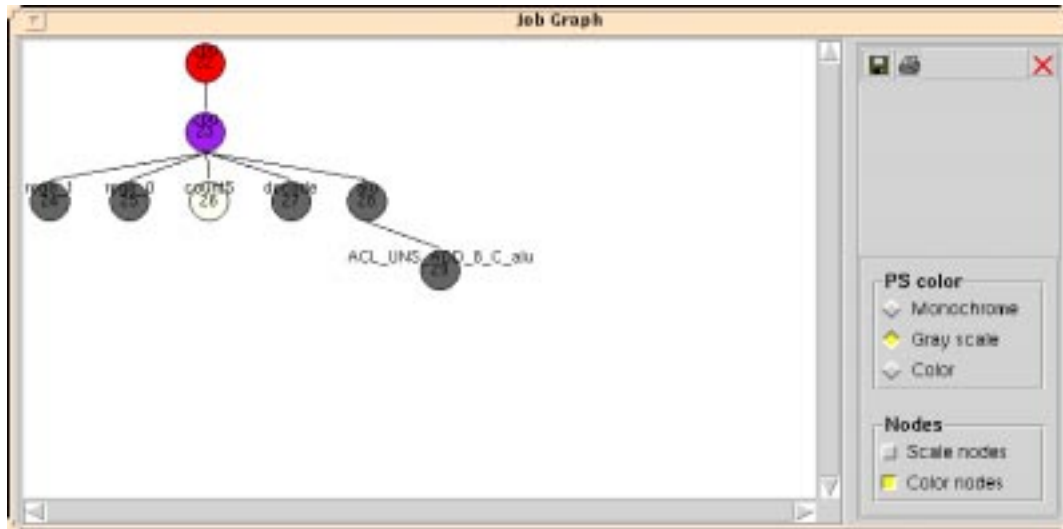
You can color the graph by selecting *Color* under PS color. You can scale the nodes by selecting *Scale nodes* and *Color nodes* under Nodes. You must close the current job graph and reopen another one to apply your selections.

When redrawing the graph, remote jobs are shown in blue, lumped jobs in beige, and the top job in red.

2. To print the graph, click on the printer (middle) icon located at the upper right-hand corner of the graph..
3. To save the graph, click on the save (left) icon located at the upper right-hand corner of the graph.
4. To close the graph, click the red X (right) icon located at the upper right-hand corner of the graph.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Figure 2-8 Job Dependency Graph



Viewing the Actual Runtime Graph

To view an Actual Runtime Graph for any top-level job:

1. Right-click the job and select *Draw Actual Runtime Graph*. The GUI displays the Actual Runtime Graph, as shown in [Figure 2-9](#) on page 36.

This graph shows the actual distribution of jobs to remote CPUs, detailing their start and finish time in seconds. Remote CPUs are shown at the top of the graph and time progresses forward from the top to the bottom of the graph. Jobs that ran first are at the top of the graph.

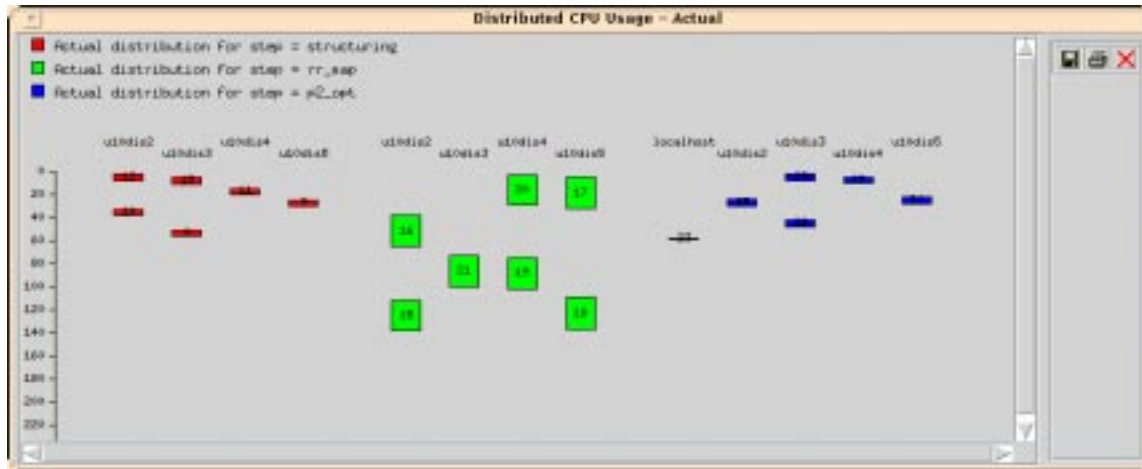
Gaps between jobs indicate the processing time by the master to absorb the results. When optimizing timing distribution, gaps indicate the required wait time for a dependent job to complete.

At any time, the number of jobs running in parallel is shown on the horizontal axis. You can pan to the left and right in the pop-up pane to show both distribution phases. The longest running job is the first to start because it is larger.

2. To print the graph, click on the printer (middle) icon located at the upper right-hand corner of the graph..
3. To save the graph, click on the save (left) icon located at the upper right-hand corner of the graph.
4. To close the graph, click the red X (right) icon located at the upper right-hand corner of the graph.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Figure 2-9 Actual Runtime Graph



Viewing the Ideal Runtime Graph

To view an Ideal Runtime Graph for any top-level job:

1. Right-click the job and select *Draw Ideal Runtime Graph*. The GUI displays the Ideal Runtime Graph, as shown in [Figure 2-10](#) on page 37.

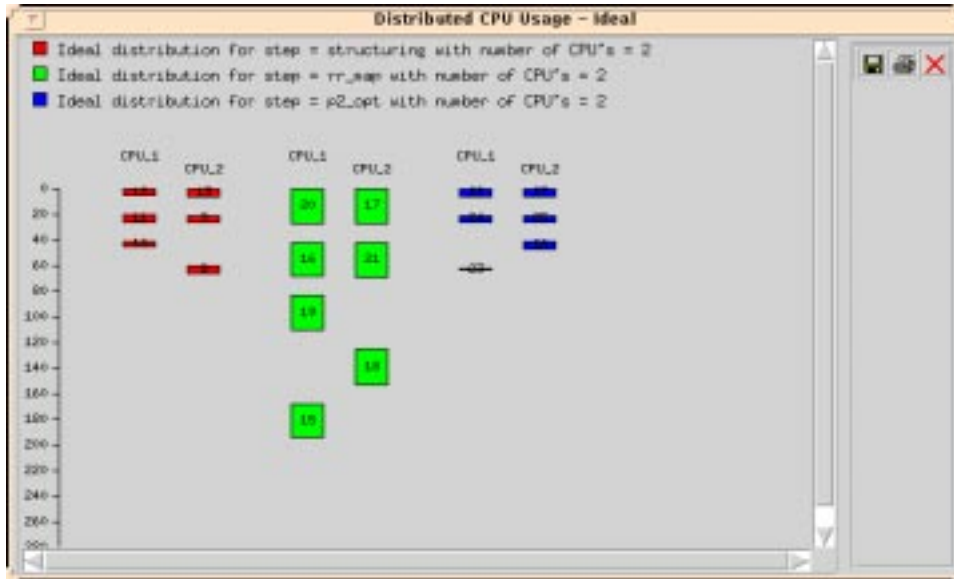
This graph allows you to determine the point at which adding CPUs no longer provides decreased run time. The results are based upon the job dependencies and the job run time characteristics.

