OpenSPARC™ T2 Processor Design and Verification User's Guide

Sun Microsystems, Inc.
www.sun.com

Part No. 820-2729-12
November 2008, Revision A

Submit comments about this document at: `http://www.sun.com/hwdocs/feedback`

Contents

Figures

Tables

Preface

The *OpenSPARC™ T2 Processor Design and Verification User's Guide* gives an overview of the design hierarchy on the OpenSPARC T2 processor. It also describes the files, procedures, and tools needed for running simulations and synthesis on the OpenSPARC T2 processor.

This book covers the following topics:

- Design and Verification implementation overview
- Design and Verification directory and files structure
- System and Electronic Design Automation (EDA) tools required to run simulations and synthesis
- Tools and scripts required to run simulation or complete regressions, including simulation flow
- Synthesis flow and scripts

How This Document Is Organized

Chapter 1 describes quick steps to run simulations after you download the design and verification files from the web site. It also includes system requirements and EDA tools requirements to run simulations and synthesis.

Chapter 2 gives an overview of the OpenSPARC T2 design hierarchy and directory structure.

Chapter 3 gives an overview of the OpenSPARC T2 verification environment implementation and directory structure. The verification environment includes test benches, tests, scripts, and Verilog Programming Language Interface (PLI).

Chapter 4 describes the synthesis flow and synthesis scripts.

Appendix A provides Design and Verification commands.

Using UNIX Commands

This document might not contain information about basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices. Refer to the following for this information:

- Software documentation that you received with your system
- Solaris™ Operating System documentation, which is at:

  `http://docs.sun.com`

Shell Prompts

Shell	**Prompt**
C shell	*machine-name*`%`
C shell superuser	*machine-name*`#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell superuser	`#`

Typographic Conventions

Typeface[*]	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`

`AaBbCc123`	What you type, when contrasted with on-screen computer output	`%` **`su`**
		`Password:`

AaBbCc123	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this. To delete a file, type `rm` *filename*.

* The settings on your browser might differ from these settings.

Related Documentation

The documents listed as online or download are available at:

`http://www.opensparc.net/`

Application	Title	Part Number	Format	Location
Documentation	*OpenSPARC T2 Core Microarchitecture Specification*	820-2545	PDF	Online
Documentation	*OpenSPARC T2 System-On-Chip (SoC) Microarchitecture Specification, Part 1 of 2*	820-2620	PDF	Online
Documentation	*OpenSPARC T2 System-On-Chip (SoC) Microarchitecture Specification , Part 2 of 2*	820-5090	PDF	Online
Documentation	*OpenSPARC T2 Processor Megacell Specification*	820-2728	PDF	Online
Documentation	*OpenSPARC T2 Processor Design and Verification User's Guide*	820-2729	PDF	Online
Documentation	*OpenSPARC T2 Behavioral Model Specification*	820-6778	PDF	Online

Documentation, Support, and Training

Sun Function	URL
Documentation	`http://www.sun.com/documentation/`
Support	`http://www.sun.com/support/`
Training	`http://www.sun.com/training/`

Third-Party Web Sites

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

`http://www.sun.com/hwdocs/feedback`

Please include the title and part number of your document with your feedback:

OpenSPARC T2 Processor Design and Verification User's Guide,
part number 820-2729-12

CHAPTER 1

Quick Start

This chapter covers the following topics:

- System Requirements
- EDA Tool Requirements
- Running Simulations and Synthesis

Before you start running simulations or synthesis, make sure you meet system requirements and that you have the required Electronic Design Automation (EDA) tools. Once you download the OpenSPARC T2 `tar` file from the `http://www.opensparc.net` web site, follow the steps in this chapter to get started and run your first regression on the OpenSPARC T2 design.

1.1 System Requirements

OpenSPARC T2 regressions are currently supported to run on SPARC systems running the Solaris 9 or Solaris 10 Operating System and x86_64 systems running Linux operating system.

Disk space requirements are listed in TABLE 1-1.

TABLE 1-1 Disk Space Requirements

Disk Space required	Required for:
3.1 Gbyte	Download, unzip or uncompress, and extract from the tar file
0.3 Gbyte	Run a mini-regression

TABLE 1-1 Disk Space Requirements *(Continued)*

Disk Space required	Required for:
11.5 Gbyte	Run a full regression
1.1 Gbyte	Run synthesis
17 Gbyte	Total

EDA Tool Requirements

TABLE 1-2 describes the commercial EDA tools required for running simulations for the OpenSPARC T2 processor and synthesizing OpenSPARC T2 Verilog Register Transfer Level (RTL) code.

TABLE 1-2 EDA Tools Requirements

	T2w/o IO SubSystem		T2 w/ IO SubSystem	
	Solaris 5.9 Solaris 5.10	Linux	Solaris 5.9 Solaris 5.10	Linux
EDA Simulation Tools				
VCS	2006.06-SP2-1	2006.06-4	2006.06-SP2-1	2006.06-4
Vera	X-2005.12-1	X-2005.12-11	X-2005.12-1	X-2005.12-11
NC Verilog	6.11.s3	06.20-s006	6.11.s3	06.20-s006
Debussy	2008.04	2008.04	2008.04	2008.04
Denali PureSpec	3.2.053	--	3.2.053	--
Software Tools				
C/C++ Compiler	gcc 3.3.2	gcc 3.3.2	gcc 3.3.2	gcc 3.3.2
EDA Synthesis Tools:				
Design Compiler	X2005.09	X2005.09	X2005.09	X2005.09

1.2 Running Simulations and Synthesis

This section outlines the steps needed to obtain the simulation tools, set up the simulation environment, run the simulation, and read its log file.

▼ Get the Simulation Files

1. **Download the file.**

   Download the OpenSPARCT2.tar.bz2 file from the http://www.opensparc.net web site.

For this procedure's examples, the destination directory is:

/home/johndoe/OpenSPARCT2

2. **Change directories to the directory where you downloaded the file. For example:**

```
% cd /home/johndoe/OpenSPARCT2
```

3. **Use the bunzip2 command to unzip the file.**

```
% bunzip2 OpenSPARCT2.tar.bz2
```

4. **Extract the tar file using the tar command.**

```
% tar -xvf OpenSPARCT2.tar
```

   This step creates the files and subdirectories listed in TABLE 1-3 in your current directory.

TABLE 1-3 Contents of the OpenSPARCT2 Directory

Name	Type	Description
OpenSPARCT2.cshrc	File	File to set up environment variables and paths for the SPARC/Solaris platform
OpenSPARCT2.cshrc.linux	File	File to set up environment variables and paths for the x64/Linux platform
README	File	Instructions to set up and run simulations
lib	Directory	Verilog libraries

TABLE 1-3 Contents of the OpenSPARCT2 Directory *(Continued)*

Name	Type	Description
`verif`	Directory	Verification directories and files
`design`	Directory	Verilog RTL for OpenSPARC T2 design
`tools`	Directory	Tools and scripts needed to run simulations and synthesis
`doc`	Directory	Documentation in PDF form for the OpenSPARC T2 processor

▼ Set Up Environment Variables

Edit the `OpenSPARCT2.cshrc` file to set the required environment variables as shown in TABLE 1-4:

TABLE 1-4 Environment Variables in `.cshrc` File

Environment Variable	Usage	Example value
`DV_ROOT`	Running simulations and synthesis	`/home/johndoe/OpenSPARCT2` (Directory where you ran the `tar` command above)
`MODEL_DIR`	Running simulations	`/home/johndoe/OpenSPARCT2_model` (Directory where you want to run your simulations)
`VERA_HOME`	Running simulations	`/import/EDAtools/vera/vera,v6.2.10/5.x` (Directory where Vera is installed)
`NOVAS_HOME`	Running simulations	`/import/EDAtools/debussy/debussy,v5.3v19/5.x` (Directory where Debussy is installed)
`VCS_HOME`	Running VCS simulations	`/import/EDAtools/vcs7.1.1R21` (Directory where VCS is installed)
`NCV_HOME`	Running NCV simulation	`/import/EDAtools/ncverilog/ncverilog,v6.11.s3/5.x` (Directory where NCV is installed)
`SYN_HOME`	Running synthesis	`/import/EDAtools/synopsys/synopsys.vX-2005.09` (Directory where Synopsys is installed)

TABLE 1-4 Environment Variables in `.cshrc` File *(Continued)*

Environment Variable	Usage	Example value
`DENALI_HOME`	Running Simulation with IO SubSystem using PureSpec Transactor	`/import/EDAtools/denali/v3.2.053` (Directory where Synopsys is installed)
`CC_BIN`	Compiling PLI code	`/import/freetools/local/gcc/3.3.2/bin` (Directory where C++ Compiler binaries are installed)
`LM_LICENSE_FILE`	Running simulations and synthesis	`/import/EDAtools/licenses/synopsys_key:/import/EDAtools/licenses/ncverilog_key` (EDA tool license files)

Note – For x64/Linux platform, edit OpenSPARCT2.cshrc.linux file to set required environment variables

---

Once you set the environment variables from TABLE 1-4, the `OpenSPARCT2.cshrc` file sets the following environment variables:

- `TRE_ENTRY`
- `TRE_SEARCH`
- `PERL_MODULE_BASE`
- `PERL_PATH`

The `OpenSPARCT2.cshrc` script also adds the following directories to your `PATH` and `path` variables:

- `$DV_ROOT/tools/bin`
- `$VCS_HOME/bin`
- `$VERA_HOME/bin`
- `$SYN_HOME/sparcOS5/syn/bin`
- `$CC_BIN`

After completing your `OpenSPARCT2.cshrc` file edits, source it by using the `source` command:

```
% source /home/johndoe/OpenSPARCT2/OpenSPARCT2.cshrc
```

You might want to include the above command in your `~/.cshrc` file so that the above environment variables are set every time you log in.

▼ Run Your First Regression

The OpenSPARC T2 Design/Verification package comes with four test bench environments: `cmp1`, `cmp8`, `fc1`, and `fc8`.

The `cmp1` environment consists of:

- One SPARC CPU core
- Cache
- Memory
- Crossbar

The `cmp1` environment does not have an I/O subsystem.

