
High Density 65nm CLN65GP ROM Compiler User Guide

Revision: r0p0

Confidential

ARM[®]

**May 2010
Copyright 2010 ARM. All rights reserved.
ARM PUG 0099A**

Copyright © 2010 ARM. All rights reserved.

ARM PUG 0099A

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Table of Contents

Preface

Revision History	vii
Customer Support	vii
Typographic Conventions	viii

Chapter 1

Overview and Installation	1-1
1.1 Overview	1-3
1.2 High Density ROM Compiler Features	1-4
1.3 Compiler Installation	1-6
1.3.1 System Requirements	1-6
1.3.1.1 Operating System Requirements	1-6
1.3.1.2 Disk Space Requirements	1-7
1.3.2 Installing the Compiler	1-7
1.3.2.1 Installation Tasks	1-7
1.3.2.2 Directory Structure and Executables	1-8

Chapter 2

Using the Compiler	2-1
2.1 Overview	2-3
2.1.1 Running the Compiler from the Graphical User Interface (GUI).....	2-3
2.1.2 Running the Compiler from the Command Line	2-4
2.2 Views and Output Files	2-5
2.3 GUI Components	2-7
2.3.1 Generic Parameters Pane	2-8
2.3.2 Views Pane	2-8
2.3.3 Relative Footprint Pane	2-9
2.3.4 ASCII Datatable Pane	2-9

2.3.5 Message Pane	2-12
2.3.6 File Menu (Exiting the GUI)	2-12
2.3.7 Utilities Menu	2-12
2.3.8 Help Menu	2-13
2.3.9 Balloon Help	2-13
2.4 Generating Views from the GUI	2-14
2.4.1 Generating Single Views	2-14
2.4.2 Generating Multiple Views.....	2-15
2.4.3 Setting View-Specific Parameters	2-16
2.5 Generating Views from the Command Line	2-17
2.5.1 View Commands.....	2-17
2.5.2 Generating Multiple Views with View-Specific Options.....	2-18
2.6 Generating Specification and Log Files	2-19
2.6.1 Using Specification Files	2-19
2.6.2 Creating Log Files	2-20
2.6.3 Generating Parameter Information	2-21
2.7 Compiler Options	2-22
2.7.1 Command Line Syntax	2-22
2.7.2 Basic Options	2-22
2.7.3 Setting Advanced Options	2-26
2.7.3.1 Setting Advanced Options from the GUI	2-26
2.7.3.2 Setting Advanced Options from the Command Line.....	2-27
2.7.4 Advanced Options.....	2-27
2.7.4.1 Advanced View-Specific Options	2-29

Chapter 3

Synchronous ROM Compiler Architecture	3-1
3.1 Overview	3-3
3.2 Synchronous ROM Architecture and Timing Specifications (rom_via_hde_hvt_rvt_hvt)	3-4
3.2.1 ROM Description (rom_via_hde_hvt_rvt_hvt)	3-4
3.2.1.1 Basic Functionality	3-4
3.2.1.2 Test Functionality	3-5

3.2.2 ROM Pins (rom_via_hde_hvt_rvt_hvt)	3-8
3.2.3 ROM Logic Tables (rom_via_hde_hvt_rvt_hvt)	3-10
3.2.4 ROM Parameters (rom_via_hde_hvt_rvt_hvt)	3-11
3.2.5 ROM Block and Core Address Diagrams (rom_via_hde_hvt_rvt_hvt)	3-12
3.2.5.1 ROM Core Address Maps (rom_via_hde_hvt_rvt_hvt)	3-14
3.2.6 ROM Timing Specifications (rom_via_hde_hvt_rvt_hvt)	3-18
3.2.6.1 ROM Timing Diagram (rom_via_hde_hvt_rvt_hvt)	3-18
3.2.6.2 ROM Timing Parameters (rom_via_hde_hvt_rvt_hvt)	3-19
3.2.6.3 ROM Power Parameters (rom_via_hde_hvt_rvt_hvt)	3-21
3.3 ROM Power Structure (rom_via_hde_hvt_rvt_hvt)	3-22
3.3.1 ROM Current Parameters	3-22
3.3.2 Power Distribution Methodology	3-23
3.3.2.1 Noise Limits	3-25
3.3.2.2 ROM Code File	3-25
3.4 ArtiGrid Power Structure Options (rom_via_hde_hvt_rvt_hvt)	3-27
3.4.1 Power Supply Configurations	3-27
3.4.1.1 Standard Option	3-27
3.4.1.2 Power Gating Option	3-29
3.5 ROM Physical Characteristics (rom_via_hde_hvt_rvt_hvt)	3-30
3.5.1 Top Metal Layer	3-30
3.5.2 I/O Connections	3-30
3.5.3 Characterization Environments	3-31
3.6 ROM Timing Derating (rom_via_hde_hvt_rvt_hvt)	3-32

Chapter 4

EDA Tools Support	4-1
4.1 Overview	4-3
4.2 EDA Tool Support	4-3
4.3 Using the Compiler Views	4-4
4.3.1 Using the Verilog Model	4-4
4.3.2 Using the Synopsys (Liberty) Model to Generate SDF	4-5
4.3.3 Using the Synopsys (Liberty) Model to Generate SDF with Cadence Encounter	4-6

4.3.4 Using the RTL Compiler for Timing Analysis.....	4-7
4.3.5 Loading the VCLEF Description into SOC Encounter	4-9
4.3.6 Using Astro with ARM Memory Instances	4-10
4.3.6.1 Loading the VCLEF Description into the Milkyway Database.....	4-10
4.3.6.2 Loading the GDSII Layout into the Milkyway Database	4-11
4.3.7 Loading the GDSII Layout into a DFII Library	4-11
4.3.8 Using the LVS Netlist.....	4-12
4.3.8.1 Using Hierarchical LVS.....	4-12
4.3.9 Using the CeltIC Enablement Tool.....	4-14
4.3.9.1 Design Inputs	4-14
4.3.9.2 Required Scripts and Files	4-15
4.3.9.3 Writing the TCL Script for CeltIC.....	4-17
4.3.9.4 Generating CeltIC Model (CDB).....	4-18
4.3.9.5 Validation.....	4-18
4.3.10 Using VoltageStorm Flow	4-20
4.3.10.1 Compiler Generated Files	4-20
4.3.10.2 Foundry Supplied Files	4-20
4.3.10.3 ARM Supplied Files	4-20
4.3.10.4 Libgen Command File	4-21
4.3.11 VoltageStorm Flow Setup and Run	4-21
4.3.11.1 Directory Files	4-21
4.3.11.2 Flow Run Tools	4-22
4.3.11.3 Running the Flow.....	4-22
4.3.11.4 Command Files	4-22
4.3.11.5 Verification	4-24
 Chapter Appendix A- Revisions	 5-1

Preface

Revision History

The following table provides the revision history for this manual.

Part Number	Date	Updates
ARM PUG 0099A	6 May 2010	First release for r0p0

Customer Support

Customers with active Support contracts can obtain support for ARM Physical IP products by going to access.arm.com and clicking on the “Products & Services > New Technical Request.” link on the left side of the webpage. ARM recommends using this method for customers with valid support contracts, in order to obtain prompt attention to issues and questions.

Information about available Support contract options can be seen at www.arm.com/products/physicalip/support.html.

If you cannot reach us via the web support channel you can contact ARM Physical IP support via email at support-pipd@arm.com for technical issues.

Typographic Conventions

The following typographic conventions are used to assist you in distinguishing special notations, values, and elements described in this manual.

Visual Cue	Meaning
(Bullet) •	Bulleted list of important items.
Courier Type	Commands typed on the keyboard, either examples or instructions.
Dash (-) Courier Type	Text set in Courier type and preceded by a dash represents a command name (for example, -libname).
< <i>italic type</i> > <i>italic type</i>	Variable names you select, such as file and directory names are enclosed within angle brackets (< >). Italic type is used to show variable values, file, and directory names.
(Ellipsis) ...	Indicates commands or options that may be added.
<i>Italic Type with Initial Capital Letters</i>	Document, chapter, section and reference manual names.

1

Overview and Installation

This chapter contains the following sections:

- “Overview” on page 1-3
- “High Density ROM Compiler Features” on page 1-4
- “Compiler Installation” on page 1-6

1.1 Overview

ARM designs the technology that lies at the heart of advanced digital products, from wireless, networking and consumer entertainment solutions to imaging, automotive, security and storage devices. ARM's comprehensive product offering includes 16/32-bit RISC microprocessors, data engines, 3D processors, digital libraries, embedded memories, peripherals, software and development tools, as well as analog functions and high-speed connectivity products. Combined with the company's broad Partner community, they provide a total system solution that offers a fast, reliable path to market for leading electronics companies.

This manual provides information on using high density ROM compilers to create instances with a variety of parameters and views. This manual provides information about diffusion and via ROM compilers.

Note

Parameters or specifications for your compiler may differ from the default compilers. Information about deviations to the compilers can be found in the README text file that is enclosed with your compiler or in an addendum attached to this manual.

Refer to the following sections for more detailed information about this compiler.

- This chapter provides basic information about high density ROM compilers, such as features and views, plus important information about installation requirements and tasks.
- Chapter 2, “Using the Compiler,” - Provides details about using the compiler GUI or the command line to generate views and instances.
- Chapter 3, “Synchronous ROM Compiler Architecture,” - Lists the architectural details, physical characteristics of memory instances, and characterization/timing information.
- Chapter 4, “EDA Tools Support,” - Lists the tool versions supported by the compiler and provides instructions on generating specific views.

1.2 High Density ROM Compiler Features

High density ROM compilers include the following features:

- Optimized for Low Power/High Density
- Aspect Ratio Control for Efficient Floor Planning
- Memory Operation at Low Frequency
- Low Active Power and Leakage-Only Standby Power
- Timing and Power Models for Industry-Leading Design Tools
- Extra Margin Adjustment™ (EMA) Option
- Weak Bit Test (WBT)
- Integrated BIST Mux Option
- ArtiGrid™/Over-the-Cell (OTC)
- Max Static Power Corner
- Back Biasing Support
- Power Gating

A standard set of EDA support views can be generated from high density ROM compilers. These views are verified with the tools defined in the applicable EDA package; EDA tools are detailed in the README file. Refer to Chapter 4, “EDA Tools Support,” for details about using the views. Optional support is available and can be added to most existing compilers without installing a completely new compiler.

Deliverable and file names are listed below; not all items may be applicable to your compiler.

Deliverables and provided file names

Verilog

Synopsys Liberty - includes CCS-noise

VCLEF Footprint

GDSII Layout

LVS netlist

Fastscan

TetraMAX

CDB Enablement

VoltageStorm Enablement

PostScript Datasheet

ASCII Datatable

1.3 Compiler Installation

This section provides information about system requirements, installation tasks, directory structure, and compiler terminology.

1.3.1 System Requirements

Make sure that your operating system and disk (CPU) space allocation meet compiler requirements to ensure proper functioning of the compiler, as described in the following sections.

1.3.1.1 Operating System Requirements

The EDA package is supported by the Redhat Enterprise 3.0 (RHEL3) LINUX operating system.

If your operating system does not meet RHEL3 LINUX requirements, you may receive an error message when running the memory compiler. In this case, the compiler will not function until you upgrade your operating system.

To determine the name and version of your operating system, enter the following command:

```
uname -a
```

1.3.1.2 Disk Space Requirements

Make sure you have enough disk space available for your installation. There are different space requirements for the various stages you must complete before you can use the compiler.

A compiler requires approximately 500 megabytes when you copy it to the installation directory. When you uncompress the compiler file approximately 1 gigabyte is required. When you extract (tar) the compiler file, the original file and the uncompressed/extracted version are in the installation directory and need approximately 1.5 gigabytes of disk space.

1.3.2 Installing the Compiler

You must determine where you want to install the compiler on your system. In this manual, `<install_dir>` refers to the directory you choose for installation. You may also create a working directory, `<working_dir>`, where you actually run the compiler to create memory instances.

————— **Note** —————

When copying the installation files, you should create a new directory. Overwriting an existing compiler directory may corrupt the compiler installation.

1.3.2.1 Installation Tasks

Change to the installation directory:

```
cd <install_dir>
```

Uncompress and extract the installation files:

```
gtar -xvzf <install_files>.tgz
```

1.3.2.2 Directory Structure and Executables

Installing the compiler produces the following directory structure.

aci/<executable>/*

- bin/ This directory contains the compiler executable and platform-specific directories.
- lib/ This directory contains technology files, library files, executables, and subdirectories.
- doc/ This directory contains compiler documentation.
- corner/ For some compilers, this directory contains compiler timing data.

where <executable> refers to the name of the file that is run to generate an instance.

——— **Note** ———

ARM's Artisan Compiler GUI is written in Java. For your convenience, the compiler source tree includes copies of all required JRE distributions.

The following table provides the general names and executables for available memory compilers. Check your compiler GUI; the names and executables provided in your compiler GUI always supercede those in the table below.

Table 1-1. ROM Compiler Naming Conventions

Compiler	Product Name	Executable
High density Via ROM	rom_via_hde_hvt_rvt_hvt	rom_via_hde_hvt_rvt_hvt

2

Using the Compiler

This chapter contains the following sections:

- “Overview” on page 2-3
- “Views and Output Files” on page 2-5
- “GUI Components” on page 2-7
- “Generating Views from the GUI” on page 2-14
- “Generating Views from the Command Line” on page 2-17
- “Generating Specification and Log Files” on page 2-19
- “Compiler Options” on page 2-22

2.1 Overview

ARM's Artisan memory compilers provide integrated circuit designs with the highest levels of density, speed, and power. A wide range of features provides several options, including the ability to increase chip reliability and yield. The compilers tailor instances with a large variety of selectable features and create a comprehensive set of views for use with industry standard EDA tools and flows. You can run the compiler by invoking the graphical user interface (GUI) or from the command line. This chapter provides information on using the compiler to tailor instances to your design needs.

2.1.1 Running the Compiler from the Graphical User Interface (GUI)

ARM's Artisan compiler GUI allows you to configure all compiler parameters and generate all views from a single graphical interface. The output views, along with a log file, are placed in the current working directory.

This manual assumes you have added `<install_dir>/aci/<executable>/bin` to your UNIX search path. If you do not wish to do this, preface all compiler commands with `<install_dir>/aci/<executable>/bin/`.

To start the GUI from the shell, type:

```
% cd <working_dir>
% <executable>
```

where:

`<working_dir>` refers to the directory where you choose to run the compiler. Compiler output files are created in this directory; therefore, ARM strongly recommends that you run the compiler in a working directory that is different from the source directory `<install_dir>`.

Refer to the table in "Directory Structure and Executables" on page 1-8 for a list of standard compiler executables.

2.1.2 Running the Compiler from the Command Line

You can use command-line options to set parameter values, generate views, and use view-specific options.