You must provide the number of CPUs to be used. By default, an Ideal Runtime Graph is created during distributed synthesis using the value of the `dist_max_jobs` global, which you can modify in the *Distributed* pane of the *Options Setting* window.

2. To print the graph, click on the printer (middle) icon located at the upper right-hand corner of the graph..
3. To save the graph, click on the save (left) icon located at the upper right-hand corner of the graph.
4. To close the graph, click the red X (right) icon located at the upper right-hand corner of the graph.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Figure 2-10 Ideal Runtime Graph



64-bit Support for Distributed Synthesis

The general rule for distributed synthesis is that `ac_shell` processes on all remote computers run with the same command line options as the master `ac_shell`. That applies to licensed options like `-Datapath` and `-Power` as well as the `-large` option and now also the new `-64` option. The latter controls whether a 64- or 32-bit `ac_shell` is invoked.

To find out whether the current `ac_shell` runs 64 bits, use the command

```
check_option 64
```

It will return 1 for a 64- and 0 for a 32-bit `ac_shell`. Other `ac_shell` command line options can be checked with the `check_option` command in the same fashion. Note, a 64-bit `ac_shell` can only run on a 64-bit O/S. But a 32-bit `ac_shell` may run on a 32- or 64-bit O/S.

To find out whether the current O/S supports 64 bits (regardless of the `ac_shell`), check global `tcl_platform(osBits)`. This global will only be available after the `ac_shell` has been initialized completely. While this is not an issue for most command line options, it is probably too restrictive to force `ac_shell` on all remote computers to be 64 bits if the master `ac_shell` is 64 bits. It is reasonable to assume that when a remote host does support 64 bits, the remote `ac_shell` on that host should be 64 bits, provided the master `ac_shell` is 64 bits.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Important

Do not launch a 64-bit `ac_shell` on a remote host which does not support 64 bits. This will lead to (frequent) launch failures that will cause the entire 64-bit distributed synthesis run to end in an (unnecessary) error.

In many practical design situations it is probably unnecessary to run remote computers with 64-bit `ac_shell` even if the master `ac_shell` is 64 bits. Often, the size of partial designs processed remotely can easily be handled by a 32-bit `ac_shell`. But there is currently no reliable metric to determine that in advance.

If a partial design is processed remotely and if it is too large for 32 bits, the 32-bit `ac_shell` will run out of memory, eventually. The entire 64-bit distributed synthesis run will ultimately fail if that partial design does not end up on a 64-bit host with a 64-bit `ac_shell` somewhere.

In summary, the rule that all remote computers with `ac_shell` will be 64 bits like the master `ac_shell`, is too restrictive. In order to provide more flexibility, the following new commands and options have been introduced for 64-bit distributed synthesis:

- new global `dist_bits`
- new job attribute option called `-bits`
- new host configuration option item called `-bits`
- enhanced host configuration option item called `-kind`
- new job attribute option called `-rbits`
- improved host selection criteria
- new comands `set_dist_bits` and `reset_dist_bits`

More detailed descriptions and some examples follow.

Note: The following subsections apply to a 64-bit master `ac_shell` only. If the master `ac_shell` is 32 bits, the `ac_shell` on all remote computers will always be 32 bits.

Job Attribute Option `-bits`

In order to provide control over which `ac_shell` to run for 64-bit distributed synthesis, a new job attribute has been added, called `-bits`. The `-bits` attribute specifies the minimum number of bits required to run this job. If the `-bits` value of a job is 32, that job can be handled by a 32- or 64-bit `ac_shell`. But jobs with a `-bits` value of 64 can only be processed by a 64-bit `ac_shell`.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Jobs can be assigned a specific `-bits` value using the `set_dist_bits` command. Jobs with a `-bits` value of 0 (zero) will inherit the value of the new global `dist_bits`. By default the `-bits` value of all jobs is set to 0 and by default `dist_bits` is set to 64, but in a 64-bit master `ac_shell` only.

As a result, by default all jobs launched from a 64-bit master `ac_shell` will run with 64-bit `ac_shell` on all remote computers. In other words, by default `ac_shell` on all remote computers are 64 bits if the master is 64 bits. But that behavior can be changed by setting global `dist_bits` and specific job `-bits`, as shown in [“Example of Using Global Variable `dist_bits` and `set_dist_bits`”](#) on page 44.

The `-bits` value of a job is very similar to the `weight` of a job. Just like the `weight` attribute of a job controls on which hosts a job can run, the `-bits` attribute determines which hosts and `ac_shell` run this job.

Global Variable `dist_bits`

To set a default value for all jobs without a specific bit size, use

```
set_global dist_bits p2
```

where `p2` is 0, 32, or 64. Any job without an explicit `bits` value will receive the global `dist_bits` value. The 64-bit `ac_shell` will initially set global `dist_bits` to 64 while the 32-bit `ac_shell` will leave `dist_bits` at 0, meaning ignore.

Note that global `dist_bits` must not be set to a value above the bit size of the current master `ac_shell`. Specifically, if the current `ac_shell` is 32-bit, `dist_bits` can not exceed 32.

Host Configuration Item `-bits`

A new configuration item for hosts has been added, also called `-bits`. The host `-bits` indicates that that host runs a 32- or 64-bit O/S. Also, the host `-bits` value should only be 64 if there is a 64-bit `ac_shell` available for that host. The `-bits` item of a host can be set manually using the existing `set_host_config` command

```
set_host_config host_name -bits p2
```

where `p2` is any power of 2. If `p2` is 64, it means this host runs a 64-bit O/S and that a 64-bit `ac_shell` exists and runs that host. Hosts with `-bits` equal 32 (or 0) can only run a 32-bit `ac_shell`. A 64-bit `ac_shell` will be launched only on hosts with `-bits` equal to 64.