The `cmp8` environment consists of:

- Eight SPARC CPU cores
- Cache
- Memory
- Crossbar

The `cmp8` environment does not have an I/O subsystem.

The `fc1` environment consists of:

- A full OpenSPARC T2 chip with one SPARC Core
- Cache
- Memory
- Crossbar
- I/O subsystem

The `fc8` environment consists of:

- A full OpenSPARC T2 chip, including all eight cores
- Cache
- Memory
- Crossbar
- I/O subsystem

Each environment can perform either a mini-regression or a full regression.

To run a regression, use the `sims` command as described in To Run a Regression. The important parameters for the `sims` command are:

- `-sys`: system type

Set this to cmp1 or cmp8 or fc1 or `fc8`. For example: **`-sys=cmp1`**

- -group: Regression group name

The choices for -group are: cmp1_mini_T2, cmp1_all_T2, cmp8_mini_T2, cmp8_all_T2, fc1_mini_T2, and fc1_all_T2, fc8_mini_T2, and fc8_all_T2.

For example: -group=cmp1_mini_T2

- For help, type "sims -h"

▼ To Run a Regression

1. **Create the $MODEL_DIR directory.**

```
% mkdir $MODEL_DIR
```

2. **Change directory to $MODEL_DIR.**

```
% cd $MODEL_DIR
```

   This is where the simulations are run.

3. **Run a mini-regression for the cmp1 environment using the VCS simulator.**

```
% sims -sys=cmp1 -group=cmp1_mini_T2
```

   This command creates two directories:

   - A directory called cmp1 under $MODEL_DIR. The regression compiles Vera and Verilog code under the cmp1 directory. This is the Vera and Verilog "build" directory.
   - A directory named with today's date and a serial number, such as 2008_01_07_0 (the format is YYYY_MM_DD_ID) under the current directory where simulations will run. This is the Verilog simulation's "run" directory. There is one subdirectory under this directory for each diagnostics test.

   By default, the simulations are run with Vera.

4. **Once simulations are completed, run the regreport command to generate a regression report.**

```
% cd run-directory
% regreport $PWD > report.log
```

   Where *run-directory* is the "run" directory created in the above step, such as 2008_08_07_0.

The `cmp1_mini_T2` regression has 51 tests. An example of its `report.log` output is shown below:

Group	Total	PASS	FAIL	Cycles	Time	C/S
cmp1_st:	2	2	0	127399.00	2405.28	52.97
cmp1_nospec:	5	5	0	349747.50	8391.28	41.68
cmp1_mt:	15	15	0	684702.50	20884.47	32.79
cmp1_mmu:	9	9	0	245845.50	7335.27	33.52
cmp1_lsu:	5	5	0	190447.50	5876.04	32.41
cmp1_fast_idtlb:	4	4	0	102978.00	2579.26	39.93
cmp1_fast_fgu:	5	5	0	109217.50	2205.12	49.53
cmp1_fast_exu:	6	6	0	140307.00	3639.62	38.55
ALL:	51	51	0	1950644.50	53316.34	36.59

If your `report.log` file displays a similar status, you have successfully completed running a mini-regression for the OpenSPARC T2 processor.

▼ Run Your First Synthesis

The command to run a synthesis is `rsyn`. For example, to run a synthesis for one of the modules called `efu`, type:

```
% rsyn efu
```

This command runs a synthesis for the `efc` block and creates gate level netlists under the `$DV_ROOT/design/sys/iop/efu/synopsys/gate` directory.

The synthesis flow and scripts are described in more detail in Chapter 4.

CHAPTER 2

OpenSPARC T2 Design Implementation

This chapter gives details on the following topics:

2.1 OpenSPARC T2 Design

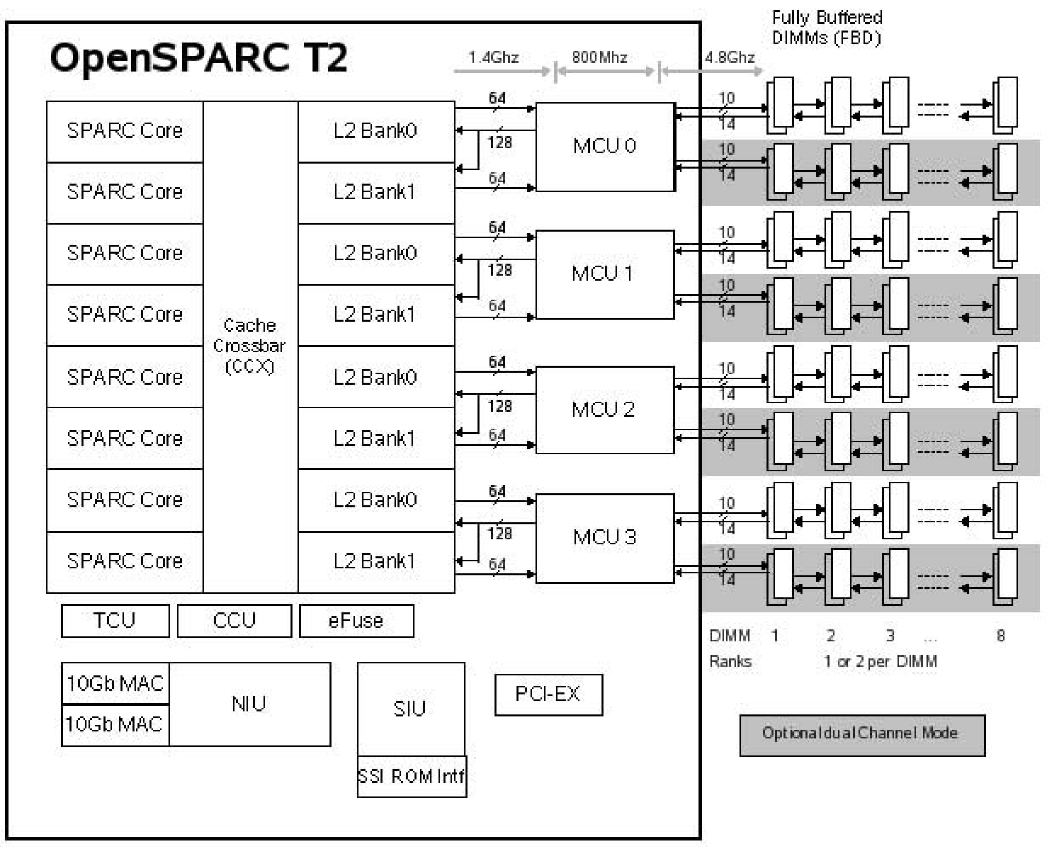
OpenSPARC T2 is a single chip multi-threaded (CMT) processor. OpenSPARC T2 contains eight SPARC physical processor cores. Each SPARC physical processor core has full hardware support for eight strands, two integer execution pipelines, one floating-point execution pipeline, and one memory pipeline. The floating-point and memory pipelines are shared by all eight strands. The eight strands are hard-partitioned into two groups of four, and the four strands within a group share a single integer pipeline. While all eight strands run simultaneously, at any given time at most two strands will be active in the physical core, and those two strands will be issuing either a pair of integer pipeline operations, an integer operation and a floating-point operation, an integer operation and a memory operation, or a floating-point operation and a memory operation. Strands are switched on a cycle-by-cycle basis between the available strands within the hard-partitioned group of four using a least recently issued priority scheme. When a strand encounters a long-latency event, such as a cache miss, it is marked unavailable and instructions will not be issued from that strand until the long-latency event is resolved. Execution of the remaining available strands will continue while the long-latency event of the first strand is resolved.

Each SPARC physical core has a 16 KB, 8-way associative instruction cache (32-byte lines), 8 Kbytes, 4-way associative data cache (16-byte lines), 64-entry fully-associative instruction TLB, and 128-entry fully associative data TLB that are shared by the eight strands. The eight SPARC physical cores are connected through a crossbar to an on-chip unified 4 Mbyte, 16-way associative L2 cache (64-byte lines). The L2 cache is banked eight ways to provide sufficient bandwidth for the eight SPARC physical cores. The L2 cache connects to four on-chip DRAM controllers, which directly interface to a pair of fully buffered DIMM (FBD) channels. In addition, an on-chip PCI-EX controller, two 1-Gbit/10-Gbit Ethernet MACs, and several on-chip I/O-mapped control registers are accessible to the SPARC physical cores. Traffic from the PCI-EX port coherently interacts with the L2 cache.

Note – OpenSPARC T2 currently does not include PCI-Express design implementation due to current legal restrictions. Equivalent model may be available in the subsequent releases of OpenSPARC T2.

A block diagram of the OpenSPARC T2 chip is shown in FIGURE 2-1

FIGURE 2-1 OpenSPARC T2 Block Diagram



2.2 OpenSPARC T2 Components

This section describes each component in OpenSPARC T2.

2.2.1 SPARC Physical Core

Each SPARC physical core has hardware support for eight strands. This support consists of a full register file (with eight register windows) per strand, with most of the ASI, ASR, and privileged registers replicated per strand. The eight strands share the instruction and data caches and TLBs. An auto-demap feature is included with the TLBs to allow the multiple strands to update the TLB without locking.

There is a single floating-point unit within each SPARC physical core for a total of eight on a T2 chip. Each floating-point unit is shared by all eight strands and fully pipelined. The theoretical floating-point bandwidth is 11 Giga Floating Point Ops (GFlops) per second making the T2 an excellent floating-point processor.

Detailed information on the core processor is provided in *OpenSPARC T2 Core Microarchitecture Specification*, see Related Documentation.

2.2.2 SPARC System-On-Chip (SoC)

Each SPARC physical core is supported by system-on-chip hardware components.

Detailed Information on the functioning units of the system-on-chip of OpenSPARC T2 are provided in the *OpenSPARC T2 System-On-Chip (SoC) Microarchitecture Specification*, see Related Documentation.

2.3 Module Directory Structure

The Verilog RTL for the OpenSPARC T2 processor is in the `$DV_ROOT/design/sys/iop/cpu` directory. The top level verilog module for the OpenSPARC T2 processor is called “cpu”. All the top-level modules that make up that RTL, and their locations, are listed in TABLE 2-1.