The syntax described below applies to all options, parameter values, and views generated from the command line. All option names and parameter values are case-sensitive.

<executable> <view_command> <-option><option_value> ...

Commands that generate views do not require a dash (-) in front of the command. Options that set parameters, such as MUX or word values, do require a dash.

For example, to obtain an rom_via_hde_hvt_rvt_hvt instance with Verilog view, MUX = 8, words = 256, type:

```
rom_via_hde_hvt_rvt_hvt verilog -mux 8 -words 256
```

The table in "Directory Structure and Executables" on page 1-8 provides a list of standard compiler executables. Refer to "Generating Views from the Command Line" on page 2-17 for details about specific commands you can use to generate specific views.

2.2 Views and Output Files

You can generate a variety of views from the GUI or the command line. Each view may consist of one, or more, output file. You can apply basic and advanced options or parameters to each view. Refer to "Generating Views from the GUI" on page 2-14, "Generating Views from the Command Line" on page 2-17, and "Compiler Options" on page 2-22 for details about adding these parameters to your views.

The following table lists standard and optional views you can generate and output file(s) associated with each view. The instance name is the executable name, in capital letters.

Table 2-2. Views and Output Files

View	Output Files	Note(s)
Verilog model	<instance_name>.v	
Synopsys (Liberty) for each corner - includes CCS-noise	<instance_name>_<corner>_syn.lib	1,2, 3
VCLEF footprint	<instance_name>.vclef <instance_name>_ant.lef <instance_name>_ant.clf	
GDSII layout file	<instance_name>.gds2	
Mentor FastScan	<instance_name>.fastscan	
Synopsys (Liberty) TetraMAX	<instance_name>.tv	4
CDB Enablement	N/A	
VoltageStorm Enablement	N/A	
PostScript datasheet	<instance_name>.ps	
ASCII datatable	<instance_name>.dat	

- ¹ You can create timing models using any of the Process Voltage Temperature (PVT) corners for which the memory compiler was characterized. The compiler may support more than four characterization corners; however, you can only create timing models for only four corners at a time. The characterization corner name (that is, slow, fast, fast@-40C, fast@125C, typical) is inserted into the output filename (for example, rom_via_hde_hvt_rvt_hvt_<corner>_syn.lib).
- ² The typical and slow Synopsys models are generated with maximum delays, and the fast model is generated with minimum delays. ARM recommends that critical path, setup and hold analysis be performed for all applicable corners.
- ³ Synopsys models are generated with maximum alternate current (AC) values for each supported corner. Depending on chip design, overall chip level worst case power conditions can occur under the fast corner (PVT conditions) or under the "Maximum Static Power" corner condition. The worst case static power occurs under the

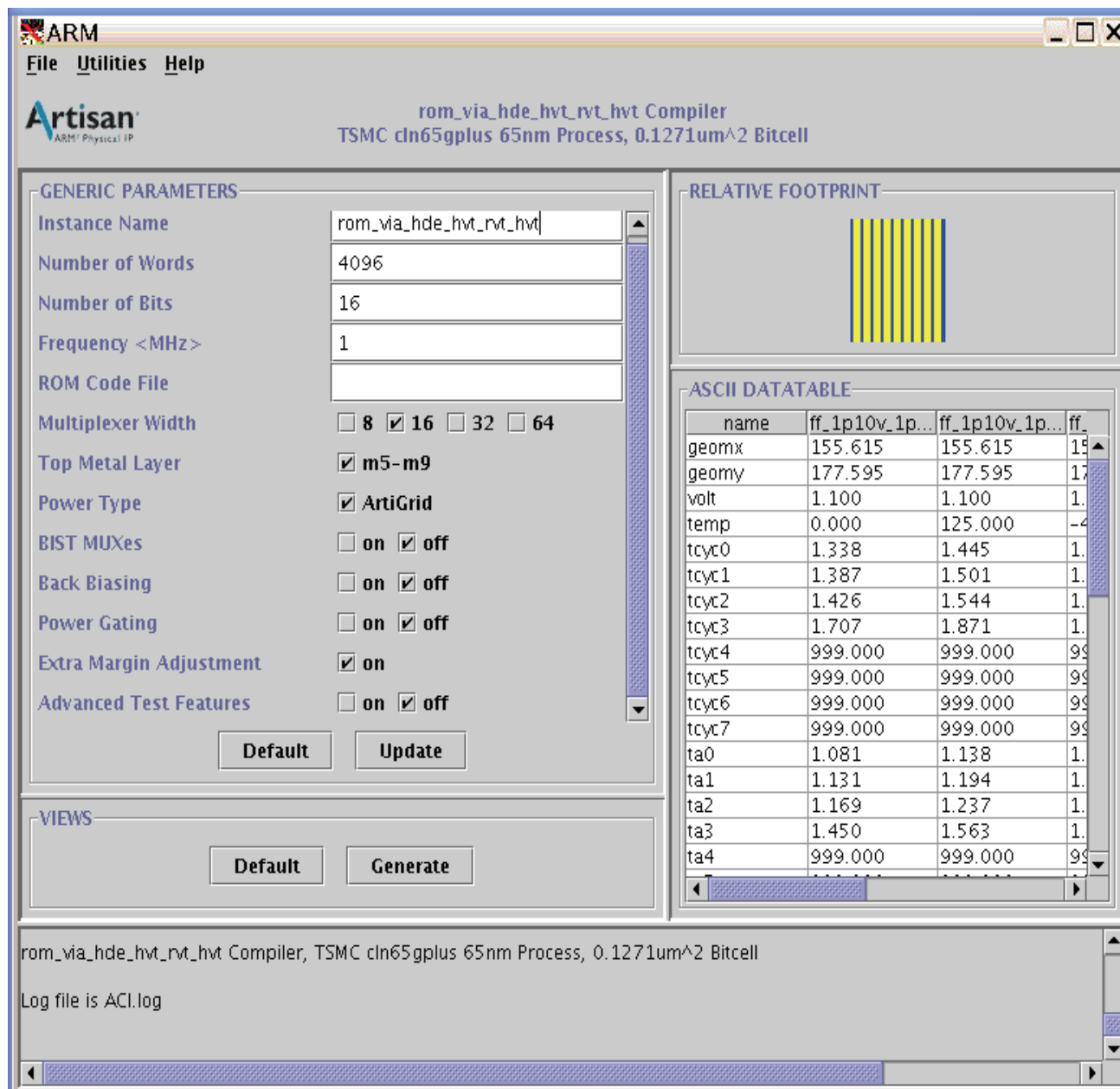
maximum temperature, fast process and maximum VDD. The static power corner models both AC and static power under this condition. You may need to perform chip level power analysis under both the fast and “static power” corners to determine the maximum overall power dissipation, AC plus static, for your design.

- ⁴ This optional support is available for free to ARM’s Artisan Access (Free) Library Program licensees under ARM’s Artisan EDAPlus programs with EDA partners.

2.3 GUI Components

A sample ROM compiler GUI is shown in Figure 2-1. The GUI for your compiler may not look exactly the same as this sample. For instance, your compiler may have additional characterization corners or features that are not enabled. You can resize the GUI by clicking on its border and dragging it to the desired position.

Figure 2-1. Example: ROM Compiler GUI



2.3.1 Generic Parameters Pane

The *Generic Parameters* pane of the GUI contains standard input fields and check boxes. The generic parameters are the most commonly used parameters used to configure a compiler instance. Refer to Figure 2-1 "Example: ROM Compiler GUI" on page 2-7. You can change the value of a generic parameter by typing the new value in the input field or by selecting the box corresponding to the desired value for each option.

When you want to submit the values of the generic parameters and update the ASCII datatable, click on the *Update* button in the *Generic Parameters* pane.

For example, you can generate views for a specific instance with 256 words, 16 bits, and multiplexer width 8. Enter "256" in the *Number of Words* field, "16" in the *Number of Bits* field, and select the box corresponding to "8" for multiplexer width. Leave all other parameters set to their default values. Click on the *Update* button.

Be sure to enter values that are within the pre-determined ranges for your compiler. Refer to Chapter 3 for parameter ranges. You can also use the message pane in the compiler GUI to determine parameter ranges. If you attempt to generate views with an out-of-range value, a message identifying the legal (valid) range is displayed in the message pane.

To reset the generic parameters to their default values, click on the *Default* button in the *Generic Parameters* pane.

2.3.2 Views Pane

You can generate a single view at a time from the *Views* pane in the compiler GUI. To generate a single view, select the view you want from the *Views* pull-down menu and click on the *Generate* button in the *Views* pane. The corresponding view is generated and placed in the current working directory *<working_dir>*. A list of available views and output files is shown in Table 2-2 "Views and Output Files" on page 2-5. For detailed information about using these views refer to Chapter 4, "EDA Tools Support".

You can also generate multiple views at one time. Refer to "Generating Multiple Views" on page 2-15.

You can cancel a view generation from the GUI. When a view is being generated, a window displays a message stating which view is being generated, and a *Cancel* button. Click on the *Cancel* button to cancel generating that view.

2.3.3 Relative Footprint Pane

The *Relative Footprint* pane of the GUI shows how the aspect ratio of the ROM changes as the words, bits, and MUX parameters are varied. Refer to Figure 2-1 "Example: ROM Compiler GUI" on page 2-7, which shows the relative footprint in the top right-hand corner of the GUI.

When you change a generic parameter and press the Update button, the relative footprint is automatically updated.

2.3.4 ASCII Datatable Pane

When you invoke the GUI, values for the default instance are displayed in the *ASCII Datatable* pane. When you change the generic values in the GUI, and click on the *Update* button in the *Generic Parameters* pane, the ASCII datatable automatically updates to reflect the new values.

You can obtain a print-ready copy of these values in two ways. From the Views pane of the GUI, select PostScript datasheet or ASCII datatable. The resulting datasheet (*<instance_name>.ps*) or text file (*<instance_name>.dat*) show the values in the datatable.

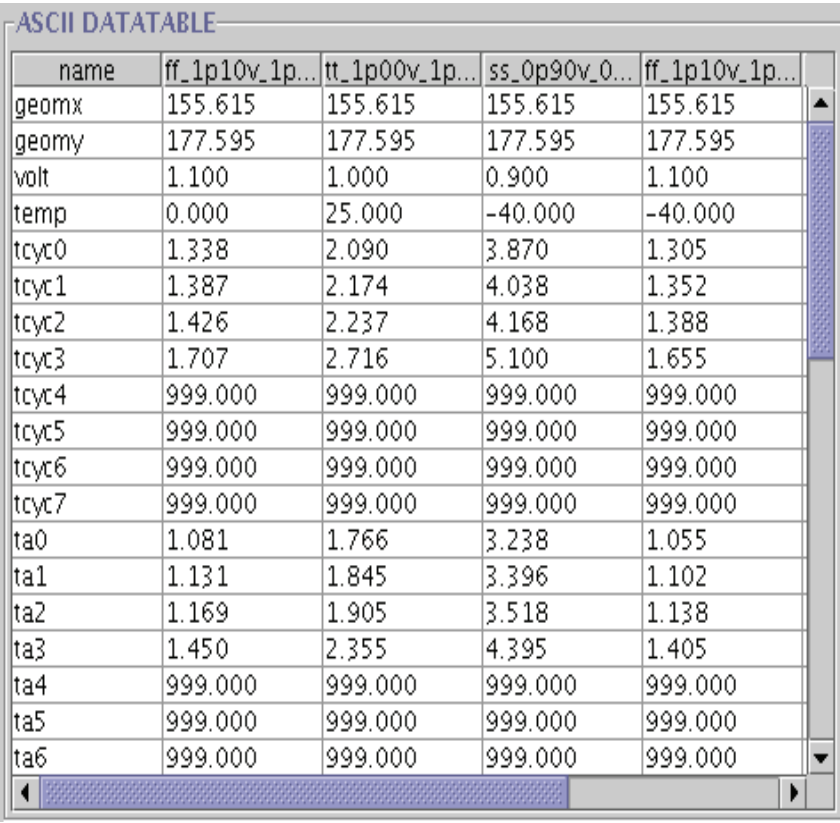
Figures 2-2 and 2-3 show sample ASCII datatable panes. The corners and parameters shown may not be the same as those in your high density compiler GUI. Information about minor deviations to the high density compilers can be found in the README text file that is enclosed with your compiler or in an addendum attached to this manual.

In the ASCII databale and postscript data sheets, non-power values are displayed in decimal format. Current/power values in datatables and datasheets may be shown in scientific notation or in decimal format.

——— **Note** ———

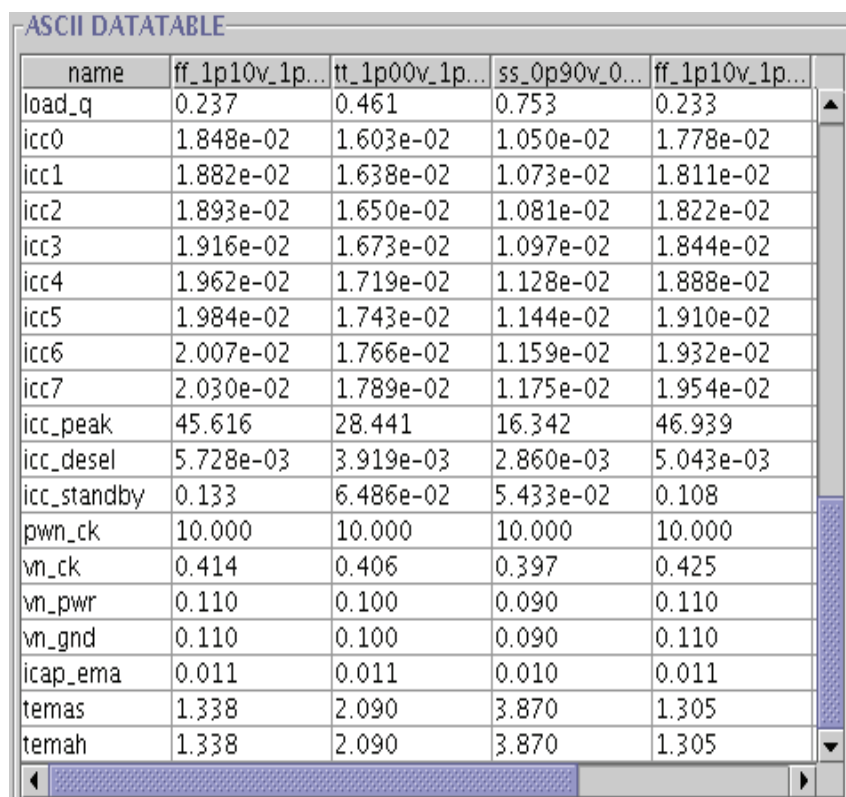
The display allows a maximum of three figures to the right of the decimal point (0.000). In order to fully present values that exceed this limit, scientific notation uses an “E” to notify you that notation has been invoked and by how many places you must move the decimal point to the left in order to obtain the correct decimal value.