If the hosts in the host list are configured automatically, using the command

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

```
set_host_config -auto -all
```

the `-bits` value of all hosts will be set automatically. Only hosts which run a 64-bit O/S and where a 64-bit `ac_shell` is available and runs, will show `-bits` as 64, otherwise `-bits` will be 32. The `set_host_config -auto` command option will set `-bits` accordingly. Verify the host `-bits` value using command

```
get_host_info host_name
```

or

```
get_host_info host_name -bits
```

and compare it against the host `-kind` string. If the host `-kind` shows a 64-bit O/S it is still possible that the `-bits` value for that host is 32. This means that although this host runs 64 bit O/S, there is no 64-bit `ac_shell` available for that host.

The host `-bits` item must be set to 64 only if the O/S on that machine does support 64-bit and if a 64-bit `ac_shell` is available and runs on that machine. Otherwise, host `-bits` should remain 32 (or zero).

Important

When the `-bits` item of a host is configured manually, using the command `set_host_config host_name -bits p2`, a 64-bit `ac_shell` will only be launched on hosts that have `-bits` set to 64. An incorrect `-bits` value for a host could result in 64-bit `ac_shell` launch failures and will cause the distributed synthesis run to fail.

Automatic host configuration with command `set_host_config -all -auto` determines an accurate `-bits` value for hosts automatically. If the remote host runs a 64-bit O/S that is reflected in the `-kind` item of the host. But only if a 64-bit `ac_shell` exists and runs on that host the `-bits` item will be set to 64, otherwise it will be 32.

In other words, it is possible that the host `kind` shows a 64-bit O/S, for example {AIX 4.3 64-bits} but the `-bits` item is only 32. That means that while the O/S would support 64 bits, there is no 64-bit `ac_shell` available for that host.

Host Item `kind`

An automatically configured 64-bit `-host` will show `kind` as {O/S name O/S version 64-bits} see [“Some Examples for Three Different Machines, After Executing `set host config auto`”](#) on page 43. For non-64-bit hosts the `-host -kind` item will just show the O/S name and the O/S version.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Note: The `-kind` value is no longer the result of the Cshell command `/bin/uname -sm`. Instead, the value of standard Tcl globals `tcl_platform(os)` and `tcl_platform(osVersion)` is now used to obtain this information.

Try the following commands:

```
get_host_info host_name
```

or

```
get_host_info host_name -kind
```

Host Selection

When a suitable host needs to be found for a particular job, four criteria are taken into consideration:

- The host `-bits` value must be at least equal to the `-bits` value of the job
- The weight of the host must be equal or above the job `-weight`
- The load of the host must be below the value of global `dist_max_load`
- The host can not be embargoed, either permanently or temporarily

The `-bits` rule only applies if the value of both the host and job `-bits` is non-zero. Similarly, the `-weight` criteria is only applied if the host and job `-weight` are both non-zero.

Once a suitable host has been found, the `ac_shell` will be launched as follows. If the master `ac_shell` runs 64 bits and if host `-bits` is 64, the 64-bit `ac_shell` will be launched. Otherwise a 32-bit `ac_shell` is launched.

These rules guarantee that a job requiring 64 bits will only be launched on hosts that support 64 bits and that can run a 64-bit `ac_shell`. In other words, there is no chance that a 64-bit `ac_shell` will be invoked on a 32-bit host.

Also, jobs that do not require 64 bits based on their `-bits` value will still be implemented with a 64-bit `ac_shell` if the host `-bits` is 64. This is done on purpose to err on the safe side.

Job Attribute `-rbits`

Jobs have another new attribute called `-rbits`. This attribute shows the `-bits` size of the remote `ac_shell` which actually ran that job. The `-rbits` value will be set after the job is completed. It will be 64 if a 64-bit `ac_shell` ran this job, otherwise it is 32.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Under normal conditions the `-rbits` value of a job should never be below the `-bits` value of a job. The `-rbits` value of a job is set by the job manager and can only be queried by the user using the command

```
get_job_info job -rbits
```

A zero value for `-rbits` means that the job did not start, did not run, or some other error occurred.

Launch Mode `host_list` vs `batch`

The rules described above apply only if the master `ac_shell` was invoked with the `-64` command line option and if the global `dist_launch_mode` is set to `host_list`.

For launch mode `-batch` (i.e., jobs are submitted to LSF) the `ac_shell` on all remote computers will always run the same as the master `ac_shell`. If the master is a 64-bit `ac_shell`, the `ac_shell` on all remote computers submitted to LSF will also be a 64-bit `ac_shell`.

Special Command Options

Using `get_host_info all -bits n` returns the total number of hosts which can run executables of n -bits. For example, `get_host_info all -bits 64` returns the total number of hosts currently in the host list which could run a 64-bit `ac_shell`.

The command `get_host_info all -bits n weight w` returns the total number of hosts which can run an `ac_shell` of n -bits and can handle jobs of weight w . For example, `get_host_info all -bits 64 weight 10` returns the total number of hosts which can run jobs of weight up to 10 with a 64-bit `ac_shell`.

Added Commands

Two additional commands have been implemented for 64-bit distributed synthesis, `set_dist_bits` and `reset_dist_bits`. A combination of the new global `dist_bits` and the command `set_dist_bits` can be used to specify the bit size of the `ac_shell` required to process a particular module. The command syntax is

```
set_dist_bits non-negative_power_of_2 | list_of_module_names_or_ids  
reset_dist_bits list_of_module_names_or_ids
```

These commands allow the `-bits` value for a distributed synthesis job to be specified on the module in the design. They are very similar to the existing `set_dist_weight` and `reset_dist_weight` commands which attach the job weight to design modules.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

When running 64-bit distributed synthesis, the `-bits` value of a job determines whether that job requires a 32- or 64-bit remote `ac_shell`. Note that `reset_dist_bits ...` is equivalent to `set_dist_bits 0`

However, the `-bits` size of a job can never exceed the `-bits` size of the master `ac_shell`. For example, if you are running distributed synthesis from a 32-bit `ac_shell`, you can not specify that any modules should be run with a 64-bit `ac_shell`. This is done on purpose to prevent a remote 64-bit `ac_shell` from generating a design which could be too large for the 32-bit master `ac_shell` to read back in.

When the job list is being created at the beginning of a distributed synthesis command, the bit size for each job is compared with the bit size of the master `ac_shell`. If the bit size of any job exceeds the master bit size, the command will not run distributed synthesis.