TABLE 2-1 OpenSPARC T2 Top-Level Clusters

Module Name	Number of Instances	Instance Names	Directory Location under `$DV_ROOT/design/sys/iop`	Description
ccu	1	ccu	ccu	Clock Control Unit
ccx	1	ccx	ccx	CPU-Cache Cross bar
db0	1	db0	db0	Debug Unit
db1	1	db1	db1	Debug Unit
dmu	1	dmu	dmu	Data Management Unit
efu	1	efu	efu	e-Fuse Cluster
fsr	1	fsr	fsr	FBDIMM Serdes macro
fsr_bottom	1	fsr_bottom	fsr_bottom	FBDIMM Serdes Macro
fsr_left	1	fsr_left	fsr_left	FBDIMM Serdes Macro
fsr_right	1	fsr_right	fsr_right	FBDIMM Serdes Macro
l2b	8	l2b[0-7]	l2b	L2$ bank
l2t	8	l2t[0-7]	l2t	L2 $ tag
l2d	8	l2d[0-7]	l2d	L2 $ data
mcu	4	mcu[0-3]	mcu	Memory Controller
mio	1	mio	mio	Miscellaneous I/O
ncu	1	ncu	ncu	Non-cacheable Unit
rst	1	rst	rst	Reset Unit
sii	1	sii	sii	System Interface Unit - Inbound
sio	1	sio	sio	System Interface Unit - Outbound
spc	8	spc[0-7]	spc	SPARC CPU core
tcu	1	tcu	tcu	Trap Control Unit
esr	1	esr	esr	Ethernet SerDes model

TABLE 2-1 OpenSPARC T2 Top-Level Clusters *(Continued)*

Module Name	**Number of Instances**	**Instance Names**	**Directory Location under** `$DV_ROOT/design/sys/iop`	**Description**
`mac`	1	`mac`	`mac`	Ethernet MAC
`tds`	1	`tds`	`tds`	Ethernet Transmit Data Path
`rdp`	1	`rdp`	`rdp`	Ethernet Receive Data Path
`rtx`	1	`rtx`	`rtx`	Ethernet Receive and Transmit

2.4 Megacells

The OpenSPARC T2 design contains many megacells, which are custom blocks for static random access memory (SRAMs), translation lookaside buffer (TLB), TAGs, Level 2 Cache, and so on. These megacells are instantiated in the top-level clusters. The detailed descriptions of all megacells, including their function descriptions, I/O lists, block diagrams, and timing diagrams, are in the *OpenSPARC T2 Megacell Specification*, see Related Documentation.

CHAPTER 3

OpenSPARC T2 Verification Environment

This chapter describes the following topics:

3.1 OpenSPARC T2 Verification Environment

The OpenSPARC T2 verification environment is a highly automated environment. With a simple command, you can run the entire regression suite for the OpenSPARC T2 processor, containing hundreds of tests. With a second command, you can check the results of the regression.

The OpenSPARC T2 Design and Verification package comes with four test bench environments: `cmp1, cmp8, fc1 and` `fc8`.

The `cmp1` environment consists of:

- One SPARC CPU core
- Cache
- Memory
- Crossbar

The `cmp1` environment does not have an I/O subsystem.

The cmp8 environment consists of:

- Eight SPARC CPU cores
- Cache
- Memory
- Crossbar

The cmp8 environment does not have an I/O subsystem.

The fc1 environment consists of:

- A full OpenSPARC T2 chip, with one SPARC Core
- Cache
- Memory
- Crossbar
- I/O subsystem

The fc8 environment consists of:

- A full OpenSPARC T2 chip, including all eight cores
- Cache
- Memory
- Crossbar
- I/O subsystem

The I/O subsystem includes PCI-Express transaction level behavioral SystemC model. For the Network Interface Unit, it can build with RTL, or Behavioral SystemC model. TABLE 3-1 shows different IO subsystem build options for the FC1 and FC8 environment.

TABLE 3-1 Config CPP Arguments for the IO Subsystem Build

config_cpp_args	Description
	Default mode for the Fullchip build, No PCI-Express and Network Interface units included.
-DPEU_SYSC_NIU_RTL	Includes PCI-Express behavioral model and Network Interface Unit RTL design.
-DPEU_SYSC_NIU_SYSC	Includes PCI-Express and Network Interface Unit behavioral SystemC model.

The verification environment uses source code in various languages. TABLE 3-2 shows a summary of the types of source code and their uses.

TABLE 3-2 Source Code Types in the Verification Environment

Source Code Language	Used for:
Verilog	Chip design, test bench drivers, and monitors.
Vera	Test bench drivers, monitors, and coverage objects. Use of Vera is optional.
SystemC	Transaction Level Behavioral Model for the SOC interface modules.
PERL	Scripts for running simulations and regressions.
C and C++	PLI (Programming Language Interface) for Verilog.
SPARC Assembly	Verification tests.

3.2 Running a Regression

For each environment, there is a mini-regression and a full regression. TABLE 3-3 describes the regression groups.

TABLE 3-3 Details of Regression Groups

Regression Group name	Environment	No. of Tests
`cmp1_mini_T2`	`cmp1`	6
`cmp1_all_T2`	`cmp1`	768
`cmp8_mini_T2`	`cmp8`	7
`cmp8_all_T2`	`cmp8`	648
`fc1_mini_T2`	`fc1`	6
`fc1_full_T2`	`fc1`	350
`fc1_full_T2 with PIU/NIU subsystem`	`fc1`	440
`fc8_mini_T2`	`fc8`	17
`fc8_full_T2`	`fc8`	535
`fc8_full_T2 with PIU/NIU subsystem`	`fc8`	577

▼ To Run a Regression

1. **Run the `sims` command with your chosen parameters.**

   For instance, to run a mini-regression for the `cmp1` environment using the VCS simulator, set up the `sims` command as follows:

```
% sims -sys=cmp1 -group=cmp1_mini_T2
```

   To run regressions on multiple groups at the same time, specify multiple `-group=` parameters at the same time. For a complete list of command-line options for the `sims` command, see Appendix A.

2. **Run the `regreport` command to get a summary of the regression.**

```
% regreport $PWD/2007_08_07_0 > report.log
```

3.2.1 What the `sims` Command Does

When running a simulation, the `sims` command performs the following steps:

1. Compiles the design into the `$MODEL_DIR/cmp1` or `$MODEL_DIR/fc8` directory, depending on which environment is being used.
2. Creates a directory for regression called *$PWD/DATE_ID*, where *$PWD* is your current directory, *DATE* is in `YYYY_MM_DD` format, and *ID* is a serial number starting with 0. For example, for the first regression on August07, 2007, a directory called *$PWD*`/2007_08_07_0` is created. For the second regression run on the same day, the last ID is incremented to become *$PWD*`/2007_08_07_1`.
3. Creates a `master_diaglist.`*regression_group* file under the above directory. such as `master_diaglist.cmp1_mini_`T2 for the `cmp1_mini_T2` regression group. This file is created based on diaglists under the `$DV_ROOT/verif/diag` directory.
4. Creates a subdirectory with the test name under the regression directory created in step 2 above.
5. Creates a `sim_command` file for the test based on the parameters in the diaglist file for the group.
6. Executes `sim_command` to run a Verilog simulation for the test. If the `-sas` option is specified for the test, it also runs the SPARC Architecture Simulator (SAS) in parallel with the Verilog simulator. The results of the Verilog simulation are compared with the SAS results after each instruction.

The sim_command command creates many files in the test directory. Following are the sample files in the test directory:

```
diag.ev        diag.s        raw_coverage   seeds.log
status.log     vcs.log.gz diag.exe.gz   midas.log
sas.log.gz     sims.log      symbol.tbl     vcs.perf.log
```

The status.log file has a summary of the status, where the first line contains the name of the test and its status (PASS/FAIL).

```
Rundir: tlu_rand05_ind_03:cmp1_st:cmp1_mini_T2:0   PASS
```

7. Repeats steps 4 to 6 for each test in the regression group.

3.3 PLI Code used for the Test Bench

Verilog's PLI (Programming Language Interface) is used to drive and monitor the simulations of the OpenSPARC T2 design. There are eight different directories for PLI source code. Some PLI code is in C language, and some is in C++ language. TABLE 3-4 gives the details of PLI code directories and VCS libraries.

TABLE 3-4 PLI Source Code and Object Libraries

PLI Name	Source Code location under $DV_ROOT	VCS Object Library Name	Description
iob	verif/env/common/pli/cache	libiob.a	Cache warming routines
mem	model/infineon	libbwmem_pli.a	Memory read/write
socket	verif/env/common/pli/socket	libsocket_pli.a	Sockets to SAS
utility	verif/env/common/pli/utility	libbwutility_pli.a	Utility functions
monitor	verif/env/common/pli/monitor/c	libmonitor_pli.a	Various
global_chkr	verif/env/common/pli/global_chkr/	libglobal_chkr.a	Various checkers

VCS object libraries are statically linked libraries (.a files) which are linked when VCS compiles the Verilog code to generate a simv executable.

Makefile is provided to compile PLI code. There is a makefile file under $DV_ROOT/tools/pli directory which will compile static executable (.a file) of the PLI code.

3.4 Behavioral Model Files

The simulation verification environment provides SystemC behavioral simulation models for the IO sub-systems; Network Interface and PCI-Express Interface units.

The SystemC behavioral models are tested with the VCS 2006-06-SP2-1 and SystemC 2.2. You can find the SystemC model files in:

TABLE 3-5 Behavioral Model File Directories

Directory	Contents
`$DV_ROOT/verif/model/pcie`	PCI Express behavioral model
`$DV_ROOT/verif/model/systemc/niu`	Network Interface behavioral model

3.5 Verification Test File Locations

The verification or diagnostics tests (diags) for the OpenSPARC T2 processor are written in SPARC assembly language (the file names have a `.s` extension). Some diagnostics test cases in SPARC assembly are automatically generated by Perl scripts.

The main diaglist for `cmp1` is `cmp1.diaglist`. The main diaglist for `fc8` is `fc8.diaglist`. These main diaglists for each environment also include many other diaglists. The locations of various verification test files are listed in TABLE 3-6.