For example, the scientifically notated value of 3.214E-2 means that the decimal point must be moved two places to the left to yield a decimal equivalent of 0.03214.

Figure 2-2. Example: ASCII Datable Pane


The image shows a software window titled "ASCII DATABLE". Inside, there is a table with 5 columns and 20 rows. The columns are labeled "name", "ff_1p10v_1p...", "tt_1p00v_1p...", "ss_0p90v_0...", and "ff_1p10v_1p...". The rows contain numerical data for various sensors. The table is scrollable, with a vertical scrollbar on the right and a horizontal scrollbar at the bottom. The data is as follows:

name	ff_1p10v_1p...	tt_1p00v_1p...	ss_0p90v_0...	ff_1p10v_1p...
geomx	155.615	155.615	155.615	155.615
geomy	177.595	177.595	177.595	177.595
volt	1.100	1.000	0.900	1.100
temp	0.000	25.000	-40.000	-40.000
tcyc0	1.338	2.090	3.870	1.305
tcyc1	1.387	2.174	4.038	1.352
tcyc2	1.426	2.237	4.168	1.388
tcyc3	1.707	2.716	5.100	1.655
tcyc4	999.000	999.000	999.000	999.000
tcyc5	999.000	999.000	999.000	999.000
tcyc6	999.000	999.000	999.000	999.000
tcyc7	999.000	999.000	999.000	999.000
ta0	1.081	1.766	3.238	1.055
ta1	1.131	1.845	3.396	1.102
ta2	1.169	1.905	3.518	1.138
ta3	1.450	2.355	4.395	1.405
ta4	999.000	999.000	999.000	999.000
ta5	999.000	999.000	999.000	999.000
ta6	999.000	999.000	999.000	999.000

Figure 2-3. Example: ASCII Datable Pane with Scientific Notation


The screenshot shows a window titled "ASCII DATABLE" containing a table with 5 columns. The first column is labeled "name" and lists various parameters. The other four columns contain numerical values, some in scientific notation. The table is scrollable, with a vertical scrollbar on the right and a horizontal scrollbar at the bottom.

name	ff_1p10v_1p...	tt_1p00v_1p...	ss_0p90v_0...	ff_1p10v_1p...
load_q	0.237	0.461	0.753	0.233
icc0	1.848e-02	1.603e-02	1.050e-02	1.778e-02
icc1	1.882e-02	1.638e-02	1.073e-02	1.811e-02
icc2	1.893e-02	1.650e-02	1.081e-02	1.822e-02
icc3	1.916e-02	1.673e-02	1.097e-02	1.844e-02
icc4	1.962e-02	1.719e-02	1.128e-02	1.888e-02
icc5	1.984e-02	1.743e-02	1.144e-02	1.910e-02
icc6	2.007e-02	1.766e-02	1.159e-02	1.932e-02
icc7	2.030e-02	1.789e-02	1.175e-02	1.954e-02
icc_peak	45.616	28.441	16.342	46.939
icc_desel	5.728e-03	3.919e-03	2.860e-03	5.043e-03
icc_standby	0.133	6.486e-02	5.433e-02	0.108
pwn_ck	10.000	10.000	10.000	10.000
vn_ck	0.414	0.406	0.397	0.425
vn_pwr	0.110	0.100	0.090	0.110
vn_gnd	0.110	0.100	0.090	0.110
icap_ema	0.011	0.011	0.010	0.011
temas	1.338	2.090	3.870	1.305
temah	1.338	2.090	3.870	1.305

You can resize the columns in the ASCII datatable by clicking on the column border and dragging it to the desired width. The columns can be rearranged by clicking on the header cell of a column and dragging it to the desired position.

The “name” column lists the acronym for a characterized parameter. The other columns contain values for these parameters at selectable PVT corners. Characterized parameters are described in the “Timing Parameters” section in Chapter 3. Also, by moving your cursor over the parameter name in the GUI, you can get a brief description of that parameter.

The units for parameters in the ASCII datatable are listed in Table 2-3.

Table 2-3. ASCII Datable Units

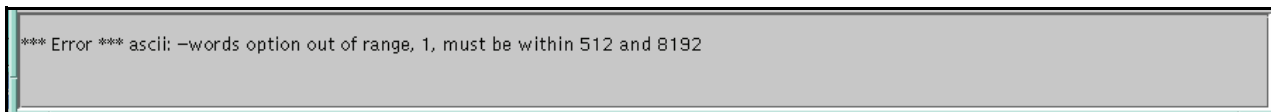
Parameters	Units
Geometry	Microns
Current-consumption	Milliamperes
Timing	Nanoseconds

2.3.5 Message Pane

The message pane is located at the bottom of the GUI frame. This pane displays messages when you generate a new instance. Messages include information about successfully generated views and associated output files, an updated ASCII datatable, and when generic parameters are reset to their default values.

The message pane also displays error messages, such as when an invalid value is entered into the GUI or when view generation is not successful. For example, if you enter “1” in the *Number of Words* field, and click the *Update* button, the error message in the message pane indicates that this value is out of range, as shown in Figure 2-4. The valid range, 512 to 8192 in this case, is also provided.

Figure 2-4. Example: Message Pane



The log file stores all the messages that appear in the message pane. You can clear the message pane by selecting the *Utilities* Menu, then selecting *Purge Message Area*. The messages are still retained in the log file.

2.3.6 File Menu (Exiting the GUI)

To exit the GUI, select the *File* pull-down menu, then select *Exit*.

2.3.7 Utilities Menu

You can use the Utilities Menu to access other menus and options. You can use it to write specification files, generate multiple views, select corners, and set advanced options. Figure 2-5 shows a sample Utilities pull-down menu.

Figure 2-5. Example: Utilities Pull-Down Menu

Refer to "Using Specification Files" on page 2-19, "Generating Multiple Views" on page 2-15, and "Setting Advanced Options from the GUI" on page 2-26, for details on how to perform these tasks.

2.3.8 Help Menu

The *Help* pull-down menu displays a list of documents that are shipped, in electronic format, with the GUI. When you select a document the Adobe PDF reader, *acroread*, launches to open the document. If the document does not display properly, ask your system administrator to ensure that *acroread* is available to you and is in your path.

2.3.9 Balloon Help

The GUI contains balloon help messages that give brief explanations of compiler features. The balloon help messages appear as your mouse pauses over an active area such as ASCII datatable parameters. Pausing your mouse over the ASCII datatable allows you to view brief descriptions of the parameters and the process-temperature-voltage (PVT) data for each characterization environment (corner).

2.4 Generating Views from the GUI

You can create single or multiple views with specific parameters from ARM's Artisan compiler GUI.

2.4.1 Generating Single Views

You can create a single view for each instance directly from the GUI. For each new instance, update the text fields and check boxes in the Generic Parameters pane and click *Update*. In the Views pane, select a view from the pull-down menu and click *Generate*. When you generate a view, the Message pane at the bottom of the GUI displays a message, showing the success of the generated view and any output file(s) created.

You can also set additional parameters in the Advanced Options dialog box under the Utilities pull-down menu. For additional information, go to "Setting Advanced Options" on page 2-26.

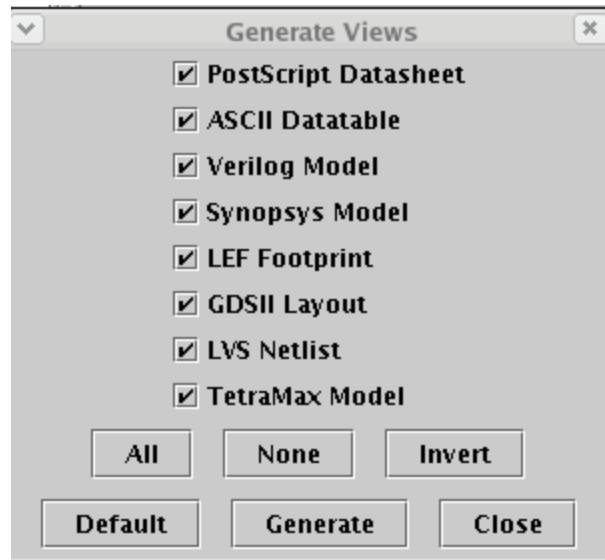
For instance, if you want to create datasheets for multiple instances, you can start by changing the generic parameters shown in the Generic Parameters section or in the Advanced Options pull-down menu of the GUI. Click on Update in the GUI. Select "Postscript Datasheet" from the Views pull-down menu in the Views pane of the GUI. Click on Generate.

An output datasheet called `<instance_name>.ps` is placed in your current `<working_dir>` directory. Now, you can change the parameters and instance name to suit another instance. Click on Generate to create a new datasheet for your new instance.

2.4.2 Generating Multiple Views

You can also generate all available views, or a selection of views, at one time. Select the *Utilities* pull-down menu from the GUI, as shown in Figure 2-5 "Example: Utilities Pull-Down Menu" on page 2-13. Then select *Generate Menu*, such as the sample shown in Figure 2-6 "Example: Generate Menu." The *Generate Menu* window shows a list of views that you can generate.

Figure 2-6. Example: Generate Menu



When this menu is first opened, all views are selected. Click on the box corresponding to a view to toggle between selecting and deselecting the view. When a view is selected, the box contains a check mark. When you click on the *All* button, all views listed are selected. When you click on the *None* button, all views listed are deselected. When you click on the *Invert* button, the selection of views is inverted, or reversed.

When you click on the *Default* button, the parameters for selected views are reset to their default values. When you click on the *Generate* button, all selected views are generated and placed in the current working directory *<working_dir>*. When you generate multiple views, the Message pane at the bottom of the GUI displays a message, that shows the success of generated views and any output files created.

If you close the *Generate-Menu* window and reopen it during the same GUI session, the most recently selected list is recalled.

If you cancel the view generation operation, the current view is cancelled, and the remaining views are not generated.

2.4.3 Setting View-Specific Parameters

The *Views* pane of the GUI provides a pull-down menu that displays the views you can generate. When you select certain views, an input field appears. You can enter a view-specific parameter in this field, as described below.

To select a view, click on the corresponding option in the pull-down menu. If the view is not available, a “Not available” message is displayed in the *Views* pane. When you click on a view and click *Generate*, the parameters related to that view appear in the message pane at the bottom of the GUI.

Click *Default* in the Views pane to set the view-specific parameters to their default values. As you move from view to view, the parameter values for each view are retained.

——— **Note** ———

You can also refer to "[Advanced View-Specific Options](#)" on page 2-29 for more information on options that are specific to particular views.

2.5 Generating Views from the Command Line

You can generate views directly from the command line. Refer to Chapter 4, "EDA Tools Support" for details about using the views and output files.

2.5.1 View Commands

The following table provides the commands you can use to generate standard and optional views from the command line.

Table 2-4. View Commands

View	View-Command
Verilog model	verilog
Synopsys (Liberty) - includes CCS-noise	synopsys
VCLEF footprint	vclef-fp
GDSII layout file	gds2
Synopsys TetraMAX	tmax
CDB Enablement	N/A
Voltage Storm Enablement	N/A
PostScript datasheet	postscript
ASCII datatable	ascii

You can run the compiler directly from the command line using various commands which instruct the compiler to produce the selected view or instance with specific parameters. Refer to "Running the Compiler from the Command Line" on page 2-4 for instructions. Refer to "Command Line Syntax" on page 2-22 for details about writing commands to run the compiler correctly.

You may use more than one command on your command line.

For example:

```
% rom_via_hde_hvt_rvt_hvt vclef-fp -diodes on -mux 8 ...
```

You may use more than one view command on your command line. The specification file and individual option selections apply to all of the views selected for the run.

For example:

```
% rom_via_hde_hvt_rvt_hvt vclef-fp gds2 synopsys -mux 8 ...
```

2.5.2 Generating Multiple Views with View-Specific Options

Certain options are view-specific, they only apply to certain views. For example, the `site_def` option only applies to the VCLEF view and the `libname` option only applies to the Synopsys (Liberty) view.

When generating multiple views on your command line, you must specify the view-specific options in detail, as in the following example:

```
% <executable> <view_command_1> <view_command_2>  
  -<view_command_1>.<view-specific_option_1>  
  <view-specific_option_value_1>  
  -<view_command_2>.<view-specific_option_2>  
  <view-specific_option_value_2>
```

For example, to generate both Synopsys and VCLEF-fp views, apply the `-libname` option and supply a view-specific value for each option, type:

```
% rom_via_hde_hvt_rvt_hvt synopsys vclef-fp -synopsys.libname  
  <syn_userlib>  
  -vclef-fp.diodes on
```

The following table shows these view-specific commands, in sequential order, compared to the entries in the previous examples.

Commands (in sequence)	Example
%<executable>	%rom_via_hde_hvt_rvt_hvt
<view_command_1> <view_command_2>	synopsys vclef-fp
-<view_command_1>.<view-specific_option_1>	-synopsys.libname
<view-specific_option_value_1>	syn_userlib
-<view_command_2>.<view-specific_option_2>	-vclef-fp.diodes
<view-specific_option_value_2>	on

2.6 Generating Specification and Log Files

You can generate files that save the parameters and specifications of each instance. This section tells you how to write a specification file for each instance and create a log file to record commands and output files. In addition, this section describes instance parameter information that displays in the top of most views. This feature allows you to easily identify the parameters generated with each view.

2.6.1 Using Specification Files

You can create an ASCII text file for each instance you generate, with all the specific parameters and options you selected for that instance.

When you select the Utilities pull-down menu, then select *Write Spec*, a specification file is generated and placed in the current working directory <working_dir>. The name of the generated specification file is <instance_name>.spec, where *instance_name* is the name shown in the *Generic Parameters* pane of the GUI at the time the specification file is generated. This file may be edited and used for subsequent runs.

You can create or modify your own specification file using any ASCII text editor. The format for this file is a list of the options you want to apply to your model. You may use either a space or an equal sign "=" between the option name and the value. When using options in a specification file, do not place a dash "-" in front of the option name. Figure 2-7 is an example of a simple specification file that includes a few options.

Figure 2-7. Example: Specification File

```
prefix MY_  
instname <instname>  
mux 8  
words 256  
vclef-fp.site_def=off  
vclef-fp.diodes=on  
top_layer=m8
```

Name your specification file, and save it. Your specification file can now be used to configure the generated instance the next time compiler is invoked. Launch the GUI with the parameter values from your specification file as the defaults:

```
% <executable> -spec <spec_file>
```

where *<spec_file>* is the name of your specification file.

The specification file may also be used with the compiler commands. Refer to the "Compiler Options" section on page 2-22.

2.6.2 Creating Log Files

A log file containing a record of GUI-generated instances and output messages is placed in the same working directory. A log file is not created when you generate instances from the command line.

When a view is generated or parameters are initialized to default values, a message is recorded in the log file and shown in the message pane of the GUI. The usual name for this file is ACI.log. Although the message pane clears when you select the *Utilities* Menu and then select *Purge Message Area*, the log file retains a full record of messages.