Some Examples for Three Different Machines, After Executing `set_host_config auto`

```
ac_shell[1]> info hostname
bigfoot.cAdence.COM
ac_shell[2]> get_host_info localhost -bits
32
ac_shell[3]> get_host_info localhost
{name localhost} {namedomain bigfoot.cAdence.COM} {-bits 32} {cpus 1}
{embargo on} {kind {SunOS 5.5.1}} {memory 1024} {nice 1} {release v4.1}
{rlogin OK} {ruser OK} {speed 300} {swap 1025}
ac_shell[4]> get_host_info e65eng -bits
64
ac_shell[5]> get_host_info e65eng
{name e65eng} {namedomain e65eng.Cadence.COM} {-bits 64} {cpus 10}
{embargo on} {kind {SunOS 5.7 64-bits}} {memory 14336} {nice 1}
{release v4.1} {rlogin OK} {ruser OK} {speed 400} {swap 22007}
ac_shell[6]> get_host_info sunblade -bits
64
ac_shell[7]> get_host_info sunblade force
{name sunblade} {namedomain sunblade.cAdence.COM} {access 0} {-bits 64}
{cpus 2} {embargo on} {kind {SunOS 5.8 64-bits}} {memory 4096} {nice 1}
{release v4.1} {rlogin OK} {ruser OK} {speed 750} {swap 16385}
```

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Example of Using Global Variable `dist_bits` and `set_dist_bits`

Assume the design in a 64-bit `ac_shell` is very large and would not fit in 32 bits. But most of the partial designs that run as distributed synthesis jobs are rather small and can be handled by a 32-bit `ac_shell`, except for one or two very large modules, that require a 64-bit `ac_shell`. Also, assume that most machines on the host list run 32 bits except for a couple 64-bit machines. Finally, assume that a 64-bit `ac_shell` is available for the 64-bit machines.

The following command sequence allows this design to be run distributed synthesis optimally, since it will utilize all 32- and 64-bit machines if needed.

- Set host list and auto config all hosts

```
set_host_list ....
set_host_config -all -auto -force
```

- Check whether there are any host and `ac_shell` that are 64-bits

```
if { [ check_option 64 ] } {
    if { [ get_host_info -all -bits 64 ] < 1 } {
        issue_message -type error "No 64-bit hosts available"
    }
}
```

- Use a 32-bit remote `ac_shell` for most modules

```
set_global dist_bits 32
```

- Except for two large modules which require 64 bits

```
set_dist_bits 64 {module1 module2}
```

- Go do it

```
do_optimize -distributed ...
....
```

Setting `dist_bits` to 32 will make sure that most remote jobs will run on a 32-bit hosts using a 32-bit `ac_shell`. However the jobs for modules "module1" and "module2" will run on a 64-bit host with a 64-bit `ac_shell`.

If the job `-bits` size could not be specified explicitly per job or module, this design would only run distributed synthesis on the few 64-bit machines available.

Renamed and Obsoleted Commands

This section is intended to be read and used by current users of the Distributed tool. Disregard this section if you are a new user.

Distributed Processing of Ambit BuildGates Synthesis Using Distributed Synthesis

Renamed Commands

Four distributed synthesis commands have been renamed for consistency as follows:

Note: Both commands will work and can be used, however, the 'is now' command is the preferred command.

`set_weight is now set_dist_weight`

`reset_weight is now reset_dist_weight`

`set_distribution_point is now set_dist_point`

`reset_distribution_point is now reset_dist_point`

Obsoleted Commands

The commands `set_weight`, `reset_weight`, `set_distribution_point` and `reset_distribution_point` will still exist for the 4.0 release, but will become obsolete in the next major release, 5.0.

Restarting Distributed Synthesis Jobs

This chapter discusses the following topics:

- [Restart Feature](#) on page 46
- [Limiting Distributed Synthesis Jobs](#) on page 53

Restart Feature

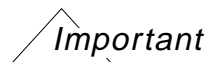
The restart feature in distributed synthesis allows remote jobs to be killed and automatically restarted.

Restarting a distributed synthesis job must be done by the user, while not running master `ac_shell`. Due to standard permission restrictions, typically only the user that started the distributed synthesis command can restart a job.

There are several new globals related to restart they are `dist_restart_signal`, `dist_restart_embargo`, `dist_restart_delay` and `dist_max_restarts`. There is also a script called the `restart` script and new command `check_restart`, each discussed in more detail below.

Global Variable `dist_restart_signal`

To restart a distributed synthesis job, the currently running job must first be killed by an agreed upon signal. That signal is referred to as the restart signal.



Make sure that the restart signal is unique so that it can be distinguished from other signals, used to interrupt or kill jobs altogether.

The restart signal is the signal used to kill a running job on a remote machine. Most important is that the user must send the signal to the remote job, either by invoking the `restart` script that the distributed synthesis software generated for this purpose, or by running a

Distributed Processing of Ambit BuildGates Synthesis

Restarting Distributed Synthesis Jobs

standard Unix shell `kill` command or, when running in batch mode, by the batch `kill` command in LSF called `bkill`.

The global `dist_restart_signal` only indicates to the master `ac_shell` (1) that distributed synthesis jobs can be restarted, and if so (2) which signal will be used to kill the running job. If `dist_restart_signal` is set to a catchable signal, remote jobs will be initialized such that the signal can be caught and the remote job will exit in an orderly fashion. The master `ac_shell` will reschedule jobs killed by the restart signal for relaunch, at some later time on some other or possibly the same host.

Restarting of remote jobs is enabled and handled correctly only if the global `dist_restart_signal` is set to a catchable signal, like `QUIT`, `HUP`, `TERM`, `USR1` or `USR2`. See “[Catchable Signals](#)” on page 50. By default, the global `dist_restart_signal` is set to `NULL` (zero), which disables restarting of distributed synthesis jobs.

Global Variable `dist_restart_embargo`

The global `dist_restart_embargo` specifies a time period to embargo a host where a running job was killed for restart. The main idea is that distributed synthesis jobs are restarted to free up a host, to take a host off-line, etc. When a job is killed by the restart signal, the host where the job was running will not be used to run additional jobs for the time period specified by `dist_restart_embargo`.

For example, if `dist_restart_embargo` is set to `1:00:00` a host will be embargoed for an hour after a remote job running on that host is restarted. No jobs will be launched on an embargoed host. If `dist_restart_embargo` is not set (i.e., zero), then killed jobs may be restarted on that same host again and freeing up a host would become impossible. However, by default, `dist_restart_embargo` is set to zero.

Global Variable `dist_restart_delay`

The new global `dist_restart_delay` is set to delay the restart of a job hit by the signal specified by the `dist_restart_signal` global variable. By default, `dist_restart_delay` is set to zero, meaning the normal delay is used for scheduling the job restart. When a job is rescheduled for restart, the master log file will indicate the delay time used to schedule the restart time. An example for the log shown below is

```
Info:   Job 2 restart #2 scheduled for 41.372 (+0.750) approx. (at 40.623) .....
```

The time shown in parenthesis is the restart delay applied for the restart of this particular job.