TABLE 3-6 Verification Test File Directories

Directory	Contents
`$DV_ROOT/verif/diag`	All diagnostics, various diagnostic list files with the extension`.diaglist`.
`$DV_ROOT/verif/diag/assembly`	Source code for SPARC assembly diagnostics. More than 1400 assembly test files.
`$DV_ROOT/verif/diag/efuse`	EFuse cluster default memory load files.

CHAPTER 4

OpenSPARC T2 Synthesis

This chapter describes the following topics:

- Synthesis Flow for the OpenSPARC T2 Processor
- Synthesis Output

The scripts provided in the source code are for the Synopsys Design Compiler.

4.1 Synthesis Flow for the OpenSPARC T2 Processor

There are two types of synthesis scripts:

- One set to run the Synopsys Design Compiler (`rsyn` and `syn_command`)
- One set used as input for the Design Compiler

The main script used to run Synopsys Design Compiler is called `rsyn`. This is a PERL script that calls a second script, `syn_command`, once for each module you are synthesizing. The command-line options for the `rsyn` script are described in CODE EXAMPLE 4-1.

CODE EXAMPLE 4-1 Command-Line Options for `rsyn` Script

```
rsyn : Run Synthesis for OpenSPARC T2

     -all
         to run synthesis for all blocks
     -h / -help
         to print help
     -syn_q_command='Your job Queue command'
         to specify Job queue command. e.g. specify submit command
         for LSF or GRID
     block_list :
         specify list of blocks to synthesize

     Examples:

     rsyn -all
     rsyn efu
```

Synthesis scripts for most of the modules are provided in the `$DV_ROOT/design` sub-directories. There are no synthesis scripts for the following types of modules:

- Megacell modules (SRAMS, TLB, TAG, Cache, etc.)
- Top-level hierarchical modules

Synopsys scripts, their locations, and their descriptions are listed in TABLE 4-1.

TABLE 4-1 Synthesis Script Details

Script name	Location	Description
`run.scr`	`$DV_ROOT/design/sys/synopsys/script`	Main synthesis script that calls `user_cfg.scr`
`project_sparc_cfg.scr`	`$DV_ROOT/design/sys/synopsys/script`	SPARC module-specific synthesis script
`project_io_cfg.scr`	`$DV_ROOT/design/sys/synopsys/script`	I/O module-specific synthesis script
`target_lib.scr`	`$DV_ROOT/design/sys/synopsys/script`	Target library-specific script
`user_cfg.scr`	*Module directory*`/synopsys/script`	Module-specific synthesis script

The top-level Synopsys script, `run.scr`, calls the module-specific script named `user_cfg.scr`. The `user_cfg.scr` script calls the `project_sparc_cfg.scr` script or the `project_io_cfg.scr` script, depending on whether the module belongs to `sparc` or `io`.

The list of all modules with synthesis scripts is in the `$DV_ROOT/design/sys/synopsys/block.list` file.

Each module has:

- A `synopsys` directory under the module directory
- A `script` directory under each `synopsys` directory
- The `user_cfg.scr` file under the `script` directory

For example, the `efc` module-specific synthesis script has the following directory path:

```
$DV_ROOT/design/sys/iop/efu/synopsys/script/user_cfg.scr
```

The target library is set to a generic library called `lsi_10k.db` in the `target_lib.scr` script. Modify this file to set your own target library and its required variables.

4.2 Synthesis Output

Running synthesis for a module creates files and directories under the *Module name*/`synopsys` directory, described in TABLE 4-2.

TABLE 4-2 Synthesis Output

Name	Type	Description
`dc_shell.log`	File	Log file from running Design Compiler
`command.log`	File	Command log from running Design Compiler
`log`	Directory	Area report files from Design Compiler
`gate`	Directory	Gate netlist generated by Design Compiler
`.template`	Directory	Template directory used by Design Compiler

APPENDIX **A**

Design and Verification Commands

This appendix provides the commands used in OpenSPARC T2 design and verification.

A.1 sims

NAME

sims - Verilog rtl simulation environment and regression script

SYNOPSIS

sims [args ...]

Note – Use "=" instead of "space" to separate args and their options.

where args are:

SIMULATION ENV

```
    -sys=NAME
            sys is a pointer to a specific testbench configuration
            to be built and run. a config file is used to associate
            the sys with a set of default options to build the
            testbench and run diagnostics on it. the arguments
            in the config file are the same as the arguments passed
            on the command line.
    -group=NAME
            group name identifies a set of diags to run in a
            regression. The presence of this argument indicates
            that this is a regession run. the group must be found
            in the diaglist. multiple groups may be specified to be
            run within the same regression.
```

Note – If -sys=NAME option is specified then NAME.diaglist is used as root diaglist instead of the master diaglist.

```
    -group=NAME -alias=ALIAS
            this combination of options gets the diag run time options
            from the diaglist based on the given group and alias.
            the group must be found in the diaglist. the alias is
            made up of diag_alias:name_tag. only one group should be
            specified when using this command format.
```

VERILOG COMPILATION RELATED

```
    -sim_q_command="command"
          defines which job queue manager command to use to launch jobs.
          Defaults to /bin/sh and runs simulation jobs on the local machine.

    -vcs_build/-novcs_build
           builds a vcs model and the vera testbench. defaults to off.

    -sysc_build
           builds the systemc behavioral model. This is required to build any IO
          subsystem behavioral model. Currently supported only with the vcs_build.

    -vcs_build_args=OPTION
           vcs compile options. multiple options can be specified using
           multiple such arguments.

    -vcs_clean/-novcs_clean
           wipes out the model directory and rebuilds it from scratch.
           defaults to off.

    -vcs_full64
           sets the vcs -full64 compile flag so that the compiler is a
           64 bit executable, and produces a 64 bit executable simv.
           will use the 64 bit version of vera, and link in the 64 bit
           versions of 0in, debussy, and denali tools.

    -vcs_use_2state/-novcs_use_2state
           builds a 2state model instead of the default 4state model.
```

```
        this defaults to off.

-vcs_use_initreg/-novcs_use_initreg
        initialize all registers to a valid state (1/0).
        this feature works with -tg_seed to set the seed of the random
        initialization. this defaults to off.

-vcs_use_fsdb/-novcs_use_fsdb
        use the debussy fsdb pli and include the dump calls in the
        testbench. this defaults to on.

-vcs_use_vcsd/-novcs_use_vcsd
        use the vcs direct kernel interface to dump out debussy files.
        this defaults to on.

-vcs_use_vera/-novcs_use_vera
        compile in the vera libraries. if -vcs_use_ntb and -vcs_use_vera are
        used, -vcs_use_ntb wins.
        this defaults to off.

-vcs_use_ntb/-novcs_use_ntb
        enable the use of NTB when building model (simv) and running simv.
        if -vcs_use_ntb and -vcs_use_vera are used, -vcs_use_ntb wins.
        this defaults to off.

-vcs_use_rad/-novcs_use_rad
        use the +rad option when building a vcs model (simv).
        defaults to off.

-vcs_use_sdf/-novcs_use_sdf
        build vcs model (simv) with an sdf file.
        defaults to off.

-vcs_use_radincr/-novcs_use_radincr
        use incremental +rad when building a vcs model (simv).
        defaults to off.
       (This is now permanently disabled as synopsys advises against
        using it.)

-vcs_use_cli/-novcs_use_cli
        use the +cli -line options when building a vcs model (simv).
        defaults to off.

        use this switch, in conjunction with -nosimslog during runtime
        if you need to pass ctrl-c to the vcs/axis model and continue
        with CLI activity.
```

```
Use this with VCS versions before 2006.

    -vcs_use_ucli/-novcs_use_ucli (Unified cli)
           use the -debug_all option when building a vcs model (simv).
           defaults to off.

           use this switch, in conjunction with -nosimslog during runtime
           if you need to pass ctrl-c to the vcs/axis model and continue
           with UCLI activity.

           At runtime, use -vcs_run_arg=-ucli to get the UCLI at time zero,
           or          use -vcs_run_arg=-gui  to get the UCLI GUI at time zero.
           At runtime, use -vcs_run_arg=-tbug to get NTB debug in the GUI.

Use this with VCS versions 2006 and up.

    -flist=FLIST
           full path to flist to be appended together to generate the
           final verilog flist. multiple such arguments may be used and
           each flist will be concatenated into the final verilog flist
           used to build the model.

    -graft_flist=GRAFTFILE
           GRAFTFILE is the full path to a file that lists each verilog
           file that will be grafted into the design. the full path to
           the verilog files must also be given in the GRAFTFILE.

    -vfile=FILE
           verilog file to be included into the flist

    -config_rtl=DEFINE
           each such parameter is placed as a `define' in config.v to
           configure the model being built properly. this allows
           each testbench to select only the rtl code that it needs
           from the top level rtl file (ciop.v in blackwidow).

    -model=NAME
           the name of a model to be built. the full path to a model
           is MODEL_DIR/model/vcs_rel_name.

    -vcs_rel_name=NAME
            specify the release of the model to be built. the full path
            to a model is MODEL_DIR/model/vcs_rel_name.

VERA/NTB COMPILATION RELATED

    VERA and NTB share all of the vera options except a few.
    See NTB RELATED.
    -vera_build/-novera_build
           builds the vera/ntb testbench. default on.

    -vera_clean/-novera_clean
           performs a gmake clean on the vera/ntb testbench before building
           the model. defaults to off.

-vera_build_args=OPTION
           vera/NTB testbench compile time options.
           Multiple options can be specified using multiple such
           options. these are passed as arguments to the gmake call
           when building the vera/NTB testbench.
           (Eg: -vera_build_args=VERA_SYS_DEFS="-DSPC_BENCH -DGATESIM")

           For NTB, -vera_build_args=NTB_BUILD_ARGS="+error+10 -ntb_define ABCD"
           can be used to add something directly to the "vcs -ntb_cmp" command.
           For the -ntb_lib option, NTB_BUILD_ARGS will affect both the vshell and
           bench+diag builds. See -vera_diag_args to not affect the vshell build.

    -vera_diag_args=OPTION
           vera/ntb diag compile time options.
          Multiple options can be specified using multiple such
           options.  For Vera, these args are appended to the
          "vera -cmp ..." command for the diag only.
          (Eg: -vera_diag_args="-max_error 10" or
            -vera_diag_args=-DNCU_ACK_DLY1=100)

           For NTB, these args are passed as arguments to the gmake
           call as NTB_DIAG_ARGS=" ..." to be part of the NTB
           bench+diag compile. These NTB_DIAG_ARGS are appended to
           the "vcs -ntb_cmp" command when making the libtb.so so
           they better be legal in that context. For NTB, these args
           really affect the entire bench build, not just the diag,
           BUT they they do not affect the vshell build.
           (Eg. -vera_diag_args=+error+10 or
           -vera_diag_args="-ntb_define NCU_ACK_DLY1=100")

    -vera_dummy_diag=PATH

           This option is used to give vera/NTB a path to a default
           diag or diag class (or a default program top if using RVM)
           that can be used for building purposes before an actual
           diag is chosen (NTB build of vshell file before regression
           for example).

           Also, some benches may run both asm and vera diags but the
           vera diags are only run sometimes. In this case, you need a
           dummy vera testcase class/program top to fill in when an
           actual vera diag is not being used for that run (aka the
           sometimes diag problem). If your openVera code refers to a
           testcase class, you better have one even if it does nothing
           else you will not be able to build. The dummy lets you
           build.  If your diag is implemented as the program top
           (RVM) then the dummy must have #includes for ALL of your
           vera interfaces or your vshell will be broken.

          Whenever the actual vera/NTB diag is specified, the dummy
          is automatically *not* used (at regression time for
          example).  The vera_dummy_diag should be specified in the
          bench config file.  This option applies to Vera and NTB
          but NTB only when using the -ntb_lib option).

     -vera_pal_diag_args=OPTION
           vera/ntb pal diag expansion options
          (i.e. "pal OPTIONS -o diag.vr diag.vrpal")
          multiple options can be specified using multiple such arguments.

     -vera_proj_args=OPTION
           vera proj file generation options. multiple options can be
           specified using multiple such arguments.

    -vera_vcon_file=ARG
           name of the vera vcon file that is used when running the simulation.

    -vera_cov_obj=OBJ
           this argument is passed to the vera Makefile as a OBJ=1 and to
           vera as -DOBJ to enable a given vera coverage object. multiple
           such arguments can be specified for multiple coverage objects.

    -vera_gmake/-novera_gmake
          this argument optionally lets the flow skip running gmake for the
          vera/NTB build, while maintaining other operations within the
          -vera_build flow. default ON (execute gmake)
```