By default, each time the GUI is invoked, the log file created for that run overwrites the existing log file. You may choose to keep the existing log file(s) and create a new log file for the current session. To launch the GUI and keep existing logs, creating a new log file:

```
% <executable> -keeplogs
```

The next log file is an incremental name generated by the system, for example, ACI.log.1.

2.6.3 Generating Parameter Information

Parameter information identifies the strings and selections in the compiler's fields and options. When you generate an instance from the GUI, a message containing parameter information appears in the message pane at the bottom of the GUI. These values are shown as you would enter them on the command line, as a set of parameters/options and their values.

Parameter information for an instance also outputs at the beginning of all generated views, except the postscript datasheet and ASCII datatable views. An example is shown in Figure 2-8 "Example: Parameter Information." You must change the instance name in the GUI to retain information unique to each instance. If the instance name does not change, the previous information will be overwritten when you regenerate.

Figure 2-8. Example: Parameter Information

```
* name:          xxx Compiler
*
*                Artisan xxnm Process
* version:       2008Q3V1
* comment:       080822_6:38_03
* configuration: -instname INSTANCExxx -words 4096 -bits 16
-frequency 1 -mux 16 -drive 6 -wp_size 8 -top_layer met8 -
power_type artigrid -cust_comment "080822_6:38_03" -
left_bus_delim "[" -right_bus_delim "]" -pwr_gnd_rename
"VDD:VDD,GND:VSS" -prefix "" -pin_space 0.0 -name_case
upper -check_instname on -diodes on
```

Included in this parameter information is the customer comment option, which allows you to add a unique identifier such as the date and time you generate a particular instance. You can use this option to add more detail than in the instance name. The example above shows a parameter information string, with "080822_6:38_03" as the customer comment. For more information on the customer comment string, see "Setting Advanced Options" on page 2-26.

2.7 Compiler Options

The standard options described in this section allow you to customize your compiler to create specific memory instances. These options can be accessed from the command line and from the compiler GUI.

The option is listed first, followed by the type of parameter, and then a description of the option. For instance the parameter for the "MUX" option is a number. You can refer to the parameters tables in Chapter 3 for standard parameter ranges.

In some compilers, some options or parameters may not be available when other options are selected. The options will still be visible, but are shown in grey when disabled.

2.7.1 Command Line Syntax

Use the following syntax for all options, including as many options as you want to set. Refer to "Directory Structure and Executables" on page 1-8, for information about the executable for your compiler.

```
<executable> [<view_command>] [-spec <filename>] [-mux <number>]
```

You may supply any combination of options and specification files on the command line. On the command line, use the dash '-' in front of the option. In the case of duplicate options or parameters, the last option set takes precedence.

2.7.2 Basic Options

-spec filename

The specification file contains a list of basic, advanced, and view-specific options and values for the compiler. You may create a new specification file, or files, to customize the options that you want to use repeatedly. If you are creating new specification files, be sure to read the "Using Specification Files" section on page 2-19.

-help

You can access the help information for a compiler or compiler view. Information such as available views and options is displayed as a text message on the command line. You can access help information that applies to specific views by including the view command for that view. Refer to "View Commands" section on page 2-17 for a list of view commands.

To access the help for a compiler, be sure you have added `<install_dir>/aci/<executable>/bin` to your UNIX search path. If you do not wish to do this, preface all compiler commands with `<install_dir>/aci/<executable>/bin/` and type:

```
<executable> -help
```

To access the help for a compiler view, type:

```
<executable> <view_command> -help
```

For example, type "rom_via_hde_hvt_rvt_hvt ascii -help" to view the help information related to the ASCII datatable, such as compiler parameters.

-instname *name*

The default instance name is the same as the executable name, in capital letters. For instance, if the executable is `rom_via_hde_hvt_rvt_hvt`, the default instance name is `rom_via_hse`. Refer to "Directory Structure and Executables" on page 1-8, regarding the executable name for your compiler.

You can set the instance name to any alphanumeric value. To avoid name conflicts for instances within the same library, you must enter a unique instance name. To avoid tool compatibility issues, an instance name of 16 characters or fewer is recommended, and it should begin with a letter. See the additional information in the `-check_instname` and `-prefix` option descriptions.

`-words` *number*

The compiler GUI shows the default words value. The range for words depends on the multiplexer width, limited by the physical array of memory cells. Refer to Chapter 3 for the word ranges for standard compilers.

`-bits` *number*

The compiler GUI shows the default bits value. The range for bits depends on the multiplexer width, limited by the physical array of memory cells. Refer to Chapter 3 for the bit ranges for standard compilers.

`-frequency` *number*

The compiler GUI shows the default frequency value, in megahertz (MHz). The frequency can be set to any positive integer value, up to the inverse of the cycle time in nanoseconds (ns) multiplied by 1000. The frequency parameter is used to scale the AC current-consumption datatable values. If it is left as the default value of 1.0 megahertz, the units on the AC current-consumption datasheet values will be milliamperes per megahertz.

`-code_file` *filename*

This field defines the data programming filename for the ROM. An Artisan-format ROM code file must be provided for each generated instance. The ROM code file is only required for some views. The GUI checks for a valid ROM code file, and issues an error message if a ROM code file does not exist or is invalid. For more details, refer to the *ROM Code File* section in Chapter 3.

`-mux` *number*

The compiler GUI shows the default MUX value and any choices for this option. Refer to Chapter 3 for the MUX values for specific compilers.

`-top_layer` *m5-9*

The compiler GUI shows the default value for the top metal layer and any choices for this option. For more details, refer to "Top Metal Layer" section on page 3-30.

`-power_type` *ArtiGrid*

Power is supplied through the ArtiGrid structure that is over-the-cell (OTC).

`-bmux` *on/off*

When this option is turned on, the compiler adds MUXes at most input and output pins. The MUXes can be used by BIST engines to access the memory in test mode. The default value is off.

`-back_biasing` *on/off*

This option allows you to select the back bias pins in order to reduce leakage. When back biasing is selected the VPW and VNW pins are available at the instance boundary as input pins. The default value is off.

`-power_gating` *on/off*

This is a power structure option in which the core array and periphery power supplies are separated. It is not available if ArtiGrid is selected. The default value is off.

`- atf` *on/off*

This is an advanced test feature used to enable/disable weak bit test (WBT) feature. The weak bit test (WBT) feature detects a weak bit in a core array. Selecting on (1) asserts the WBT high. The default value is off (0).

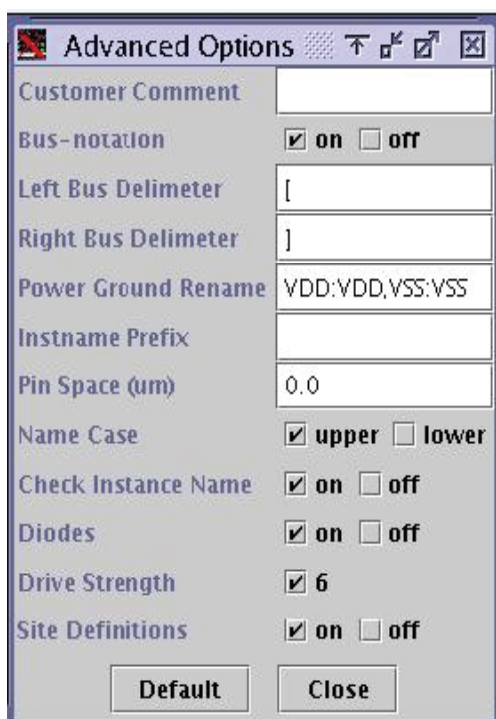
2.7.3 Setting Advanced Options

You can set advanced options from the compiler GUI or by entering the options on the command line. This section demonstrates the methods for setting these options, including some options that apply only to specific views. Some of these options depend on the setting of the generic parameters in the main GUI.

2.7.3.1 Setting Advanced Options from the GUI

From the GUI, select the *Utilities* pull-down menu, then select the *Advanced Options* window. This window displays the advanced options that you can specify, as shown in Figure 2-9. Enter a string or number in the fields, or select the check boxes as needed.

Figure 2-9. Example: Advanced Options Window



2.7.3.2 Setting Advanced Options from the Command Line

From the command line, type the executable name, then the option preceded by a dash "-" and then the parameter you want to specify. You can type as many options and parameters as you need. Refer to the "Command Line Syntax" section on page 2-22 to be sure you are entering commands correctly.

2.7.4 Advanced Options

`-customer_comment string`

A customer-specified text field of up to 64 characters. The allowed character set is alphanumeric, plus the following characters: '_', '-', '.', ':', '=', and '+.

The customer comment string is included in the parameter information that appears at the top of all views, except the postscript datasheet and ASCII datatable. You can use this string to differentiate between different instances with more characters than the instance name allows. For more information about the customer comment string, see "Generating Parameter Information" on page 2-21.

`-bus_notation on`

By default, all bus pins are represented by bus notation in the interface of models (e.g., A[3:0]).

`-left_bus_delim /|<|/`

The Advanced Options menu shows the default value for the Left Bus Delimiter option. This option specifies the left bus delimiter for physical views, including VCLEF, GDSII, and LVS. Delimiter choices are "[," "<," or "{." Bus delimiters for front-end views are tool specific, and are not affected by using this option.

`-right_bus_delim /|>|/`

The Advanced Options menu shows the default value for the Right Bus Delimiter option. This option specifies the right bus delimiter for physical views, including VCLEF, GDSII, and LVS. Delimiter choices are "],," ">," or "}." Bus delimiters for front-end views are tool specific, and are not affected by using this option.

`-pwr_gnd_rename` *string*

This option allows you to name the power and ground net names for backend views. An example string, VDD:VCC,VSS:GND, will rename power "VCC" and ground "GND."

When back biasing is selected, VNW and VPW pins are visible.

`-prefix` *name*

This option allows you to assign a prefix for the instance name. If this option is left blank, no prefix is applied to the instance name. The prefix is counted when using `-check_instname`.

`-name_case` *upperlower*

The default is upper. This option specifies the case for sub circuit and net names, excluding the top-level instance name and power/ground names.

`-check_instname` *on/off*

The Advanced Options menu shows the default value for the Check Instance Name option and any choices for this option. An instance name should begin with a letter, and should not exceed 16 characters. If this option is turned on and the name does not meet these requirements, the GUI issues error messages, and does not generate views. If the option is turned off, the GUI issues warning messages, but it will generate views. Both errors and warnings display in the message pane, and are recorded in the log file.

If the compiler was launched from the command line and the instance name is greater than 16 characters, the error and warning messages appear on the terminal.

`-drive` *number*

The Advanced Options menu shows the output drive strength.

2.7.4.1 Advanced View-Specific Options

This section lists advanced options that apply only to certain views.

`-diodes on|off`

The Advanced Options menu shows the default value for the Diodes option and any choices for this option. Because the default value indicates how the compiler was validated, the LVS rule set may not support the non-default value.

If the Diodes option is off, antenna diodes present in the GDSII view are omitted in the LVS Netlist view.

`-site_def on|off`

The default value is off. This option specifies whether VCLEF contains a site definition. This option applies to only the VCLEF model.

3

Synchronous ROM Compiler Architecture

This chapter contains the following sections:

- “Overview” on page 3-3
- “Synchronous ROM Architecture and Timing Specifications (rom_via_hde_hvt_rvt_hvt)” on page 3-4
- “ROM Power Structure (rom_via_hde_hvt_rvt_hvt)” on page 3-22
- “ArtiGrid Power Structure Options (rom_via_hde_hvt_rvt_hvt)” on page 3-27
- “ROM Physical Characteristics (rom_via_hde_hvt_rvt_hvt)” on page 3-30
- “ROM Timing Derating (rom_via_hde_hvt_rvt_hvt)” on page 3-32

3.1 Overview

This chapter describes the architecture, features, timing characterization, and physical characteristics for via ROM compilers. In this manual, contact ROMs are considered structurally similar to via ROMs and are not addressed separately.

———— **Note** ————

Information about significant functional or architectural deviations to the compilers can be found in the README text file enclosed with your compiler or in an addendum attached to this manual.

The following table lists the complete compiler names, product names, and executable names for the compilers described in this section.

Table 3-1. ROM Compiler Information

Compiler	Product Name	Executable
High density Via ROM	rom_via_hde_hvt_rvt_hvt	rom_via_hde_hvt_rvt_hvt

The following sections provide detailed information about your compiler.

- “Synchronous ROM Architecture and Timing Specifications (rom_via_hde_hvt_rvt_hvt)” on page 3-4
- “ROM Power Structure (rom_via_hde_hvt_rvt_hvt)” on page 3-22
- “ROM Physical Characteristics (rom_via_hde_hvt_rvt_hvt)” on page 3-30
- “ROM Timing Derating (rom_via_hde_hvt_rvt_hvt)” on page 3-32

3.2 Synchronous ROM Architecture and Timing Specifications (rom_via_hde_hvt_rvt_hvt)

This section describes the ROM compiler. It includes pin descriptions, logic tables, block and input parameters, block diagrams, core address maps, timing diagrams, and timing and power parameters. Executable names for applicable compilers are provided for your reference.

3.2.1 ROM Description (rom_via_hde_hvt_rvt_hvt)

Descriptions for the basic functionality of this compiler are provided. In addition, descriptions of features you can use to enable the test functions of your compiler are provided.

3.2.1.1 Basic Functionality

ROM access is synchronous and is triggered by the rising edge of the clock, CLK. The read port input address and chip enable are latched by the rising edge of the clock, respecting individual setup and hold times.

The value of chip enable must be low (CEN=0) for a read operation to occur. During a read cycle, data on the data output bus Q[n-1:0] is read from the memory location specified on the address bus A[m-1:0]. A read cycle is initiated if CEN is asserted at the rising edge of CLK. The contents of the location specified by the address, A, are driven on the data output bus, Q. The ROM is allowed to access non-existing physical addresses but the outputs will be unknown.

Power dissipation is minimized using static circuit implementations. A standby mode is provided to further reduce power dissipation during periods of non operation (CEN=1). While in standby mode, address inputs are disabled, and a read will not occur. Data outputs will remain stable. You can eliminate switching current in the input stages and reduce deselected current to near leakage by holding all input pins steady during standby mode.

Static power consumption of the ROM is limited to leakage provided all of the input signals except CLK are held steady.

ROM compilers are provided in via programmable format. Programmable via-ROMs encode the data by placing or omitting a via or contact in a pre-determined location.

A higher layer in the process fabrication step results in a greater bit cell area, but with a smaller number of fabrication changes for design modifications or upgrades. A lower layer results in a smaller bit cell area, but with a greater number of fabrication changes for design modifications or upgrades.

3.2.1.2 Test Functionality

The ROM compiler includes features such as Built-In Self Test (BIST) MUX and Extra Margin Adjustment (EMA), as described in this section. Details about the options you can select for these features are available in the “Compiler Options” section on page 2-22.