Distributed Processing of Ambit BuildGates Synthesis

Restarting Distributed Synthesis Jobs

Note that the globals `dist_restart_delay` and `dist_restart_embargo` are very different. The former applies to the job being restarted, while the latter affects the host where the job was running when it was killed by the `dist_restart_signal`.

Global Variable `dist_max_restarts`

This global variable limits the number of times jobs can be restarted. By default `dist_max_restarts` is set to zero, meaning the number of restarts for any jobs is unlimited. If the number of restarts of any job exceeds `dist_max_restarts` the job will be considered an error.

Important

Jobs are not only restarted after being killed by a `dist_restart_signal`, but jobs can also be restarted after a license failure. A restart after a missing license is also included in the restart count of a job.

Restart Script

For each running, restartable, remote job the master `ac_shell` will write a Cshell script in the working directory of distributed synthesis. Each script can be used to kill and restart a particular job. The name of the script includes the job id, the name of the master host, process id of the master `ac_shell` and the extension `restart`. An example is

```
.job2.bigfoot.14436.restart
```

It is important to keep in mind that the restart script for a particular job only exists as long as the job is actually running. This script will be removed as soon as the `job` finishes.

To restart a distributed synthesis job, just run the restart script. For example, from the working directory use

```
./job2.bigfoot.14436.restart
```

To kill a job permanently or to send another signal to the remote job, specify the signal name as the optional argument

```
./job2.bigfoot.14436.restart KILL
```

or any other name for a signal. Other useful signals may be `INT`, `USR1`, and `USR2`.

Important

Do not use numeric signal identifiers, since the value may differ across platforms.

Distributed Processing of Ambit BuildGates Synthesis

Restarting Distributed Synthesis Jobs

Command `check_restart`

There is a new command to test restarting of jobs, either on specific hosts or in batch mode. The command syntax is

```
check_restart [ signal ] [ batch | hosts ] [ options ]
```

where valid options are

- `-batch_queue [queue_name]`
- `-debug`
- `-force`
- `-interval [time]`
- `-max_jobs [n]`
- `-signal [signal]`
- `-timeout [time]`
- `-verbose`
- `-silent`

The `check_restart` command will run a distributed synthesis job on each specified host for a limited time and kill and restart each job. An example

```
check_restart QUIT host1 host2
```

starts two remote `ac_shells`, one on host `host1` and one on host `host2`. The master `ac_shell` then sends restart signal `QUIT` to each remote job in an attempt to kill and restart that job. If everything works as intended, the remote jobs are launched, killed and restarted several times until the `check_restart` command is interrupted or the `dist_max_restart` value is exceeded. See [“Examples of check_restart”](#) on page 51 for sample runs of `check_restart`.

- `signal` and `-signal signal` — the first argument `signal` is the name of a catchable signal to be used as a restart signal. The restart signal can also be specified as the value of the `signal` option. If neither the first argument, nor the `-signal` option are supplied, the current value of the global `dist_restart_signal` will be used.
- `hosts ...` — is a list of host names where to run jobs to be restarted.

If no hosts are specified, the current list of hosts from the `get_host_list` command will be retrieved.

Distributed Processing of Ambit BuildGates Synthesis

Restarting Distributed Synsthesis Jobs

- `batch` and `batch_queue queue_name` — to run in batch mode, specify `batch`. Optionally, supply a batch queue to be used with the `batch_queue` option. If no batch queue is specified, the value of global `dist_batch_queue` is undefined, and the default batch queue will be used.
- `debug` — prints out detailed information during the `check_restart` command. This is intended to debug problems.
- `force` — the `check_restart` command uses the generated `restart` script to restart jobs. However, when the option `force` is used, the `restart` script is not invoked. Instead, the restart signal is sent directly to the remote job. When running in batch, the LSF `bkill` command will be issued, in addition to sending the kill signal.
- `interval time` — specifies the rate at which jobs are restarted, i.e., the rate at which restart signals are sent. Any value over 1 second is acceptable, default is 2 seconds. Note that if the interval time is larger than the timeout period, no jobs will be restarted since the job will exit before that.
- `max_jobs n` — limits the number of jobs run at any given time. Normally, one job will be launched for each host, but if that is too much, the `max_jobs` options can be used to lower the number of jobs running at the same time. When running in batch, by default only one job is submitted. Use `max_jobs` to increase the number of batch jobs running for `check_restart`.
- `timeout time` — this option is used to extend the time remote jobs are active to increase the chance to hit remote jobs with the restart signal. The default and minimal value for `timeout` is 16 seconds. On busy hosts or networks, a longer value will probably be needed. Especially when running in batch, a much longer timeout value is necessary, possibly as long as several minutes.
- `verbose` — the `-verbose` option supplies a more detailed output. Specifically, the log output will include a line every time the restart signal is sent to a remote job or every time the restart script is invoked. Setting option `debug` implies `verbose`. The example under [“Examples of check_restart”](#) on page 51 shows a `check_restart` run with and one without the `verbose` option.
- `silent` — is the opposite of `-verbose` and limits the log output to a minimum. However, option `-debug` overrides `-silent`.

Catchable Signals

A catchable signal is defined as any signal that (1) is not handled by the database manager (called FNP in BG/PKS) and (2) can be caught. At this time, the following signals are non-catchable: NULL, INT, BUS, SEGV, ILL, XCPU, XFSZ, CHLD, SYS, STOP, KILL and ABRT.

Distributed Processing of Ambit BuildGates Synthesis Restarting Distributed Synthesis Jobs

Signals are not only used to restart jobs, but they are also used to limit and kill jobs in the globals `dist_kill_signal` and `dist_rlimit`, respectively, and possibly in the `set_dist_rlimit` command. Any distinct catchable signal can be used for `dist_restart_signal` and `dist_kill_signal`. But at this time, only `USR1` and `USR2` can be used for global `dist_rlimit` and in the `set_dist_rlimit` and `limit` commands.

There is potential conflict with signals used for restarting, limiting and killing distributed synthesis jobs. In addition to the new globals `dist_restart_signal` and `dist_rlimit`, there still exists `dist_kill_signal`. It is very important that these signals are all set to different, unique values, for example.

```
dist_restart_signal  QUIT | HUP
dist_kill_signal     TERM (default)
dist_rlimit          USR1 | USR2
```

Specifically, globals `dist_restart_signal` and `dist_kill_signal` can not be set to the same signal. To verify the settings of all distributed synthesis globals, the existing command `check_dist` can be used, and it also checks for any signal conflicts. For more information, see [Limiting Distributed Synthesis Jobs](#) on page 53.