NTB RELATED

```
    NTB and VERA share all of the vera options except these:

      -vcs_use_ntb/-novcs_use_ntb
            enable the use of NTB (compiled vera) rather than the
            conventional Vera.  if -vcs_use_ntb and -vcs_use_vera are
            used at once, then -vcs_use_ntb wins.  defaults to off.

      -ntb_lib/-nontb_lib
            enables the NTB 2 part compile where the openVera files
            get compiled separately into a libtb.so file which is
            dynamically loaded by vcs at runtime. The libtb.so file
            is built by the Vera/NTB Makefile, not sims. Use the
            Makefile to affect the build. If not using -ntb_lib, sims
            will build VCS and the openVera files together in one
            pass (uses Makefile to affect that build as well). default
            is off.

            The ntb_lib method is know as the NTB LIB method. When not
            using this ntb_lib method, the ALL IN ONE method is used.

            The NTB LIB method allows the bench to run unique openVera diags
            that are separate from the bench (via a diaglist if desired).

            The NTB ALL IN ONE method does not allow the bench to run
            unique openVera diags that are separate from the bench.
            Use this for benches that do not run openVera diags
            (perhaps the bench only runs asm diags
```

VERILOG RUNTIME RELATED

```
    -vera_run/-novera_run
           runs the vcs simulation and loads in the vera proj file
           or the ntb libtb.so file. defaults to on.

    -vcd/-novcd
           signals the bench to dump in VCD format

    -vcdfile=filename
           the name of the vcd dump file.  if the file name starts with
           a "/", that is the file dumped to, otherwise, the actual file is
           created under tmp_dir/vcdfile and copied back to the current
           directory when the simulation ends.  use "-vcdfile=`pwd`/filename"
           to force the file to be written in the current directory directly
           (not efficient since dumping is done over network instead of to
           a local disk).

    -vcs_run/-novcs_run
           runs the vcs simulation (simv). defaults to off.

    -vcs_run_args=OPTION
           vcs (simv) runtime options. multiple options can be specified
           using multiple such arguments.

           The order of vcs_run_args (plusargs) given to simv is:
                  args embedded in diag (using !SIMS+ARGS: ..), if any
                  args given in the command line, if any
                  args from diaglist : alias definition, if any
                  args from diaglist : <runargs>..</runargs>, if any
                  args from the config file, if any
```

```
-vcs_finish=TIMESTAMP
       forces vcs to finish and exit at the specified timestamp.

-fast_boot/-nofast_boot
       speeds up booting when using the ciop model. this passes the
       +fast_boot switch to the simv run and the -sas_run_args=-DFAST_BOOT
       and -midas_args=-DFAST_BOOT to sas and midas. Also sends
       -DFAST_BOOT to the diaglist and config file preprocessors.

-debussy/-nodebussy
       enable debussy dump. this must be implemented in the testbench
       to work properly. defaults to off.

-start_dump=START
       start dumping out a waveform after START number of units

-stop_dump=STOP
       stop dumping out a waveform after STOP number of units

-fsdb2vcd
      runs fsdb2vcd after the simulation has completed to generate a vcd file.

 -fsdbfile=filename
        the name of the debussy dump file.
     If the file name starts with a "/", that is the file dumped to,
     otherwise, the actual file is created under tmp_dir/fsdbfile
     and copied back to the current directory when the simulation ends.
     Use "-fsdbfile=`pwd`/filename" to force the file to be
     written in the current directory directly (not efficient since
     dumping is done over network instead of to a local disk).

-fsdbDumplimit=SIZE_IN_MB
     max size of Debussy dump file.  minimum value is 32MB.
      Latest values of signal values making up that size is saved.

-fsdb_glitch
     turn on glitch and sequence dumping in fsdb file. this will collect
     glitches and sequence of events within time in the fsdb waveform.
     beware that this will cause the fsdb file size to grow significantly.
     this is turned off by default. this option effectively does this:
     setenv FSDB_ENV_DUMP_SEQ_NUM 1
     setenv FSDB_ENV_MAX_GLITCH_NUM 0

-rerun
       rerun the simulation from an existing regression run directory.

-overwrite
       overwrite current run dir when doing a -rerun. default is to
       create a rerun_<n> subdir for reruns.
```

```
    -post_process_cmd=COMMAND
           post processing command to be run after vcs (simv) run completes

    -pre_process_cmd=COMMAND
           pre processing command to be run before vcs (simv) run starts

    -use_denalirc=FILE
          use FILE as the .denalirc in the run area. Default copies
          env_base/.denalirc
```

SUNV OPTIONS

```
         -sunv_run/-nosunv_run
                runs the sunv program to convert structural files,
                e.g. <file>.sv to verilog. defaults to off.

         -sunv_args=ARGS
                sunv options. Multiple options can be specified using
                multiple such arguments.  In addition, a portion of
                these arguments can be provided in a file using the
                sunv option -optfile=<file>.

         -sunv_use_nonprim/-nosunv_use_nonprim
                use a list to hold primitives that we want to remove from the
                default primitive.list. defaults to off.

         -sunv_nonprim_list=FILE
                name of file holding the list of primitives that we want to remove.
               this is only used if -sunv_use_nonprim is specified.
```

VLINT OPTIONS

```
         -vlint_run/-novlint_run
                runs the vlint program. defaults to off.

         -vlint_args
                vlint options. The <sysName>.config file can contain
                the desired vlint arguments, or they can also be given on
                the command line.  Typically the -vlint_compile is given
                on the command line.

                vlint also requires identification of a rules deck.

         -illust_run
                run illust after x2e

         -illust_args
                illust options
```

```
        -vlint_top
               top level module on which to run vlint

VERIX OPTIONS

    -verix_run/-noverix_run
           runs the verix program. defaults to off.

    -verix_libs
           specify the library files to add to the vlist

    -verix_args
           verix template options. The <sysName>.config file can contain
           these desired verix arguments

           verix also requires <top>.verix.tmplt in the config dir.

    -verix_top
           top level module on which to run verix


THARAS HAMMER RELATED

    -hcs_build
           build a model to be run on the Hammer Hardware Accelerator.

    -hcs_build_args
           build arguments for Hammer Hardware Accelerator

    -hcs_run
           run a model on the Hammer Hardware Accelerator.

    -hcs_run_args
           run arguments for the Hammer Hardware Accelerator.

    -hcs_drm_tokens
           tokens for drmsubmit licenses

AXIS RELATED

    -axis_build
           build a model to be run on the Axis Hardware Accelerator.

    -axis_build_args
           build arguments for Axis Hardware Accelerator

    -axis_run
           run a model on the Axis Hardware Accelerator.
    -axis_run_args
           run arguments for the Axis Hardware Accelerator.
```

PALLADIUM RELATED

```
    -palladium_build
           build a model to be run on the palladium Hardware Accelerator.

    -palladium_build_args
           build arguments for palladium Hardware Accelerator

    -palladium_run
           run a model on the palladium Hardware Accelerator.


    -palladium_run_args
           run arguments for the palladium Hardware Accelerator.
```

ZEROIN RELATED

```
    -zeroIn_checklist
           run 0in checklist

    -zeroIn_build
           build 0In pli for simulation into vcs model

    -zeroInSearch_build
           build 0in search pli for simulation into vcs model

    -zeroIn_build_args
           additional arguments to be passed to the 0in command

    -zeroIn_dbg_args
           additional debug arguments to be passed to the 0in shell
```

SAS/SIMICS RELATED

```
    -sas/-nosas
           run architecture-simulator. If vcs_run option is OFF,
           simulation is sas-only. If vcs_run option is ON, sas
           runs in lock-step with rtl. default to off.

    -sas_run_args=DARGS
           Define arguments for sas.
```

```
TCL/TAP RELATED

    -tcl_tap/-notcl_tap
           run tcl/expect TAP program. If vcs_run option is OFF,
           simulation is tcl-only. If vcs_run option is ON, tcl
           runs in lock-step with rtl. default to off.
```