3.2.1.2.1 Extra Margin Adjustment (EMA)

Extra margin adjustment provides the option of adding delays to the internal timing pulse. This delay provides extra time for memory read operations by slowing down the memory access. There are three input pins, EMA[2], EMA[1], EMA[0], for each instance. The access time and cycle times are progressively increased as the pins are driven from 000 to 011 and from 100 to 111.

The EMA allocation is:

EMA000	OEN delay t_0 and no Precharge delay
EMA001	OEN delay t_1 and no Precharge delay
EMA010	OEN delay t_2 and no Precharge delay
EMA011	OEN delay t_3 and no Precharge delay

Where, $t_3 > t_2 > t_1 > t_0$

EMA100	OEN delay $(t_0 + t_{ow})$ and Precharge delay t_{ow}
EMA101	OEN delay $(t_1 + t_{ow})$ and Precharge delay t_{ow}
EMA110	OEN delay $(t_2 + t_{ow})$ and Precharge delay t_{ow}
EMA111	OEN delay $(t_3 + t_{ow})$ and Precharge delay t_{ow}

Note

The EMA setting is static and should be set before the memory operation.

3.2.1.2.2 Weak Bit Test (WBT)

The WBT feature detects a weak bit in a core array and is controlled by an external WBYT that is available at the instance memory. When WBT is asserted high (1), the margin between the sense amplifier output and output enable is reduced. A weak bitcell takes longer to manifest itself at the input sense node and causes a read '0' fail at the output. All other global timing pulses remain unchanged. The WBT pin needs to be tied low during normal operation.

3.2.1.2.3 Built-In Self Test (BIST) MUX

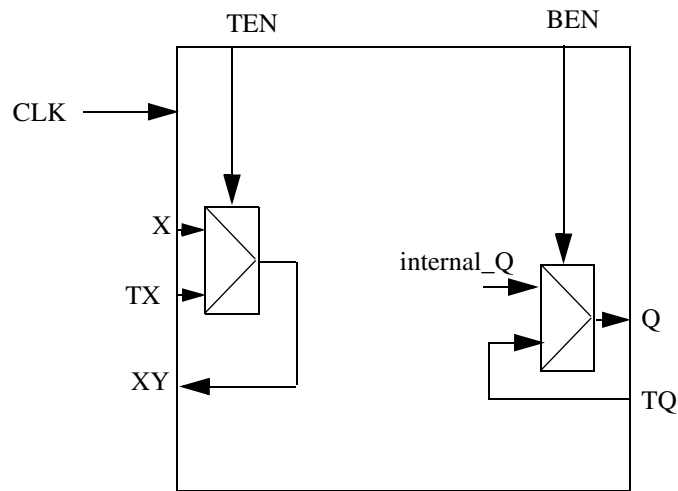
When the BIST feature is enabled, MUX are added to critical input pins and to all data output (Q) pins, CEN and A[].

One of the inputs of the input BIST MUX is connected to system signals while another is connected to the BIST outputs. When the Test Enable (TEN) pin is high, system inputs are selected; when TEN is low, test inputs are selected. The BIST MUX outputs are available as pins for testing. The address bus, A, has as its test address bus, TA[], and the MUX output is defined as AY[]. Similarly, CEN has test inputs defined as TCEN and MUX has outputs defined as CENY.

The bypass MUX are added before the output pins Q. These MUX allow the test tools to have direct control of the shadow logic, without going through the memory. The test MUX input on the bus Q[] is named TQ[]. When bypass enable, BEN, pin is low, TQ[] is driven to output. When BEN is high, the data from memory is output at Q[].

Figure 3-1 shows the BIST MUX block diagram.

Figure 3-1. BIST MUX Block Diagram



X can be A and CEN

3.2.1.2.4 Back Biasing Support

Back Biasing Support is a user selectable option that allows you to choose to drive the back bias pins in order to reduce leakage. When the option is selected, VPW and VNW pins are available at the instance boundary as input pins.

When there is a back biasing voltage fed to the pins, the performance is slower than standard operating mode performance.

If the option is not selected, the VPW and VNW pins are internally tied and not brought to the instance boundary.

3.2.2 ROM Pins (rom_via_hde_hvt_rvt_hvt)

Figure 3-2 shows basic and optional test pins for the ROM compiler.

Figure 3-2. ROM Basic and Test Pins

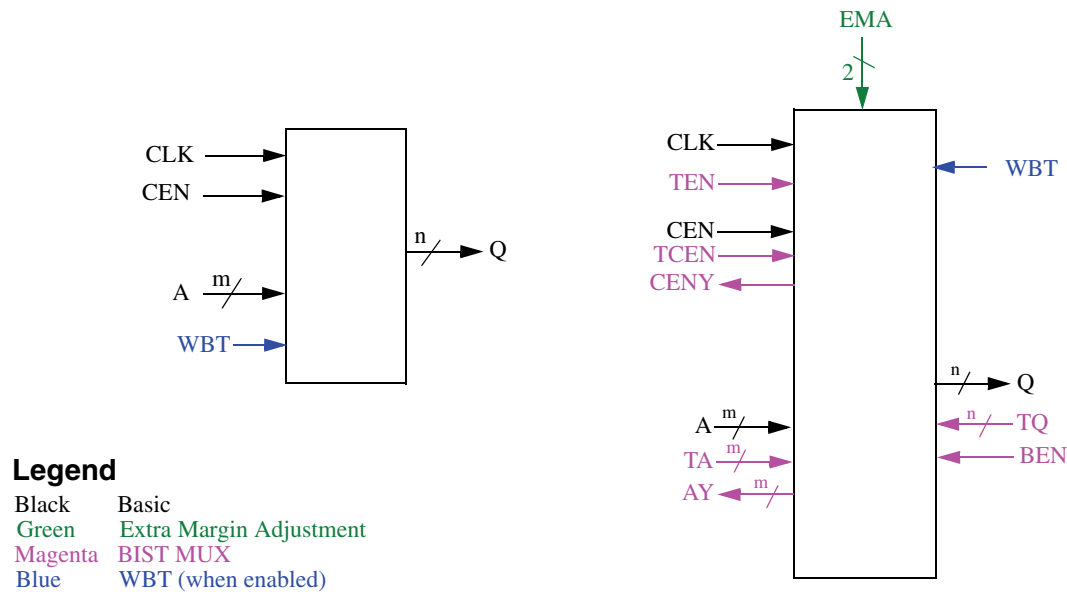


Table 3-2 provides the ROM compiler pin descriptions.

Table 3-2. Pin Descriptions for ROM Compilers

Name	Type	Description
Basic Pins		
A[m-1:0]	Input	Address (A[0] = LSB)
CEN	Input	Chip Enable, active low
CLK	Input	Clock
Q[n-1:0]	Output	Data output (Q[0] = LSB)
Extra Margin Adjustment Support Pins		
EMA[2:0]	Input	Extra Margin Adjustment
WBT	Input	Weak Bit Test; 0 = normal operation, 1 = test operation
BIST Mux Pins		
TEN	Input	Test Mode Enable, active low for test operation: 0=Test operation, 1=Normal operation
TA[m-1:0]	Input	Test Address (TA[0] = LSB)
AY[m-1:0]	Output	Address MUX output (AY[0] = LSB)
TCEN	Input	Test mode Chip Enable, active low
CENY	Output	CEN MUX output
BEN	Input	Bypass Mode Enable, active low for bypass operation. 0=Bypass operation, 1=Normal operation
TQ[n-1:0]	Input	Bypass Q input (if BEN=0, Q[n-1:0]=TQ[n-1:0])
Power Down Mode Pins		
PGEN	Input	Power down mode enable; active low
Back Bias Mode Pins		
VPW	Input	P-well back biasing voltage supply pin
VNW	Input	N-well back biasing voltage supply pin

3.2.3 ROM Logic Tables (rom_via_hde_hvt_rvt_hvt)

This section provides logic tables for basic ROM functions and for the individual test functions.

Table 3-3 presents the standard mode logic table.

Table 3-3. Standard Mode Logic Table

State	VDD	VSS	Outputs
Normal operation	On	On	Q (normal output)

Table 3-4 presents the standard mode logic table.

Table 3-4. Power Gating Mode Logic Table

PGEN	State	VDDP (internal signal)	VSSP (internal signal)	VSSE	Outputs
0	Normal mode	On	On	On	Q (normal outputs)
1	Power down	On	Off	On	Unknown

Logic functions for the basic ROM compiler features are shown in Table 3-5.

Table 3-5. ROM Basic Functions

CEN	Data Output	Mode	Function
H	Last Data	Standby	Address inputs are disabled and data outputs remain stable.
L	ROM data	Read	The memory location specified by the address bus, A[m-1:0], is read and driven onto the data output bus Q[n-1:0].

Table 3-6 describes the behavior of BIST MUX at inputs to the memory.

Table 3-6. ROM BIST MUX at Memory Input

TEN	Mode	Function
H	Regular Mode	In this mode, the regular pins, A[] and CEN are used for the internal operations described in Table 3-5. These inputs are also available at the MUX outputs: AY[] and CENY respectively.
L	Test Mode	In this mode, the test pins, TA[] and TCEN, are used for the internal operations described in Table 3-5. These inputs are also available at the MUX outputs: AY[] and CENY, respectively.

Table 3-7 describes the behavior of BIST MUX at the output pins.

Table 3-7. ROM BIST MUX at Memory Output

BEN	Mode	Q	Function
H	Regular Mode	Last Data Read	In this mode, the data from the last read operation is available at Q.
L	Test Mode (Data Bypass)	Data at bypass pin	Data at the bypass pin TQ is available at output Q. This operation is not clocked. All other operations described in Table 3-5, for basic features, work when BEN=0, with the exception that data does not appear at the output.

Table 3-8 lists the WBT pin states.

Table 3-8. WBT Pin States

Pin State	Mode
0	Normal (default) status
1	Weak bit test mode active

3.2.4 ROM Parameters (rom_via_hde_hvt_rvt_hvt)

The standard input and block parameters of a synchronous ROM are described in Table 3-9. Refer to your compiler GUI for specific input ranges for your compiler. If you enter an invalid value and update the GUI, the message pane in the GUI displays an error message and the specific range for your compiler.

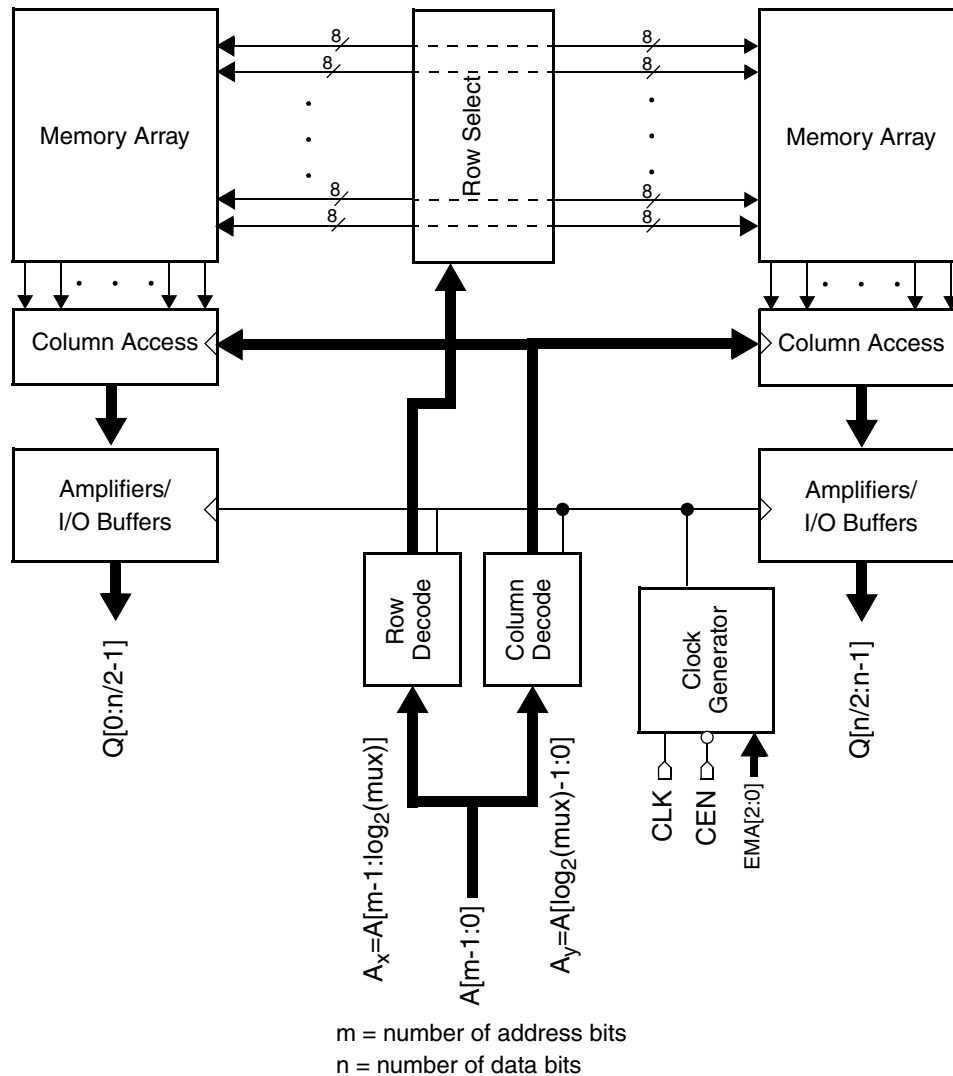
Table 3-9. ROM (rom_via_hde_hvt_rvt_hvt) Parameters

Input Parameters		
Parameter	Ranges	
number of words	MUX = 8	256 to 4096, increment = $\text{mux} \bullet 8$
	MUX = 16	512 to 8192, increment = $\text{mux} \bullet 8$
	MUX = 32	1024 to 16384, increment = $\text{mux} \bullet 8$
	MUX = 64	2048 to 32768, increment = $\text{mux} \bullet 8$
number of bits	MUX = 8	2 to 128, increment = 1
	MUX = 16	2 to 128, increment = 1
	MUX = 32	2 to 64, increment = 1
	MUX = 64	2 to 32, increment = 1
frequency	1 to $1/\text{t}_{\text{cyc}} \bullet 1000$, increment = 1	
top compiler metal layer	m5 - m9	
back biasing	on, off; default off	
power gating	on, off; default off	
extra margin adjustment	on	
Block Parameters		
Parameter	Ranges	
total memory bits	512 to 1,048,576 bits = words \bullet bits	
rows in memory matrix	256 to 32768; increment = 8; rows = word depth/MUX	
columns in memory matrix	MUX * 2 to 2048, increment = MUX; columns = word width * MUX	
address lines	MUX = 8	8 to 12
	MUX = 16	8 to 13
	MUX = 32	8 to 14
	MUX = 64	8 to 15

3.2.5 ROM Block and Core Address Diagrams (rom_via_hde_hvt_rvt_hvt)

The synchronous ROM instance block diagram is shown in Figures 3-3 and 3-4.