Important

Distributed synthesis commands will NOT run if `dist_restart_signal` and `dist_kill_signal` are set to the same signal. In addition, a warning message will be printed if the global `dist_rlimit` or the `rlimit` value of any module or job is set to a signal matching `dist_restart_signal` or `dist_kill_signal`.

Examples of `check_restart`

The following is a sample run of the most basic `check_restart` command, checking signal `QUIT` on two hosts. Two jobs, `job2` and `job 3` are launched, one on each host and start running. Soon thereafter `job 2` gets the restart signal and is killed, as indicated by the first `WARNING` line. The next line shows that `job 2` is scheduled to be restarted. Meanwhile, `job 3` is running and `job 2` is restarted. Shortly thereafter, `job 3` receives the restart signal and is rescheduled, see the third `WARNING` line. This process is repeated until either the `dist_max_restarts` value is met or until the `check_restart` command is interrupted.

```
ac_shell[1] check_restart  QUIT  e65eng  sunblade

Info: Job Mgr checking restart signal [QUIT] with e65eng sunblade (at 11.598)
      JMGR-101.

Info: Job 1 Creating -distributed jobs ... (at 11.830) AutoConfig all (top)
      JMGR-101.

Info: Job 1 Running -distributed jobs ... (at 11.831) AutoConfig all (top)
      JMGR-101.
```

Distributed Processing of Ambit BuildGates Synthesis Restarting Distributed Synsthesis Jobs

Job 1 waiting, bits 32, 2 children (at 11.831) AutoConfig all (top) JMGR-102.
Job 2 waiting, path 1-2, parent 1 (at 11.831) AutoConfig e65eng JMGR-102.
Job 3 waiting, path 1-3, parent 1 (at 11.831) AutoConfig sunblade JMGR-102.
Info: Job Mgr Total: 3 jobs to run (no lumped jobs) of width 2 and height 2 (at 11.832) JMGR-101.
Info: Job Mgr increased launch idle timeout to 5:26.000 (at 11.832) JMGR-101.
Info: Job 2 launched on e65eng pid 25719 (at 16.975) AutoConfig e65eng JMGR-101.
Info: Job Mgr Summary: 1 job running (1 most, 2 max), 2 to do, 0 done, 3 total, no errors (at 16.976) JMGR-101.
Info: Job 3 launched on sunblade pid 25763 (at 22.550) AutoConfig sunblade JMGR-101.
Info: Job Mgr Summary: 2 jobs running (2 most, 2 max), 1 to do, 0 done, 3 total, no errors (at 22.552) JMGR-101.
Info: Job 2 running on e65eng rpid 25740 pid 25719 (at 22.555) AutoConfig e65eng JMGR-101.
Info: Job 2 new restart file; usage: /.job2.e65eng.25622.restart (at 22.613) AutoConfig e65eng JMGR-101.
Info: Job Mgr Summary: 2 jobs running (2 most, 2 max), 1 to do, 0 done, 3 total, no errors (at 24.061) JMGR-101.
WARNING: Job 2 killed by restart signal QUIT on e65eng rpid 25740 (at 25.631) AutoConfig e65eng JMGR-106.
Info: Job 2 restart #1 scheduled for 28.147 (+2.500) approx. (at 25.647) AutoConfig e65eng JMGR-101.
Info: Job 3 running on sunblade rpid 19071 pid 25763 (at 26.483) AutoConfig sunblade JMGR-101.
Info: Job 3 new restart file; usage: /.job3.e65eng.25622.restart (at 26.602) AutoConfig sunblade JMGR-101.
WARNING: Job 3 sends WARNING: hanging for 14.899 seconds ... (at 26.869) AutoConfig sunblade JMGR-106.
Info: Job Mgr Summary: 1 job running (2 most, 2 max), 2 to do, 0 done, 3 total, no errors (at 28.446) JMGR-101.
Info: Job 2 launched on e65eng pid 25901 (at 32.766) AutoConfig e65eng JMGR-101.
Info: Job Mgr Summary: 2 jobs running (2 most, 2 max), 1 to do, 0 done, 3 total, no errors (at 32.767) JMGR-101.
WARNING: Job 3 killed by restart signal QUIT on sunblade rpid 19071 (at 32.768) AutoConfig sunblade JMGR-106.
Info: Job 3 restart #1 scheduled for 33.783 (+1.000) approx. (at 32.783) AutoConfig sunblade JMGR-101.

Distributed Processing of Ambit BuildGates Synthesis Restarting Distributed Synsthesis Jobs

```
Info: Job Mgr Summary: 1 job running (2 most, 2 max), 2 to do, 0 done, 3 total, no
errors (at 34.126) JMGR-101.
Info: Job 3 launched on sunblade pid 25970 (at 38.389) AutoConfig sunblade JMGR-101.
Info: Job Mgr Summary: 2 jobs running (2 most, 2 max), 1 to do, 0 done, 3 total,
no errors (at 38.391) JMGR-101.
Info: Job 2 running on e65eng rpid 25946 pid 25901 (at 38.393) AutoConfig e65eng
JMGR-101.
Info: Job 2 new restart file; usage: /.job2.e65eng.25622.restart (at 38.474)
AutoConfig e65eng JMGR-101.
WARNING: Job 2 sends WARNING: hanging for 13.806 seconds ... (at 38.982) AutoConfig
e65eng JMGR-106.
WARNING: Job 2 killed by restart signal QUIT on e65eng rpid 25946 (at 40.606)
AutoConfig e65eng JMGR-106.
Info: Job 2 restart #2 scheduled for 41.372 (+0.750) approx. (at 40.623) AutoConfig
e65eng JMGR-101.
Info: Job 3 running on sunblade rpid 19169 pid 25970 (at 41.783) AutoConfig sunblade
JMGR-101.
Info: Job 3 new restart file; usage: /.job3.e65eng.25622.restart (at 41.896)
AutoConfig sunblade JMGR-101.
----- etc.
```

In this second example the `check_restart` options `-force` and `-verbose` are both used. With `-verbose` a line is printed for every restart signal sent to a running job. Option `-force` uses the kill command to deliver the restart signal instead of the generated restart script.

```
check_restart QUIT e65eng sunblade -force -verbose
```

Limiting Distributed Synthesis Jobs

The distributed synthesis remote job limiting relies on the `limit` command. The `limit` command allows some CPU time or other limit to be specified for any BG/PKS command. The new `limit` command provides four different limits

```
limit {-cycle n | cpu_time | -sigUsr1 | -sigUSR2 }....
```

The signal limits `-sigUSR1` and `-sigUSR2` are particularly useful when running distributed synthesis. Just like the existing limits, signal limits will be handled synchronously such that the command being limited finishes in an orderly fashion.