Note – You _must_ compile with -tcl_tap as well, to enable to enable functions that are needed for running with tcl

```
    -tcl_tap_diag=diagname
           Define top level tcl/expect diag name.

MIDAS RELATED

midas is the diag assembler

    -midas_args=DARGS
           arguments for midas. midas creates memory image and user-event
           files from the assembly diag.

    -midas_only
           Compile the diag using midas and exit without running it.

    -midas_use_tgseed
           Add -DTG_SEED=tg_seed to midas command line. Use -tg_seed to
           set the value passed to midas or use a random value from /dev/random.

PCI

pci is the tomatillo pci bus functional model

    -pci_args
           arguments to be passed in to pci_cmdgen.pl for generation of a pci
           random diagnostic.

    -pci/-nopci
           generates a random pci diagnostic using the -tg_seed if provided.
           default is off.

    -tomatillo
           generates a random tomatillo diagnostic using the -tg_seed if provided

    -tg_seed
           random generator seed for pci/tomatillo random test generators
           also the value passed to +initreg+ to randomly initialize registers
           when -vcs_use_initreg is used.
```

```
SJM RELATED

sjm is the jalapeno jbus bus functional model

    -sjm_args
           arguments to be passed in to sjm_tstgen.pl for generation of an sjm
           random diagnostic.

    -sjm/-nosjm
           generates a random sjm diagnostic using the -tg_seed if provided.
           default is off.

    -tomatillo
           generates a random tomatillo diagnostic using the -tg_seed if provided

    -tg_seed
           random generator seed for sjm/tomatillo random test generators
           also the value passed to +initreg+ to randomly initialize registers
           when -vcs_use_initreg is used.


EFCGEN

efcgen.pl is a script to generate efuse.img files (default random),
which is used by the efuse controller after reset.
It is invoked by -efc.

    -efc/-noefc
           generates an efuse image file using the -tg_seed if provided.
           default is off.  Random if no -efc_args specified.

    -efc_args
           arguments to be passed in to efcgen.pl for generation of
     an efuse image file.
     Default is random efuse replacement for each block.

    -tg_seed
           random generator seed for efcgen.pl script
           also the value passed to +initreg+ to randomly initialize
     registers when -vcs_use_initreg is used.

VCS COVERMETER

    -vcs_use_cm/-novcs_use_cmd
           passes in the -cm switch to vcs at build time and simv at runtime
           default to off.

    -vcs_cm_args=ARGS
           argument to be given to the -cm switch
    -vcs_cm_cond=ARGS
           argument to be given to the -cm_cond switch.

    -vcs_cm_config=ARGS
           argument to be given to the -cm_hier switch

    -vcs_cm_fsmcfg=ARGS
           argument to be given to the -cm_fsmcfg switch
     specifies an FSM coverage configuration file

    -vcs_cm_name=ARGS
           argument to be given to the -cm_name switch. defaults to cm_data.
```

DFT

```
    -dftvert
          modifies the sims flow to accomodate dftvert. this skips compiling
          the vera testbench and modifies the simv command line at runtime.
```

CDMS

```
    -cdms_rel_name=CDMSREL
           specify the cdms++ release that must be collected for this
           model.

    -diff_cdms_rel
          performs a diff_release of CDMSREL from -cdms_rel_name and
          records it in a file called diff_rel.log lcoated in the model
          area. This file is copied into each run directory from the
          model area at runtime.

    -diff_cdms_curr
          uses the current (in localdir) release of CDMSREL for the
          diff_release command. Ignored if -cdms_rel_name and -diff_cdms_rel
          are not specified.
```

MISC

```
    -regress
          pretend this is a regression and run the job in DRMJOBSCRATCHSPACE
          instead of the launch directory. useful with -indrm and
          -interactive options and single jobs. automatically added for
          regressions.

    -nobuild
          this is a master switch to disable all building options.
          there is no such thing as -build to enable all build options.
    -copyall/-nocopyall
         copy back all files to launch directory after passing regression run.
         Normally, only failing runs cause a copy back of files.
         Default is off.

-copyall/-nocopyall
          copy back all files to launch directory after passing
         regression run.  Normally, only failing runs cause a
         copy back of files.
         Default is off.

    -copydump/-nocopydump
          copy back dump file to launch directory after passing
         regression run.  Normally, only failing runs cause a copy
         back of non-log files.  The file copied back is vcs.fsdb,
         or vcs.vcd if -fsdb2vcd option is set.
         Default is off.

-tarcopy/-notarcopy
          copy back files using 'tar'. This only works in copyall or
          in the case the simulations 'fails' (per sims' determination).
          Default is to use 'cp'.

    -diag_pl_args=ARGS
         If the assembly diag has a Perl portion at the end, it
         is put into diag.pl and is run as a Perl script.
         This allows you to give arguments to that Perl script.
          The arguments accumulate, if the option is used multiple
          times.

    -pal_use_tgseed
           Send '-seed=<tg_seed_value> to pal diags.  Adds
           -pal_diag_args=-seed=tg_seed to midas command line, and
           -seed=tg_seed to pal options (vrpal diags). Use -tg_seed to set
           the value passed to midas or use a random value from /dev/random.

    -parallel
            when specifying multiple groups for regressions this switch will
            submit each group to DReAM to be executed as a separate regression.
            this has the effect of speeding up regression submissions.
            NOTE: This switch must not be used with -indrm

    -reg_count=COUNT
            runs the specified group multiple times in regression mode. this
            is useful when we want to run the same diag multiple times using
            a different random generator seed each time or some such.
```

```
-regress_id=ID
        specify the name of the regression

-report
        This flag is used to produce a report of a an old or running
        regression. With -group options, sims produces the report
        after the regression run. Report for the previous
        regression run can be produced using -regress_id=ID
        option along with this option,

-finish_mask=MASK
       masks for vcs simulation termination. Simulation terminates
       when it hits 'good_trap' or 'bad_trap'. For multithread
       simulation, simulation terminates when any of the thread
       hits bad_trap, or all the threads specified by the finish_mask
       hits the good_trap.
       example: -finish_mask=0xe
       Simulation will be terminated by good_trap, if thread 1, 2 and
       3 hits the good_trap.

-stub_mask=MASK
       mask for vcs simulation termination. Simulation ends when the
       stub driving the relevant bit in the mask is asserted. This
       is a hexadecimal value similar to -finish_mask

-wait_cycle_to_kill=VAL
       passes a +wait_cycle_to_kill to the simv run. a testbench
       may chose to implement this plusarg to delay killing a
       simulation by a number of clock cycles to allow collection
       of some more data before exiting (e.g. waveform).

-rtl_timeout
       passes a +TIMEOUT to the simv run.
       sets the number of clock cycles after all threads have become
       inactive for the diag to exit with an error. if all threads hit
       good trap on their own the diag exits right away. if any of the
       threads is inactive without hitting good trap/bad trap the
       rtl_timeout will be reached and the diag fails. default is 1000.
       this is only implemented in the cmp based testbenches.

-max_cycle
       passes a +max_cycle to the simv run.
       sets the maximum number of clock cycle that the diag will take
       to complete. the default is 30000. if max_cycle is hit the diag
       exits with a failure. not all testbenches implement this
       feature.
```

```
    -norun_diag_pl
         Does not run diag.pl (if it exists) after simv (vcs) run.
         Use this option if, for some reason, you want to run an existing assembly
         diag without the Perl part that is in the original diag.

    -nosaslog
           turns off redirection of sas stdout to the sas.log file.
           use this option when doing interactive runs with sas.

    -nosimslog
           turns off redirection of stdout and stderr to the sims.log
           file.  use this option in conjunction with -vcs_use_cli or
           -vcs_use_ucli to get to the cli prompt when using vcs or to
           see a truncated vcs.log file that exited with an
           error. this must be used if you want control-c to work
           while vcs is running.

    -nogzip
           turns off compression of log files before they are copied over
           during regressions.

    -version
            print version number.

    -help
            prints this
```

IT SYSTEM RELATED

```
    -use_iver=FILE
            full path to iver file for frozen tools

    -use_sims_iver/-nouse_sims_iver
            For reruns of regression tests only, use sims.iver to choose
            TRE tool versions saved during original regression run.
            Defaults to true.

    -use_cdms_iver/-nouse_cdms_iver
            Uses the frozen iver file located under DV_ROOT if present.
            This defaults to true. This has no effect if an iver file
            is not found under the cdms tree.

    -dv_root=PATH
            absolute path to design root directory. this overrides DV_ROOT.

    -model_dir=PATH
            absolute path to model root directory. this overrides MODEL_DIR.
```

```
-tmp_dir=PATH
        path where temporary files such as debussy dumps will be created

-sims_config=FILE
        full path to sims config file

-sims_env=ENVAR=value
        force sims to set ENVAR variable to specified value.

-env_base=PATH
        this specifies the root directory for the bench environment.
        it is typically defined in the bench config file. It has no
        default.

-config_cpp_args=OPTION
        this allows the user to provide CPP arguments (defines/undefines)
        that will be used when the testbench configuration file is
        processed through cpp. Multiple options are concatenated
        together.

-result_dir=PATH
        this allows the regression run to be launched from a different
        directory than the one sims was launced from. defaults to
        PWD.

-diaglist=FILE
        full path to diaglist file

-diaglist_cpp_args=OPTION
        this allows the user to provide CPP arguments (defines/undefines)
        that will be used when the diaglist file is processed through
        cpp. Multiple options are concatenated together.

-asm_diag_name=NAME
-tpt_diag_name=NAME
-tap_diag_name=NAME
-vera_diag_name=NAME
-vera_config_name=NAME
-efuse_image_name=NAME
-image_diag_name=NAME
-sjm_diag_name=NAME
-pci_diag_name=NAME

        name of the diagnostic to be run.

-asm_diag_root=PATH
-tpt_diag_root=PATH
-tap_diag_root=PATH
-vera_diag_root=PATH
```

```
-vera_config_root=PATH
-efuse_image_root=PATH
-image_diag_root=PATH
-sjm_diag_root=PATH
-pci_diag_root=PATH

        absolute path to diag root directory. sims will perform a find
        from here to find the specified type of diag. if more than one
        instance of the diag name is found under root sims exits with
        an error. this option can be specified multiple times to allow
        multiple roots to be searched for the diag.

-asm_diag_path=PATH
-tpt_diag_path=PATH
-tap_diag_path=PATH
-vera_diag_path=PATH
-vera_config_path=PATH
-efuse_image_path=PATH
-image_diag_path=PATH
-sjm_diag_path=PATH
-pci_diag_path=PATH

        absolute path to diag directory. sims expects the specified
        diag to be in this directory. the last value of this option
        is the one used as the path.
```