Figure 3-3. ROM Low Power/High Density (rom_via_hde_hvt_rvt_hvt) Basic Block Diagram

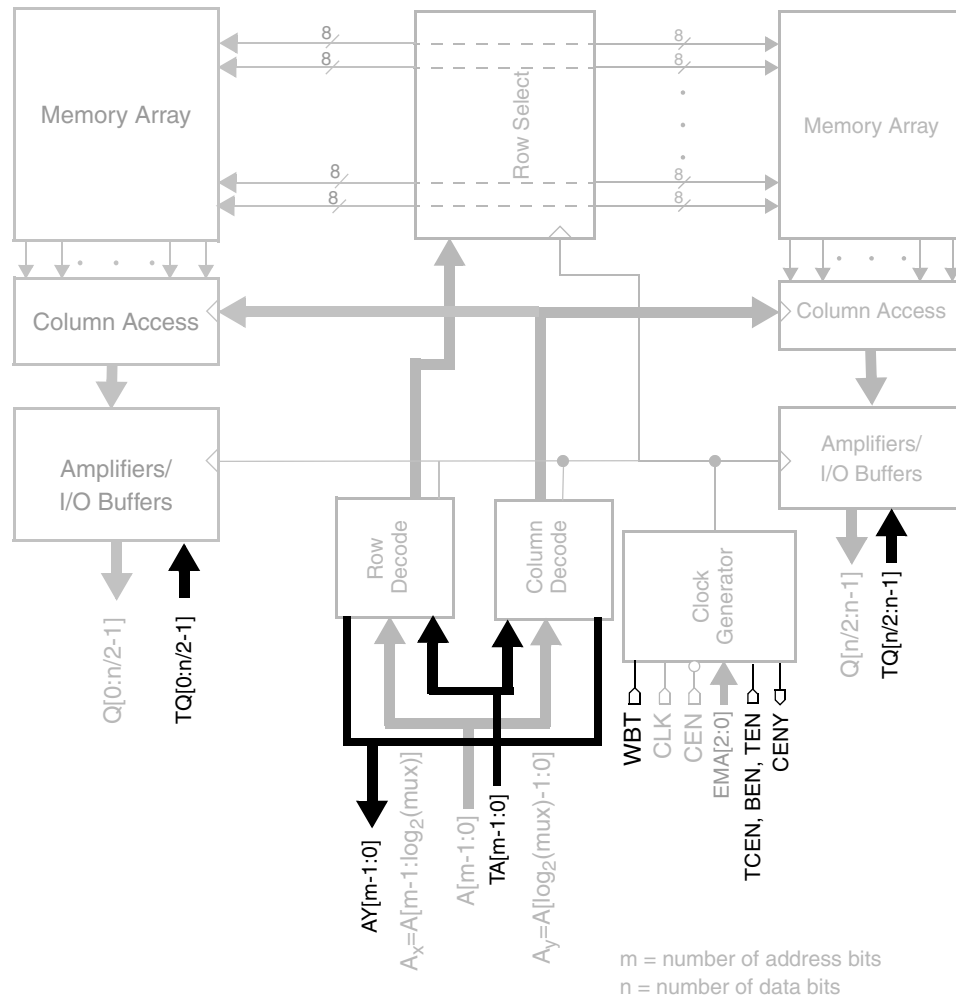


Notes:

EMA

The EMA option is always on; the EMA pin is an input bus signal.

Figure 3-4. ROM Low Power/High Density (rom_via_hde_hvt_rvt_hvt) Basic and Test Block Diagram



Notes:

EMA

The EMA option is always on; the EMA pin is an input bus signal.

BIST MUX

When the BIST MUX option is turned on, the TCEN, TEN, BEN, CENY, TQ, TA, and AY pins are available.

When the BIST MUX option is turned off, the TCEN, TEN, BEN, CENY, TQ, TA, and AY pins are unavailable.

3.2.5.1 ROM Core Address Maps (rom_via_hde_hvt_rvt_hvt)

An example of the standard physical core mapping for each MUX value is shown in Figures 3-5 through 3-9.

Figure 3-5. ROM (rom_via_hde_hvt_rvt_hvt) Mux 8: Address Mapping

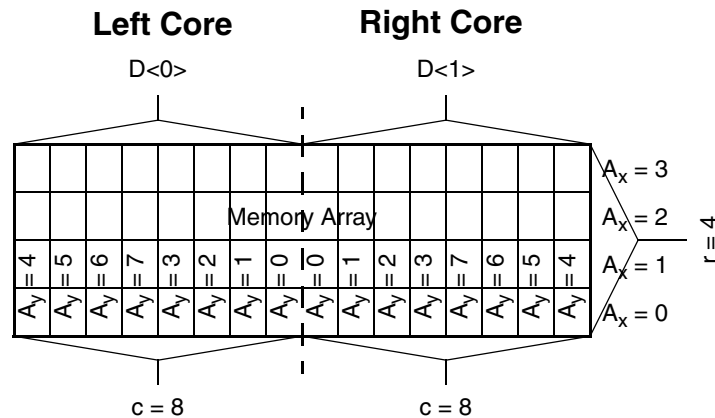


Figure 3-6. ROM (rom_via_hde_hvt_rvt_hvt) Mux 16: Address Mapping

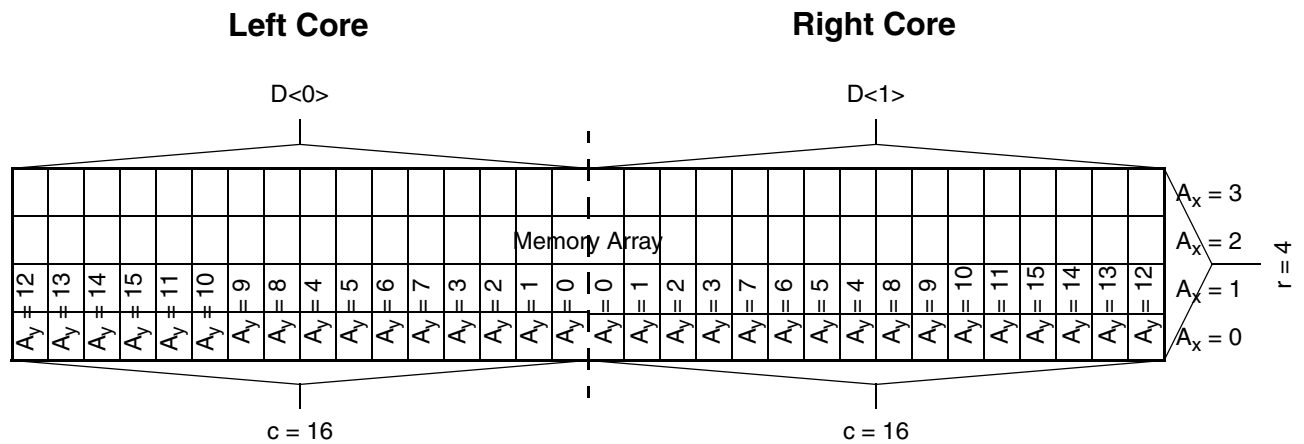


Figure 3-7. ROM (rom_via_hde_hvt_rvt_hvt) Mux 32: Core Address Mapping

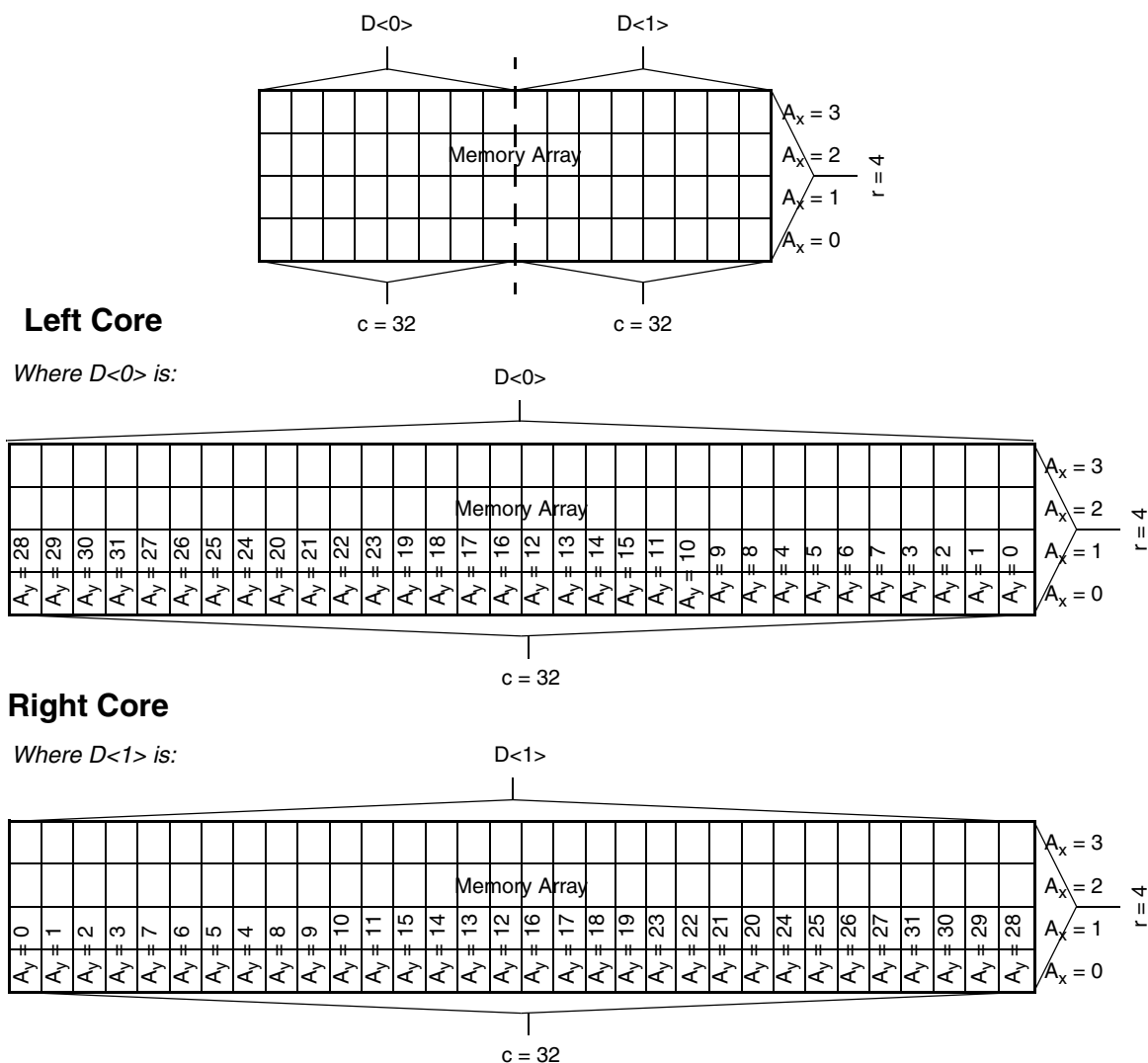


Figure 3-8. ROM (rom_via_hde_hvt_rvt_hvt) Mux 64: Left Core Address Mapping (D<0>)