To limit distributed synthesis commands, a new global called `dist_rlimit` and new commands `set_dist_rlimit` and `reset_dist_rlimit` are available. There is also a command to test distributed synthesis limits, `check_rlimit`.

Distributed Processing of Ambit BuildGates Synthesis

Restarting Distributed Synthesis Jobs

Global Variable `dist_rlimit`

By default `dist_rlimit` is set to an empty string, resulting in the distributed synthesis commands running without any limit. If `dist_rlimit` is set to any valid limit of the `limit` command, then the distributed synthesis commands in remote jobs will be pre-fixed by the limit specified.

Examples of `dist_rlimit`

```
set_global dist_rlimit 2:00
do_optimize -distributed ....
```

will limit each remote optimization command to at most 2 minutes CPU time. After 2 minutes, the remote command will finish as if it had completed normally.

or

```
set_global dist_rlimit -USR2
do_optimize -distributed ....
```

will run each remote optimization command until the remote job receives a USR2 signal. Then the optimization command will finish normally.

Important

If `dist_rlimit` is set to a `-cycle <n>` limit, the limit must be specified as a single argument as follows

```
set_global dist_rlimit { -cycle n } ....
```

Global Variables `set_dist_rlimit` and `reset_dist_rlimit`

Global commands `dist_rlimit`, `set_dist_rlimit`, and `reset_dist_rlimit` allow setting specific limits for distributed synthesis jobs. The command syntax is

```
set_dist_rlimit {limit} {list_of_module_names_or_ids}
reset_dist_rlimit {list_of_module_names_or_ids}
```

When running distributed synthesis, the value `set_dist_rlimit` determines the specific limit for that particular job. Note that `reset_dist_rlimit ...` is equivalent to `set_dist_rlimit {}`

Distributed Processing of Ambit BuildGates Synthesis

Restarting Distributed Synthesis Jobs

As with the global `dist_rlimit`, the `-cycle n` limit must be specified as a single argument in the `set_dist_rlimit` command, e.g.,

```
set_dist_rlimit { -cycle n } ....
```

Here is an example of using the global `dist_rlimit` and the command `set_dist_rlimit` to limit jobs to a specific amount of CPU time.

```
# limit all jobs to 2 hours CPU time
set_global dist_rlimit 2:00:00
# except two large modules which require 10 hours
set_dist_rlimit 10:00:00 [module1 | module2]
# go do it ....
do_optimize -distributed ...
....
```

Command `check_rlimit`

This command tests the limiting of distributed synthesis jobs, either on specific hosts or in batch mode. The command syntax is

```
check_rlimit [ limit ] [ batch | hosts ... ] [ options ]
```

where valid options are

- `-batch_queue queue_name`
- `-debug`
- `-interval time`
- `-max_jobs n`
- `-rlimit limit`
- `-timeout time`
- `-verbose`
- `-silent`

The `check_rlimit` command will run a distributed synthesis job on each specified host for a specific amount of time and with the specified limit. This example

```
check_rlimit -USR2 {host1 | host2}
```

starts `ac_shell` on two remote computers, one on host `host1` and one on host `host2`. The master `ac_shell` then sends the limit signal `USR2` to each remote job in an attempt to force the job to finish. If every thing works as intended, the remote jobs exit normally.

Distributed Processing of Ambit BuildGates Synthesis

Restarting Distributed Synsthesis Jobs

`limit` and `-rlimit limit` — the first argument `limit`

is the `limit` to be used in the test. The `limit` can also be specified as the value of the `-rlimit` option. If neither the first argument, nor the `-rlimit` option are supplied, the current value of global `dist_rlimit` will be used.

`hosts ...`— is a list of host names where to run jobs to be limited. If no hosts are specified, the current list of hosts from the `get_host_list` command will be retrieved.

`-batch` and `-batch_queue queue_name` — to run in batch mode, specify

`batch`. If no batch queue is specified, the value of global `dist_batch_queue` is undefined, the default batch queue will be used.

`-debug` — prints out detailed information during the `check_rlimit` command. Mainly intended to debug problems.

`-interval time` — specifies the rate at which jobs should receive limit signals. Any value over 1 second is acceptable, default is 2 seconds.

Note: If the interval time is larger than the timeout, jobs will not be limited since the job will exit before that.

`-max_jobs n` — limits the number of jobs run at the same time. Normally, one job will be launched for each host, but if that is too much, the `-max_jobs` option can be used to lower the number of jobs running at any given time. When running in batch, by default only one job is submitted. Use `-max_jobs` to increase the number of batch jobs running for `check_rlimit`.

`-timeout time` — this option is used to extend the time remote jobs hang around, mainly to increase the chance to hit remote jobs with the limit signal. The default and minimal value for `-timeout` is 16 seconds. On busy hosts or networks a longer value is probably be needed. Especially when running in batch, a much longer timeout value is likely required, possibly as high as several minutes.

`-verbose` — the `-verbose` option will cause more detailed output. Specifically, the log output will include a line every time the limit signal is sent to a remote job. Setting option `-debug` implies `-verbose`.

`-silent` — is the opposite of `-verbose` and limits the log output to a minimum. However, option `-debug` overrides `-silent`.

Testing the Distributed Synthesis Setup

This chapter describes how to troubleshoot the hosts used during distributed synthesis.

- [Testing Host Setup with `rac_shell`](#) on page 57
- [Testing LSF Batch Setup with `rac_shell`](#) on page 60

Testing Host Setup with `rac_shell`

The `rac_shell` script (located in the same directory as the `ac_shell` executable) launches a copy of `ac_shell` on a remote host and returns the exit status of the remote shell. During a distributed synthesis run, the master `ac_shell` uses the script to invoke each remote `ac_shell`. The script contains six tests to check that a host meets the prerequisites for running distributed synthesis.

Before running the synthesis tool in distributed mode, you should first test your distributed synthesis resources by using the `check_batch` or `check_host` command, both of which call the `rac_shell` script. An example of this is shown in the section “[Setting Up a Host List](#)” on page 19. These commands perform the same operations that the `rac_shell` script performs. Use the script and the information in this appendix to test and debug any hosts that return errors.

 **Important**

This script is not intended for standalone or manual use except for the testing purposes as specified in this appendix.

Syntax

```
rac_shell { host | batch | -help } [-batch_queue queue] -test integer
```

Distributed Processing of Ambit BuildGates Synthesis

Testing the Distributed Synthesis Setup

Arguments

<code>-batch</code>	Specifies that you want to test your LSF Batch setup.
<code>-batch_queue <i>queue</i></code>	Specifies the name of the LSF queue to test.
<code>-help</code>	Displays help on the <code>rac_shell</code> script.
<code>host</code>	Specifies the name of the host to test. If you set the <code>host</code> to <code>localhost</code> , the script tests the local machine.
<code>-test <i>integer</i></code>	Specifies an integer from 1 to 6, inclusive, that represents the test that you want to run. See “ The rac_shell Script Tests ” on page 58 for information about the tests that you can run.