ClearCase

```
-clearcase
    assume we are in ClearCase environment for setting DV_ROOT and
    launching DReAM commands. default is off (CDMS++ version control)

-noclearcase
    force clearcase option off

-cc_dv_root=PATH
    ClearCase path to design root directory. this overrides CC_DV_ROOT.
```

ENV VARIABLES

```
sims sets the following ENV variables that may be used with pre/post
processing scripts, and other internal tools:
```

TABLE A-1 Enviromental Variable

Environment Variable	Description
ASM_DIAG_NAME	Contains the assembly diag name.
SIMS_LAUNCH_DIR	Path to launch directory where sims is running the job. Useful when job is run in dream scratch space.
VERA_LIBDIR	Dir where Vera/NTB files are compiled or results are stored.
DV_ROOT	-dv_root if specified
MODEL_DIR	-model_dir if specified
TRE_SEARCH	Based on -use_iver, -use_cdms_iver -use_sims_iver
DENALI	Based on configsrch
VCS_HOME	Based on configsrch

PLUSARGS

```
+args are not implemented in sims. they are passed directly to vcs at
compile time and simv at runtime. the plusargs listed here are for
reference purposes only.

    +STACK_DIMM 32 bits physical address space - default is 31 bits

    +STACK_DIMM +RANK_DIMM 33 bits physical address space - default is 31 bits

    +max_cycle see -max_cycle

    +TIMEOUT   see -rtl_timeout

    +vcs+finish see -vcs_finish

    +wait_cycle_to_kill see -wait_cycle_to_kill
```

DESCRIPTION

```
sims is the frontend for vcs to run single simulations and regressions
```

How To Build models

```
Build a model using DV_ROOT as design root
```

```
  sims -sys=cmp -vcs_build
```

Build the vera testbench only using DV_ROOT as design root

```
  sims -sys=cmp -vera_build
```

Build a model from any design root

```
  sims -sys=cmp -vcs_build -dv_root=/home/regress/2002_06_03
```

Build a graft model from any design root

```
  sims -sys=cmp -vcs_build -dv_root=/model/2002_06_03
       -graft_flist=/regress/graftfile
```

Build a model and re-build the vera

```
  sims -sys=cmp -vcs_build -vera_clean
```

Build a model and turn off incremental compile

```
  sims -sys=cmp -vcs_build -vcs_clean
```

Build a model with a given name

```
  sims -sys=cmp -vcs_build -vcs_rel_name=mymodel
```

How To Run Models

Run a diag with default model

```
  sims -sys=cmp -vcs_run diag.s
```

Run a diag with a specified model

```
  sims -sys=cmp -vcs_rel_name=mymodel -vcs_run diag.s
```

Run a diag with debussy dump with default model

```
  sims -sys=cmp -debussy -vcs_run diag.s <dump scope args>
```

Run a diag using arguments form specified alias in a diaglist

```
  sims -vcs_run -sys=spc2 -group isa_mt -alias=isa_mmu_21:isa_mt isa_mmu_21.s
```

```
Run regressions

Run a regression using DV_ROOT as design root

  sims -group=mini

Run a regression using DV_ROOT as design root and specify the diaglist

  sims -group=mini -diaglist=/home/user/my_dialist

Run a regression using any design root

  sims -group=mini -dv_root=/afara/design/regress/model/2002_06_03

Run a regression using any design root and a graft model

  sims -group=mini -dv_root=/regress/model/2002_06_03
       -graft_flist=/home/regress/graftfile

Rerun a diag in a regression (in new rerun_x subdir)

  sims -rerun

Rerun a diag, overwriting same directory

  sims -rerun -overwrite
```

A.2 midas help

```
NAME
     midas - assembles diags (Midas Is a Diag ASsembler)

SYNOPSIS
     midas [options] <diag_name>

DESCRIPTION
     This program builds assembly diags.  It is substantially
     more involved than simply assembling the diag because it
     also has to link the diag, program the MMU, and generate
     several output files.

     The diag specified on the command line will be built.
     Pretty much everything else is configurable.
```

Options

The following are the options you need to get started:

```
-h  Display man page.

-verbose [level] / -noverbose (abbreviated -v / -nov)
    Sets verbosity level (default=2).  -noverbose (or -nov)
    is a synonym for -verbose 0, which means to generate no
    output in the absence of errors.  The highest level of
    verbosity currently defined is 3.

-version
    Return version information and exit.

-format
    Display help on the diag format and exit.

-config <file>
    Use this file as the config file instead of the one that
    is distributed with Midas.

-project <project>
    Use this project for project-specific configuration.
    Default is the environment variable $PROJECT.  Legal
    values are BW and N2.
```

Common Options

The following are the commonly-used options:

```
-diag_root <path>
    Use the specified path as a base for finding standard
    include files.  Default is $DV_ROOT.

-build_dir <path>
    Path (absolute or relative to where command is invoked)
    to directory where temporary files are generated and the
    build is done.  Default is './build'.

-dest_dir <path>
    Path (absolute or relative to where command is invoked)
    of where to store output files.  Default is '.'.

-find_root <dir>
    Interpret the diag on the command-line as the name of a
    diag to search for.  It does a breadth-first search
    under the specified directory.  The default behavior is
    not to do any search, but to assume that the specified
```

```
        diag is a full or relative path to the file.

    -find
        This is a shortcut for "-find_root
        <diag_root>/verif/diag".

    -mmu <mmu_type>
        Generate programming for the specified MMU.  Recognized
        options are "ultra2", "niagara", and "niagara2". Default
        is project-specific: "niagara" for Niagara-1 and
        "niagara2" for Niagara-2.

    -ttefmt <tte_format>
        Specifies TTE format for those MMUs that require it.
        May be "sun4u" or "sun4v".  Default is project-specific:
        "sun4v" for Niagara-1 and Niagara-2.

    -tsbtagfmt <tsbtagfmt>
        Specifies the format of the TSB tag.  Legal values are
        'tagaccess' and 'tagtarget'.  Default is
        project-specific: 'tagaccess' for Niagara-1 and
        'tagtarget' for Niagara-2.

    -force_build or -f
        Build the diag, even if it looks like we have the same
        input as before and the same args as before.

    -copy_products / -nocopy_products
        By default, the product files generated in the build
        directory are hard-linked to the destination directory.
        The reason they are hard-linked and not copied is for
        speed.  If the hard link fails, it will fall back to a
        copy in case the directories are on different physical
        disks.  If -copy_products is given, however, it will
        always do a copy, not a hard link.  Default is
        project-specific:  -nocopy_products for Niagara-1.

    -E  Stop after the preprocessing stage.

    -addphdr / -noaddphdr
        If -addphdr is enabled and the project env variable is
        N2, Midas will add PHDR commands into the diag.ld_scr (
        linker script file ). This option is currently by
        default disabled. N2 needs this option to optimize the
        size of the diag.exe file.

    -cleanup / -nocleanup
        If -cleanup is enabled, then after a successful build,
        the build directory is erased if and only if the build
        directory was created by this invocation of midas.
        Default is project-specific: -cleanup for Niagara-1.

    -force_cleanup / -noforce_cleanup
        If -cleanup is enabled, but this invocation of midas did
        not create the build directory, -force_cleanup will
        remove the build directory anyway.  Default is
        project-specific: -noforce_cleanup for Niagara-1.

    -D<symbol> or -D<symbol>=<value>
        Add a define to the preprocessing line.  Option may be
        repeated.

    -stddef / -nostddef
        Include standard preprocessor definitions on
        command-line.  -nostddef disables these.  Default is
        -stddef, but no standard symbols are currently defined.

    -I<dir>
        Add a directory to the include path used by cpp and m4.
        Path should be absolute or relative to the directory
        where midas was invoked.  Option may be repeated.

    -stdinc / -nostdinc
        With -stdinc, the standard include paths are used during
        preprocessing (both cpp and m4).  -nostdinc disables
        these.  Default is -stdinc.  The standard include
        directories are the directory where midas was invoked,
        the build directory and
        <diag_root>/verif/diag/assembly/include (keep in mind
        that <diag_root> defaults to $DV_ROOT).

    -include_build / -noinclude_build
        This option is only meaningful with -nostdinc.  If
        standard includes are switched off, -include_build will
        add the build directory back to the include path.
        Default is -noinclude_build.

    -include_start / -noinclude_start
        This option is only meaningful with -nostdinc.  If
        standard includes are switched off, -include_start will
        add the start directory (the directory where midas was
        invoked) back to the include path.  Default is
        -noinclude_start.

    -L<dir>
        Add a directory to the search path when looking for
        object files in a MIDAS_OBJ directive.  Option may be
        repeated.
```

```
-C<dir>
    Add a directory to the search path when looking for C
    source files in a MIDAS_CC directive.  Option may be
    repeated.

-pal_diag_args <args>
    If the diag is run through pal, give these arguments to
    the pal diag.  Option may be repeated.  Note that these
    arguements are given to the diag, not pal itself.  For
    instance, "midas -pal_args -abc mydiag.pal
    -pal_diag_args def -pal_diag_args ghi" will run the pal
    command-line "pal -abc mydiag.pal def ghi".

-build_threads <num_threads>
    When doing work that can be done in parallel (such as
    assembling a bunch of files), use <num_threads> to do
    it.  Default is project-specific: 3 for Niagara-1.

-print_errors / -noprint_errors
    If -noprint_errors is defined, then generation of error
    messages is turned off.  When used with -verbose 0,
    midas is completly silent.  This is probalby only useful
    for the test harness (which is why the switch is there).

-copy_products / -nocopy_products
    If this is set, then copy files from the build directory
    to the starting directory.  With -nocopy_products, the
    files are hard linked instead.  If it tries to create a
    hard link and fails, it will fall back to a copy.
    Default is -nocopy_products.

-compress_image / -nocompress_image
    If -compress_image is enabled (as it is by default),
    then allow compressed mem.images to be generated.  By
    default, all MMU-generated blocks are compressed when
    written to mem.image, meaning that instead of
    initializing unused sections to zero, they are simply
    uninitialized.  The -nocompress_image is equivalent to
    explicitly putting a 'compressimage=0' in all
    attr_text/attr_data blocks.

-env_zero / -noenv_zero
    When compressing blocks, if -env_zero is enabled the
    blocks will contain '// zero_image' directives to the
    environment.  These directives are supported only by
    Niagara, and they are used to backdoor initialize large
    tracts of memory to zero.  If -noenv_zero is used, then
    compression will simply leave the data uninitialized.
```

```
-default_radix <decimal|hex>
    Radix to assume for all parameters that do not
    explicitly start with '0x'.  Default is 'decimal'.

-gen_all_tsbs / -nogen_all_tsbs
    If -gen_all_tsbs is given, then all TSBs that are
    defined are written to the memory image.  If
    -nogen_all_tsbs, then generate only the TSBs that are
    used.  Default is project-specific: -nogen_all_tsbs for
    Niagara-1.

-allow_tsb_conflicts / -noallow_tsb_conflicts
    If -allow_tsb_conflicts is enabled, then it is legal to
    have mutiple virtual address map to the same entry in a
    TSB.  A linked-list will be created to hold all entries.
    With -noallow_tsb_conflicts (which is the default for
    N1), collisions in the TSB can only happen with the save
    VA but different contexts.  Default is project-specific.

-allow_empty_sections / -noallow_empty_sections
    If TEXT_VA is specified, then at least one attr_text
    block for the section has to be specified, and the same
    is true for DATA_VA and attr_data blocks.  If
    -allow_empty_sections is specified, then midas will
    allow you to specify a TEXT_VA(DATA_VA) for the section,
    even if the section has no attr_text(attr_data) blocks.
    Of course, any text(data) in such a section will be
    ignored.  Default is project-specific:
    -noallow_empty_sections for Niagara-1.

-allow_duplicate_tags / -noallow_duplicate_tags
    When adding to a TSB link list, it is an error to add
    the same tag twice.  -allow_duplicate_tags suspends the
    error check.  Default is project-specific:
    -noallow_duplicate_tags for Niagara-1.

-allow_illegal_page_sizes / -noallow_illegal_page_sizes
    If -allow_illegal_page_sizes, then tte_size attributes
    are not checked for valid values, though they are still
    checked against the width of the field.  For instance,
    in the Niagara MMU, there are 3 page bits, so values can
    be specified 0-7.  However, the only legal values for
    Niagara are 0, 1, 3, and 5, and unless
    -allow_illegal_page_sizes is in effect, setting page
    bits of 2, 4, 6, or 7 will cause an error.  The default
    is project-specific: -noallow_illegal_page_sizes for
    Niagara-1.
```

```
-allow_misalgined_tsb_base / -noallow_misaligned_tsb_base
    If -allow_misaligned_tsb_base is set, then a TSB base
    address need not be aligned with the TSB size.  Real
    software will never do this, but I want it to be
    possible in diags.  If an unalgined address is specified
    as the base and -allow_misaligned_tsb_base is specified,
    then midas will forcibly align the address.  Default
    should be -noallow_misaligned_tsb_base for all projects.

-errcode <error_code>
    Prints a one-line description for the midas error code.
    Then exits with status 0.
```