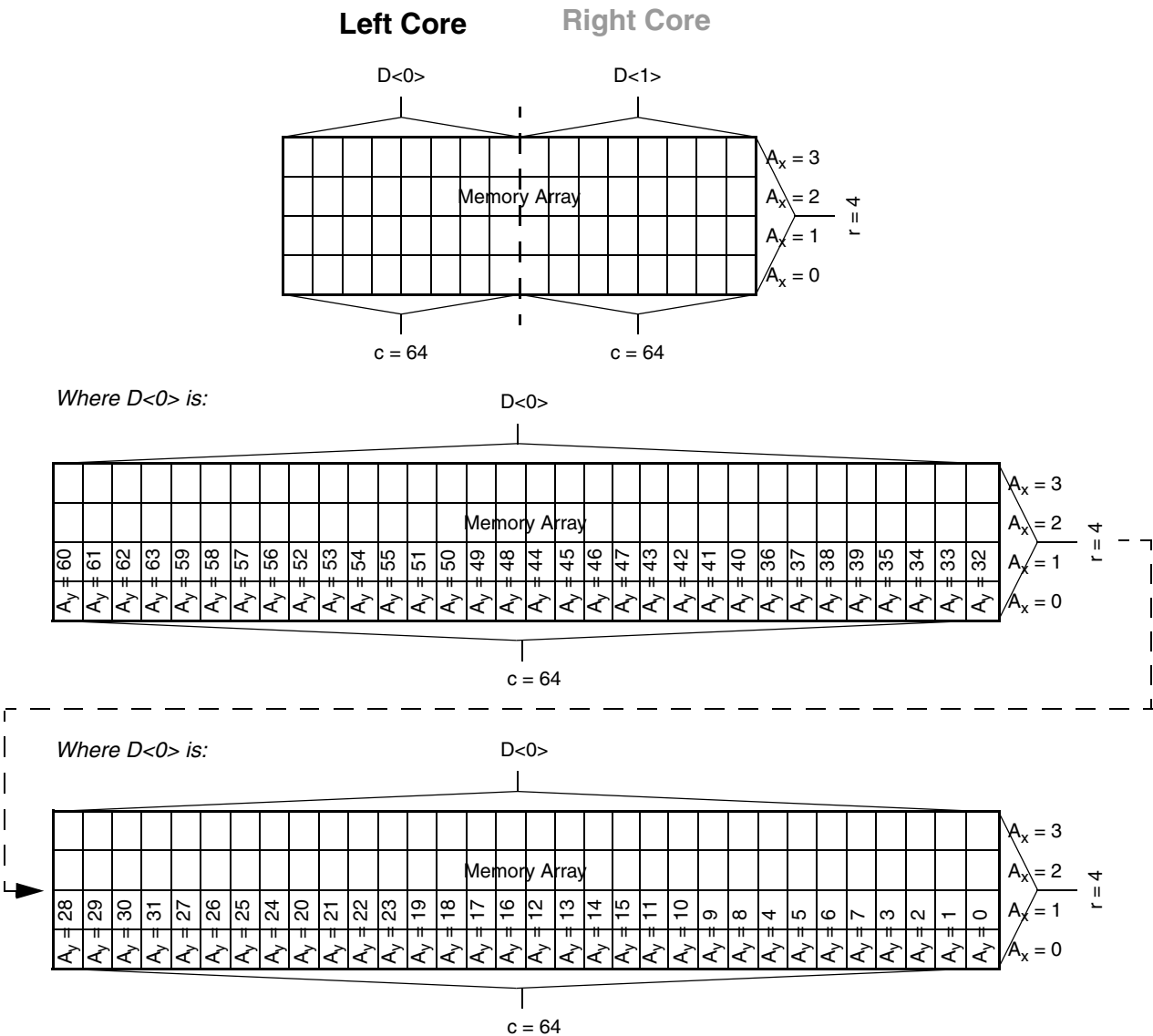
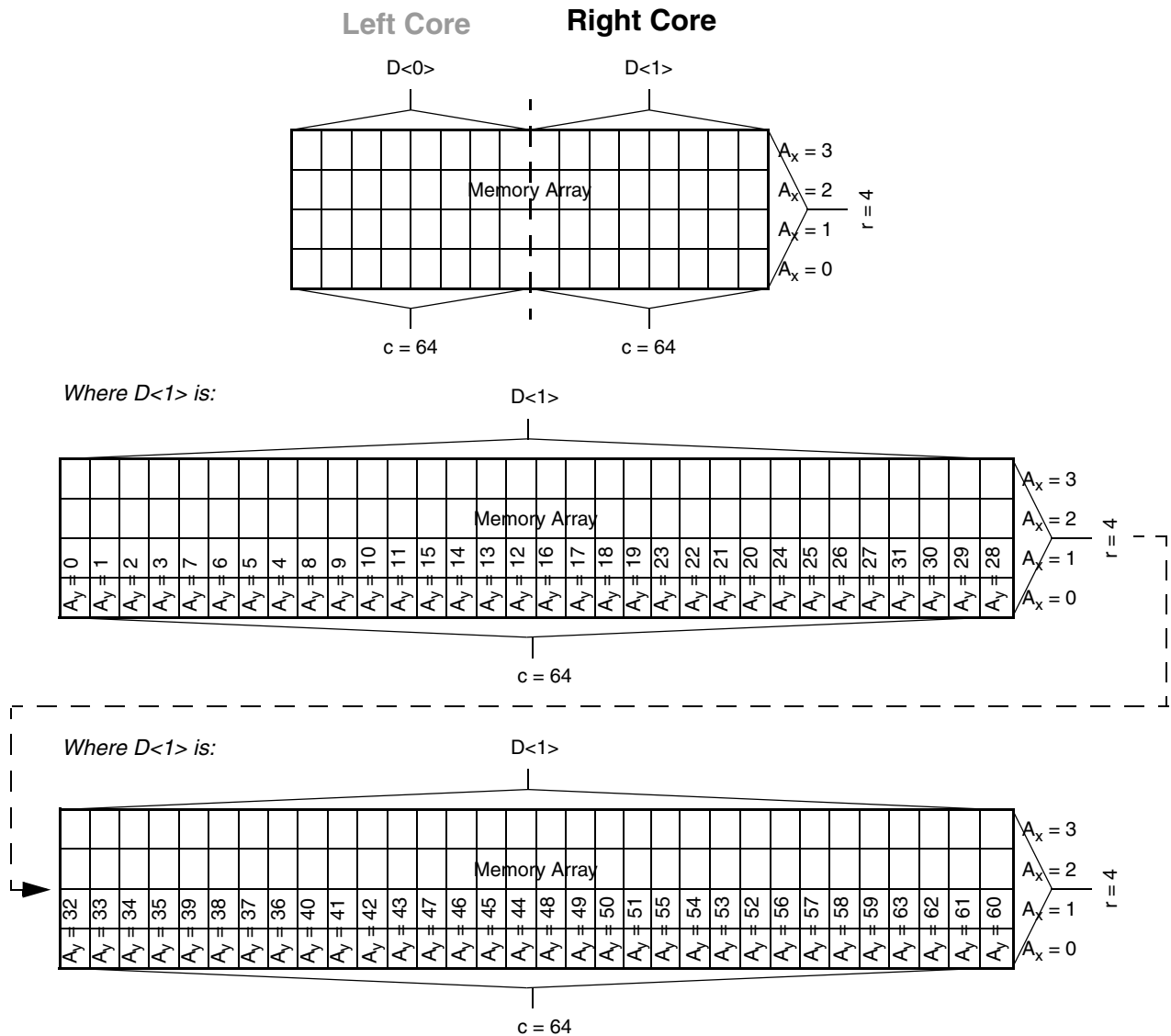


Figure 3-9. ROM (rom_via_hde_hvt_rvt_hvt) Mux 64: Right Core Address Mapping (D<1>)



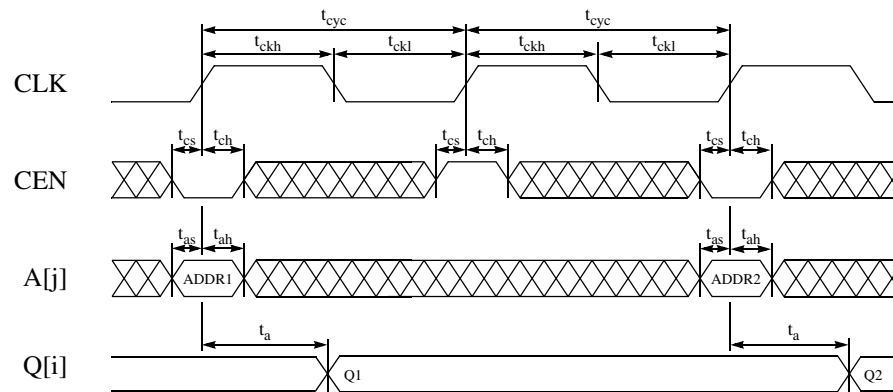
3.2.6 ROM Timing Specifications (rom_via_hde_hvt_rvt_hvt)

This section contains the timing diagrams, timing parameters, and power parameters of the synchronous ROMs.

3.2.6.1 ROM Timing Diagram (rom_via_hde_hvt_rvt_hvt)

The synchronous ROM timing diagram is shown in Figure 3-10. Standard rising/falling delays and slews are shown in this diagram. Some compilers may be designed with different percentages. You can refer to the postscript datasheet for your compiler to verify the delay and slew values.

Figure 3-10. Synchronous ROM (rom_via_hde_hvt_rvt_hvt) Read-Cycle Timing



Rising delays are measured at 50% VDD and falling delays are measured at 50% VDD.
Rising and falling slews are measured from 10% VDD to 90% VDD.

3.2.6.2 ROM Timing Parameters (rom_via_hde_hvt_rvt_hvt)

The ROM ASCII datatable contains the timing parameters listed in Table 3-10.

Table 3-10. ROM (rom_via_hde_hvt_rvt_hvt) Timing Parameters

Parameter	Symbol
Cycle time	t_{cyc}
Access time ^{1, 2}	t_a
Address setup	t_{as}
Address hold	t_{ah}
Chip enable setup	t_{cs}
Chip enable hold	t_{ch}
Clock high (minimum pulse width)	t_{ckh}
Clock low (minimum pulse width)	t_{ckl}
Clock rise slew (maximum transition time)	t_{ckr}
Output load factor (ns/pF)	K_{load}
Access time, eight numbers for eight values of EMA ^{1, 2}	$t_{a[0-7]}$
Cycle time, eight numbers for eight values of EMA	$t_{cyc[0-7]}$
Address setup, test pin	t_{tas}
Address hold, test pin	t_{tah}
Chip enable setup, test pin	t_{tcs}
Chip enable hold, test pin	t_{tch}
Test enable setup	t_{tens}
Test enable hold	t_{tenh}
Propagation delay BEN to output	t_{benq}
Extra margin enable pin setup	t_{emas}
Extra margin enable pin hold	t_{emah}
Load dependence factor on data output (ns/pF)	load_q
Load dependence factor on chip enable MUX output (ns/pF)	load_ceny
Load dependence factor on address MUX output (ns/pF)	load_ay

- ¹ The ASCII datatable and postscript datasheet show fixed delay values. These parameters have a load dependence (K_{load}), which is used to calculate: $TotalDelay = FixedDelay + (K_{load} \times C_{load})$, for timing views.
- ² Access time is defined as the slowest possible output transition for the typical and slow corners, and the fastest possible output transition for the fast corner.

Typical and slow timing models are generated with maximum delays, and the fast model is generated with minimum delays. ARM recommends that critical path, setup, and hold analysis be performed for all corners.

3.2.6.3 ROM Power Parameters (rom_via_hde_hvt_rvt_hvt)

The following table provides power parameters for standard compilers. For each of these parameters, the ASCII datatable provides values for each characterization corner, per instance. These values are also included if you generate a postscript datasheet for each instance.

Table 3-11. ROM (rom_via_hde_hvt_rvt_hvt) Power Parameters

Parameter	Symbol
AC Current ^{1, 4}	i_{cc}
Peak Current ⁴	i_{cc_peak}
Deselected Current ^{2, 4}	i_{cc_desel}
Standby Current ³	$i_{cc_standby}$
AC Current: eight numbers for eight values of EMA	$i_{cc}[0:7]$

¹ Value assumes 100% read operations, where all addresses and 50% of input and output pins switch. This value is equivalent to the read current (i_{cc_r}) value.

² Value assumes the memory is deselected, all addresses switch, and 50% of input pins switch. The logic-switching component of deselected power becomes negligibly small if the input pins are held stable by externally controlling these signals with chip select.

³ Value is independent of frequency and assumes all inputs and outputs are stable.

⁴ For most compilers, value shows dynamic current, without leakage (standby) component. Refer to the Power table in your compiler's postscript datasheet to determine if your compiler includes a DC leakage component

Current values shown in the datasheets and datatables are based on certain assumptions. You can refer to "ROM Current Parameters" on page 3-22 for information about current calculations.

Refer to "Noise Limits" on page 3-25 for more information related to power considerations.

3.3 ROM Power Structure (rom_via_hde_hvt_rvt_hvt)

The following sections describe the power structure options available with the ROM compiler.

3.3.1 ROM Current Parameters

The average current, I_{avg} , in mA, for the ROM instance is calculated from data reported in the GUI datatable, as well as the datasheet. The average current reported in the datasheet assumes 100% read operations, and no load on the output where all addresses and 50% of input and output pins [unloaded] switch.

Given:

c = average capacitance of output port (pF);

n = number of ports;

From the datatable,

$$I_{avg} = \left(icc + \left(\frac{1}{2} \cdot cvf \cdot bits \right) \right) \cdot n$$

From the datasheet,

$$I_{avg} = AC \text{ Current} + \left(\frac{1}{2} \cdot cvf \cdot bits \right) \cdot n.$$

The peak current, I_p , in mA, for the instance is reported in the GUI datatable, as well as the datasheet. This peak current is calculated during HSPICE simulations and reflects the maximum simulated value. The amplitude of the peak may be large, but the duration is very short due to ideal circuit behavior.

If the ROM is deselected and only the clock switches, then the current is the same as standby current. Standby current assumes no switching, and normal reverse-bias leakage.

———— **Note** ————

If inputs toggle, current can be 30-40% of icc because the input latches are open, and internal logic can switch. When the ROM is deselected, all addresses switch, and 50% of input pins switch, then current consumption may be up to 30-40% of icc because the input latches are open, and the internal logic can switch. The logic-switching component of deselected power becomes small if the address, data, and write enable pins are held stable by externally controlling these signals with chip select.

From the datatable,

$$I_p = icc_peak$$

From the datasheet,

$$I_p = \text{Peak Current}$$

3.3.2 Power Distribution Methodology

The chip-level power distribution must ensure that the wire widths supplied to the ROM satisfy electromigration guidelines and limit the average and peak voltage drop in the power wires to an acceptable value.

The ROM minimum supply wire widths are calculated as follows.

For example, given:

W_{em} = connection width based on electromigration (μm);

W_{iravg} = connection width based on average voltage (μm);

W_{irp} = connection width based on peak voltage (μm);

C = current density rule constant ($\text{mA}/\mu\text{m}$);

I_{avg} = average current consumed by the ROM (mA);

I_p = peak current consumed by the ROM (mA);

ΔV_{iravg} = allowable average voltage drop within the power wires on the chip (mV);

ΔV_{irp} = allowable peak voltage drop within the power wires on the chip (mV);

L_{eff} = effective wire length of power connection from power pad to the ROM (μm);

R_m = resistance of metal wire (Ohms/square);

W = connection width (μm);

we have:

$$W_{em} = \frac{I_{avg}}{C},$$

$$W_{iravg} = \frac{L_{eff} \cdot R_m}{\Delta V_{iravg}} \cdot I_{avg}$$

$$W_{irp} = \frac{L_{eff} \cdot R_m}{\Delta V_{irp}} \cdot I_p$$

$$W = \max(W_{em}, W_{iravg}, W_{irp})$$

Note

These sample calculations do not take into account the other components on the chip that may be supplied by the same wire. You must adjust wire width accordingly. The L_{eff} parameter can also be adjusted to account for the varying width of the power wires.

3.3.2.1 Noise Limits

The characterized clock noise limit is the maximum CLK voltage allowable for the indicated pulse width without causing a spurious memory cycle or other memory failure. For most compilers, the standard pulse width used in characterizing this limit is 10ns.

The power/ground noise limit is the maximum supply voltage transition allowable without causing a memory failure. Power and ground noise limits are assured at 10% of the characterized voltage.

3.3.2.2 ROM Code File

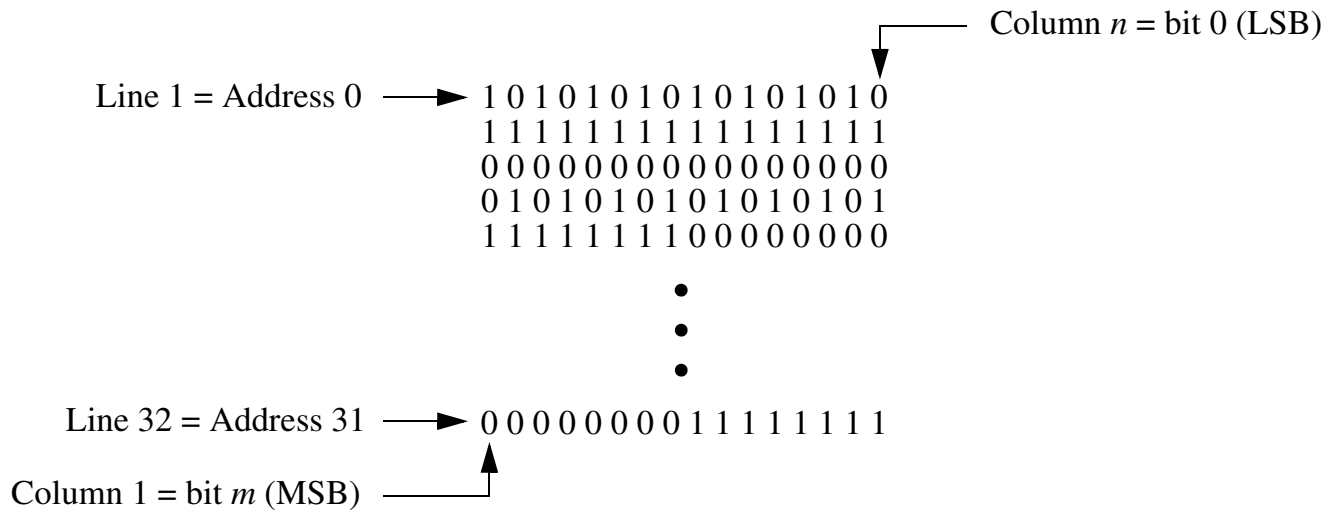
The ROM Code file defines the physical data of the ROM. An Artisan format ROM code file must be provided for each generated instance. The format has the following requirements:

- The ROM code file can contain only the characters 0 and 1
- The line number in the code file is equivalent to (address - 1) in the ROM instance
- Each character of a line corresponds to the bits of a word
- Column 1 is the most-significant bit (MSB) and column n is the least-significant bit (LSB)

Note

The address and data range from m (most-significant) to 0 (least significant). Lines and columns for a file range from n (most-significant) to 1 (least significant).