Return Values

Usually, the `rac_shell` script returns the exit status of the remote `ac_shell`. However, the `rac_shell` script may occasionally detect an error condition and display a brief message in addition to the exit status.

The `rac_shell` Script Tests

The `rac_shell` script contains the following tests to check that a host meets the prerequisites for running distributed synthesis:

Test	Description
1	Checks the local <code>~/rhosts</code> file to determine whether remote login from the local host is permitted, and verifies that the <code>AMBIT_RACSH_HOST</code> environment variable is defined on the remote host.
2	Invokes a <code>cs</code> h remotely and checks whether it has any problems.
3	Invokes an <code>ac_shell</code> on the specified remote host and forces the <code>ac_shell</code> to exit immediately, before checking out a license and before doing any initialization.
4	Creates a small Tcl test script and invokes an <code>ac_shell</code> on the specified host to run the script. The remote <code>ac_shell</code> is invoked with the option <code>-no_init</code> to skip initialization and is expected to exit immediately with a specific exit status. The remote <code>ac_shell</code> must check out a license. It exits with a different status if the check fails.

Distributed Processing of Ambit BuildGates Synthesis

Testing the Distributed Synthesis Setup

Test Description

- | | |
|---|---|
| 5 | Is the same as test 4, except that the remote <code>ac_shell</code> is invoked without the <code>-no_init</code> option. Therefore, any initialization from <code>~/ .ambit/ .acshrc</code> is performed prior to sourcing the Tcl test script. |
| 6 | Generates a larger Tcl test script to verify access to the current or a specified directory. Verification may fail when the directory is mounted. Otherwise, this test is the same as test 5. |
-

Debugging Test Failures

The `rac_shell` script displays a message to indicate whether the test passed or failed. The test may fail because it could not verify the result of the remote `ac_shell` script. If one or more of the `rac_shell` script tests fail, use the following techniques to debug your hosts:

- Use the `cs`h command `ping host` to verify the host name. If the `ping` command is not available, try `/usr/sbin/ping`.
- Try the `cs`h command `rlogin host` and check whether the password prompt appears.

If a password is required, create or modify the file named `.rhosts` in your home directory.

Add a line to the file containing `+user` or `+localhost user` or `+localhost` (where `user` is your login name and `localhost` is the name of local host). For more information about the `.rhosts` file, see your UNIX documentation or your system administrator.

- If `cs`h does not reside in the usual `/bin` directory, set the environment variable `AMBIT_CSH_PATH` to the correct location.
- Try the following commands in a `cs`h on the local host, where `host` is the name of the remote host:

```
set cmd = (remsh host -n "exit 77 >&/dev/null; echo" '$status')
echo ` $cmd `
```

The output from the `echo` command must be `77`. If the output is different, a problem may reside in the `~/ .cshrc` file or the `~/ .login` file in your home directory. The `remsh` command must return a remote exit status.

Output from `cs`h could interfere with returning the remote exit status correctly. For example, an `echo` command in the `~/ .cshrc` file or the built-in `cs`h variable `time` may create additional output and prevent the remote exit status from being passed back correctly. Either remove the offending `cs`h commands entirely from the `~/ .cshrc` file or

Distributed Processing of Ambit BuildGates Synthesis

Testing the Distributed Synthesis Setup

make sure that such `cs`h commands are not executed when running remotely. For example, instead of using:

```
set time = 10
```

use:

```
if ( $?AMBIT_RACSH_HOST ) then
    unset time
else
    set time = 10
endif
```

In this case the environment variable `AMBIT_RACSH_HOST` is set only if this is a remote `ac_shell` execution. The value of `AMBIT_RACSH_HOST` is the name of the master host from which the `rac_shell` script invoked an `ac_shell` on this remote host.

- A number of standard `cs`h commands must be executable. The commands should exist in the same directory as the `cs`h executable. There are a few exceptions, such as the `ping` command in `/usr/sbin/ping`. Tests 3 and 4 check this.
- You must be able to invoke a remote `ac_shell` using the `remsh` (or `rsh`) command. The `ac_shell` on a remote machine must be able to access the same initialization files as a local `ac_shell` and must start without errors. The exit status of the remote `ac_shell` must also be passed back correctly. Tests 3, 4, and 5 check for correct remote `ac_shell` execution with and without the usual `ac_shell` initialization.
- A remote `ac_shell` must be able to acquire a license, either an `Ambit_BuildGates` license or an `Envisia_PKS` license. Tests 4 through 6 verify this. By default, the environment variable `AMBIT_RACSH_LICENSE` is set to `no_PKS_1st`. Set it to `no_PKS` or `PKS`, as needed.
- The remote `ac_shell` must be able to access files and directories using the same absolute path name as a local `ac_shell`. Test 6 checks this and makes some adjustment for mounted files or directories, if needed. Mounted directories or files often do not have the same absolute path name on remote and local hosts. Conversely, directories or files with the same absolute path name may not be the same physical directory or file. For example, a file named `/tmp/file` is typically a different file on a remote host and a local host. It is not possible to run a remote `ac_shell` unless the same (possibly adjusted) absolute path name accesses the same directory or file on the remote and local host.

Testing LSF Batch Setup with `rac_shell`

When running the distributed synthesis tool through LSF, the default LSF method sets up the remote environment. LSF copies the entire environment from the local host and recreates the

Distributed Processing of Ambit BuildGates Synthesis

Testing the Distributed Synthesis Setup

environment on the remote host. This ensures that environment variables `LM_LICENSE_FILE` and `AMBIT_RACSH_HOST` are copied to the remote host. If this is not adequate for your specific case, you can use the `-L` option of the LSF `bsub` command to run a specific startup shell on the remote host. To do so, set the `AMBIT_BSUB_OPTIONS` environment variable prior to invoking `rac_shell.tcl`. For example:

```
setenv AMBIT_BSUB_OPTIONS "-L /bin/csh"
```

You can specify other `bsub` command options by setting the `AMBIT_BSUB_OPTIONS` environment variable. For example, to send email, add the `-B` option:

```
setenv AMBIT_BSUB_OPTIONS "-L /bin/csh -B"
```

You can modify the environment variables in [Table A-1](#) on page 61 to refine the testing process.

Table A-1 Environment Variables

Variable	Description
<code>AMBIT_BSUB_CMD</code>	Specifies the LSF command to submit to the batch queue. Default: <code>/lsf/local/bin/bsub</code>
<code>AMBIT_BSUB_OPTIONS</code>	Specifies options to the LSF command. No default