Configuring Commands

```
midas runs several commands in the course of its operation.
Several of these can be configured.  The configurable
commands are: pal, cpp, m4, gcc, as, and ld.  Each
configurable command has 3 associated options:

-std_<command>_args / -nostd_<command>_args
    When -std_<command>_args is enabled, the standard set of
    arguments for <command> are used.  Default is
    -std_<command>_args

-<command>_args <args>
    Add <args> to the argument list for the specified
    <command>.

-<command>_cmd <custom_command>
    Use <custom_command> to run the specifed <command>
    instead of the standard version.
```

Example

```
For instance, to add -foo to the link line, use my_cpp to
preprocess, and not use any standard assembler options, use:

  midas -ld_args -foo -cpp_cmd my_cpp -nostd_as_args mydiag.s
```

Configuring Filenames

```
There are several generated files, and they all have default
names.  You can configure the names of many of the files
with the following option.

-file <tag>=<name>
    Cause midas to name the file whose tag is <tag> to be
    named <name> instead of the default.  <name> is treated
    as the name of a file in the build directory.

The list of valid tags for the -file option are:

src Local version of the original source code for the diag.
    Default is 'diag.src'.

s   Assembly portion of diag before any preprocessing.
    Default is 'diag.s'.

pl  Perl portion of the diag.  Deafult is 'diag.pl'.

cpp Output of the C preprocessor.  Deafult is 'diag.cpp'.

m4  Output of the m4 preprocessor.  Default is 'diag.m4'.

ldscr Linker script.  Default is 'diag.ls_scr'.

exe Linked executable.  Default is 'diag*.exe' where * is
    application name.

image Verilog memory image.  Default is 'mem.image'.

events Events file.  Default is 'diag.ev'.

symtab Symbol table.  Default is 'symbol.tbl'.

goldfinger
    Specification to goldfinger on how to create memory
    image.  Default is 'diag.goldfinger'.

directives
    File to contain midas directives after section
    splitting.  Default is 'diag.midas'.

cmdfile
    File to stash the midas command-line.  Default is
    '.midas_args'.

oldcmdfile
    File to move old command-line options.  Default is
    '.midas_args.old'.

oldm4
    File to stash m4 output of previous run.  Default is
    '.midas.diag.m4.old'.
```

```
Running Specific Phases

The build process is broken into phases: setup, preprocess,
sectioning, assemble, link, postprocess, copydest, cleanup.
The default behavior is to run all phases.  You can,
however, restrict operation to a selected set of phases.

-start_phase <phase_name>
    Start with the named phase and run all subsequent phase.

-phase <phase_name>
    Run the specified phase.  If any -phase or -start_phase
    option exists, then by default all phases are off
    (except for the ones that -phase and -start_phase switch
    on).  You can have multiple -phase options.

-E  This option (mentioned above, which runs the
    preprocessor only) is just a shortcut for "-phase setup
    -phase preprocess").

Keep in mind that running selected phases is caveat emptor.
There are cases where phases expect data or files from
previous phases.  You may get lucky, but don't blame me if
it doesn't work.

Errors

When midas is unable to run correctly it will exit with one
of the folllowing error codes.

M_NOERROR (#0): No error.
M_MISC (#1): Miscellaneous error
M_CODE (#2): Error in midas code.
M_DIR (#3): Directory error.
M_FILE (#4): File error.
M_CMDFAIL (#5): Command failed.
M_SECSYNTAX (#6): Error in section syntax.
M_ATTRSYNTAX (#7): Error in attr syntax.
M_MISSINGPARAM (#8): Missing parameter.
M_ILLEGALPARAM (#9): Illegal parameter.
M_OUTOFRANGE (#10): Out of range.
M_NOTNUM (#11): Not a number.
M_VACOLLIDE (#12): VA collision.
M_PACOLLIDE (#13): PA collision.
M_DIRECTIVESYNTAX (#14): Directive syntax error.
M_GENFAIL (#15): File generation failed.
M_ASMFAIL (#16): Assembler failed.
M_CCFAIL (#17): C compiler failed.
M_LINKFAIL (#18): Linker failed.
```

```
    M_CPPFAIL (#19): CPP failed.
    M_M4FAIL (#20): M4 preprocessor failed.
    M_BADCONFIG (#21): Bad configuration.
    M_EVENTERR (#22): Event parsing error.
    M_ARGERR (#23): Argument error.
    M_NOSEC (#24): Undefined section.
    M_BADTSB (#25): Bad TSB.
    M_BADALIGN (#26): Bad Alignment.
    M_EMPTYSECTION (#27): Empty section.
    M_TSBSYNTAX (#28): Error in tsb syntax.
    M_APPSYNTAX (#29): Error in app syntax.'
    M_MEMORY (#30): Memory error.
    M_GOLDFINGERPARSE (#31): Goldfinger parse error.
    M_GOLDFINGERARG (#32): Goldfinger arg error.
    M_ELF (#33): ELF error.
    M_BADLABEL (#34): Bad label.
    M_GOLDFINGERMISC (#35): Uncategorized goldfinger error.
    M_GOLDFINGERVERSION (#36): Bad version of goldfinger
    M_DUPLICATETAG (#37): Duplicate tags in TSB
    M_BLOCKSYNTAX (#38): Error defining goldfinger BLOCK
```

A.3 regreport

```
DESCRIPTION

regreport examines all regression *.log files for diags under regression directory
and prints report. It is called by sims for each diag. User typically calls
regreport to generate summary of regression.

Usage: regreport <options> [<directory> [<list>]]

OPTIONS
  -1 [<regress_dir>]:
     print report for the specified or current-directory diag; [regress dir].

  -regress <output_file> <directory>:
     in regression mode, regreport writes summary status for finished
     diags to a file until all diags are finished.
     NOTE: if some diag does not produce status, regreport will wait forever.

  -ver
     print version number and exit.

  -sas_only
     vcs.log will not bw scanned, sas.log only.
  -[no]cut_name
     cuts the name from  a sss:sss:sss:ddd formatted name. Default is to cut.

  -regenerate
     will regenerate the status.log files in the diag directories.

  -clean_pass
     will clean up passing directories.

  -fails_only
     will show details for fails only

  <directory> [<list>]
     print report for all diags under <directory>. <list> is
     0 or more of simulation 'system' names, such as
     'spc2', 'cmp', 'cmp1', 'cmp8', etc. When nothing
     specified, all systems are included.
```

ENVIRONMENT VARIABLES:

```
  CLEAN_PASS : Clean passing dirs
  REGRESS_MAIL : Set to comma seperated list. Default is to send user
                 email when run in regress mode.  When set to "no"
                 sends no email at all.
  REGREPORT_FAILS_ONLY : Show details for fails only.
```