The following shows a ROM code file for a 32×16 ROM:



For Example:

Address 0 = 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
 Address 31 = 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
 Bit 0 (LSB) of Address 0 = 0
 Bit m (MSB) of Address 0 = 1

The ROM code file format is required for behavioral or physical views such as Verilog, Fastscan, GDSII, and LVS Netlist. When generating these views, the GUI checks for a valid ROM code file, and issues an error message if a ROM code file does not exist or is invalid.

If a view requires a ROM code file, the GUI creates a properly formatted, view-specific ROM code file for the corresponding view. For example, in the GUI, if the ROM code file is `source.rcf`, the instance name is `ROM2Kx16`, and a Verilog view is generated, then `ROM2Kx16.v` and `ROM2Kx16_verilog.rcf` is created with the proper Verilog format.

3.4 ArtiGrid Power Structure Options (rom_via_hde_hvt_rvt_hvt)

ArtiGrid provides the option of supplying power to a memory instance by connecting from the chip-level power grid. Power gating is available.

The user has to connect to all VDD and GND straps in order to provide sufficient current to the memory instance. Connecting wires must be properly sized and may require additional perpendicular straps to provide sufficient current to the memory instance.

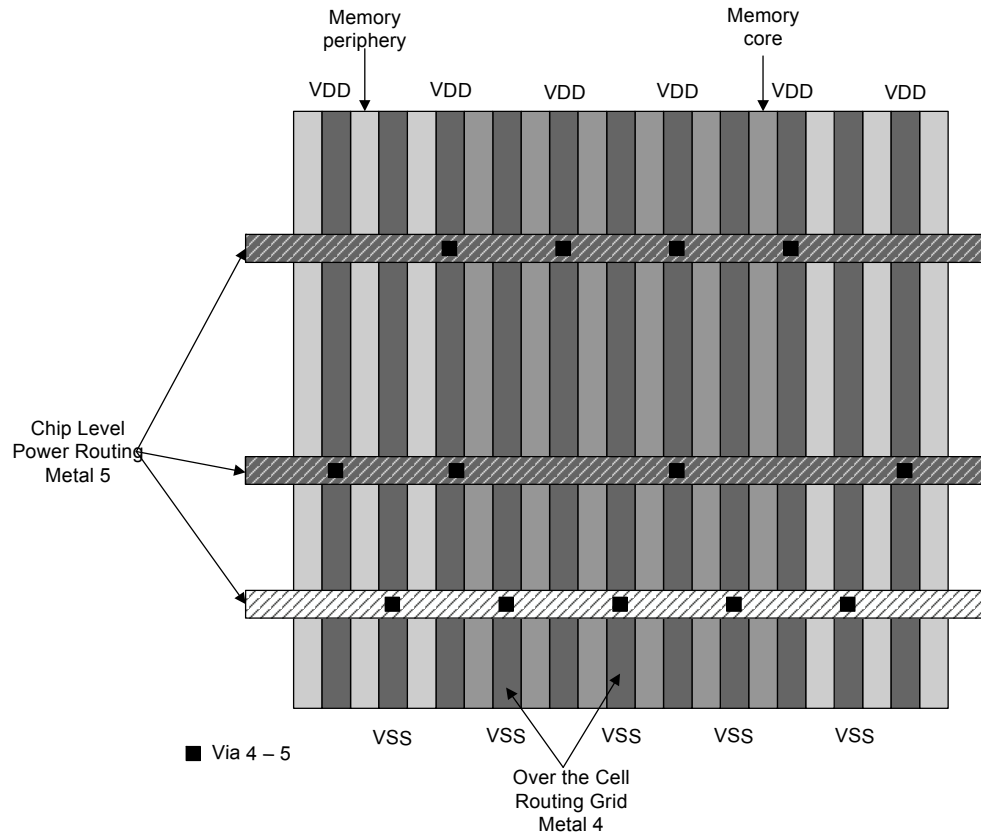
3.4.1 Power Supply Configurations

Standard option and power gating are the two different over-the-cell (OTC) power supply configurations supported by the memory compiler.

3.4.1.1 Standard Option

In this configuration, the memory instance can only be run with a common VDD for both core and periphery. See Figure 3-11. There is no retention mode but the memory instance can be run at standard voltage, low voltage, or shut down. The power pins available in this configuration are VDDE and VSSE. If back biasing support is enabled, then the VPW and VNW pins are brought out to the instance boundary.

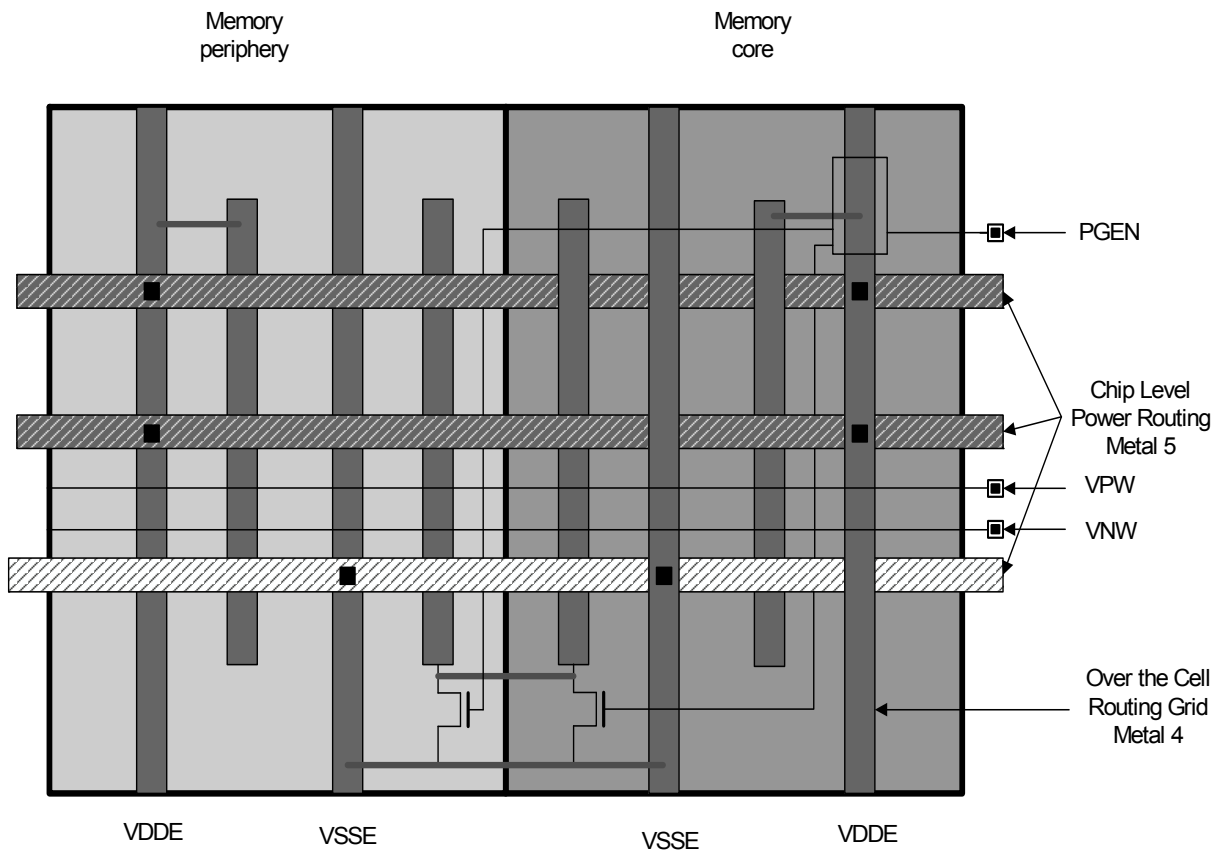
Figure 3-11. Over-the-Cell Power Routing - Standard Option



3.4.1.2 Power Gating Option

In this configuration, the VDDE and VSSE pins are seen at the instance boundary. See Figure 3-12. Along with these pins there is a PGEN pin available. When the PGEN pin is 'low', the power to both the core and periphery is isolated (power down mode). The power pins available in this configuration are VDDE and VSSE. If back biasing support is enabled, then the VPW and VNW pins are brought out to the instance boundary.

Figure 3-12. Over-the-Cell Power Routing - Power Gating Option



3.5 ROM Physical Characteristics (rom_via_hde_hvt_rvt_hvt)

This section provides physical characteristics of the ROM compiler, such as usage of top metal layers, I/O pin connections, ArtiGrid/Over-the-Cell (OTC) routing, and characterization environments.

3.5.1 Top Metal Layer

The compiler includes the capability of specifying the top-most metal layer utilized by the ROM. For example, a process may support a maximum of layers but you may elect to use only five layers for a given chip design. The layout size is the same for all top metal layer options.

All metal layers below and including metal4 are used in the design, and therefore, they are blocked. All metal layers above metal4 can be routed over the memory.

3.5.2 I/O Connections

Input/output (I/O) pins are located along the bottom edge of the memory block on any of the metal layers. The I/O pins are large enough to accommodate a pre-determined on-grid width wire connection. The pins are designed to be on the grid even when the memory is rotated or placed off the grid. Depending on the chip-level placement of a memory instance, a pin geometry may enclose multiple grid points but, as a worst case, only one is valid. A valid grid point is enclosed by the pin geometry by at least half of a wire width.

The router must access the pin by way of the routing track that corresponds to a valid grid point and is perpendicular to the cell edge. If the router approaches the pin off-grid and then bends the wire underneath the obstruction layer to connect to the valid grid point, a metal spacing or short circuit error may result.

3.5.3 Characterization Environments

By default, the compiler is characterized to fast (ff), typical (tt), and slow (ss) environments, or corners. Your compiler may have more corners. Only four corners are visible in the ASCII datatable of the compiler GUI at one time.

You can determine the characterization corners from the ASCII datatable in the compiler GUI. Move your mouse pointer (arrow) over the data columns to view the temperature and voltage corners for each column.

ARM recommends that critical path, setup, and hold analyses be performed for all applicable corners.

3.6 ROM Timing Derating (rom_via_hde_hvt_rvt_hvt)

Derating factors are coefficients that the characterization data is multiplied by to arrive at timing data that reflects different operating conditions. Standard memories do not support a timing derating methodology. By default, timing is provided for four characterization environments: two fast corners at different voltages and temperatures, typical, and slow. Some compilers may contain more environments.

Several delay calculators and the associated timing views, such as Synopsys, include a simplistic derating ability and a specified derating factor. The derating methodology supported by these delay calculators and the specified derating factor is not sufficient to accurately model the timing behavior of the memory. There is no derating for the models provided with these memories.

Relying on timing results using derating may lead to memory timing constraint violations and may cause a non-working part.

4

EDA Tools Support

This chapter contains the following sections:

- “Overview” on page 4-3
- “EDA Tool Support” on page 4-3
- “Using the Compiler Views” on page 4-4

4.1 Overview

This chapter lists the versions of tools used in designing the compiler and use of various tools to generate instances and views. These details apply to CLN65GP high density ROM compilers unless otherwise noted.

4.2 EDA Tool Support

The compilers produce standard views and models that have been verified with the tools defined in the applicable EDA Packages. See Table 4-1. As needed, refer to the README file in your compiler to determine the EDA or tool version(s) for your compiler.

Table 4-1. Tools Used for Verification

Deliverable and provided file name	Tool	Vendor
Verilog (.v)	NC-Verilog	Cadence
	VCS	Synopsys
	ModelSim (Verilog)	MTI/Mentor
Synopsys (Liberty) (.lib) ¹	Library Compiler	Synopsys
	RTL Compiler	Cadence
	SOC Encounter	
LEF (.vclef)	SOC Encounter	Cadence
TetraMAX (.tv)	TetraMAX Model	Synopsys
CDL	Calibre LVS	Mentor
	Hercules LVS	Synopsys
GDS II	Calibre LVS	Mentor
	Calibre DRC	Mentor
	Hercules LVS	Synopsys
CDB Enablement	CeltIC	Cadence
VoltageStormEnablement	VoltageStorm	Cadence

¹ Includes CCS noise data

4.3 Using the Compiler Views

This section provides information about simulating design modules that use views provided by the compilers.

4.3.1 Using the Verilog Model

Simulate the design module using these steps.

Check the syntax of the Verilog model:

```
vcs <name>.v
```

```
simv
```

where `<name>.v` is the Verilog model.

Use the SDF annotator to back-annotate the SDF timing files to the verilog model. An example command is:

```
$sdf_annotate(<sdf_file_name>, <instance>)
```

Run the simulation:

```
vcs <test-bench>.v
```

```
simv > verilog.log
```

The simulation output is written to a file, `verilog.log`, which is placed in the current directory.

4.3.2 Using the Synopsys (Liberty) Model to Generate SDF

You can generate SDF using the following steps.

Invoke the Synopsys tools:

```
dc_shell
```

Once inside `dc_shell`, execute the following Synopsys commands:

```
read_lib <name>.lib
write_lib <userlib>
link_library=<userlib>.db
target_library=<userlib>.db
read -f verilog <file>.v
write_timing -context verilog -f sdf-v2.1 -o <out>
```

where `<userlib>` is the name of your Synopsys library, `<name>.lib` is the Synopsys file, `<file>.v` is the top-level netlist, and `<out>` is the output file name.

If you are using multiple Synopsys libraries, you can read the `.lib` files into the Synopsys file and include each file in your `link_library` and `target_library` paths. You can write a script or manually remove the `lu_table_template` descriptions, `power_lut_template` descriptions, `type` descriptions, and `cell()` block for each `.lib` file. You should include all the descriptions and block into a single `.lib` file, and append the global library information found at the beginning of the generated `.lib` files to the beginning of your consolidated `.lib` file. Attach a closing bracket `"}"` to the end of the new `.lib` file.

All Synopsys models are generated with both maximum and minimum delays. ARM recommends that critical path, setup, and hold analysis be performed for all corners. In the Synopsys model, data from SPICE characterization is used for setup and hold times. In SDF, if the hold time is a negative number, it is set to zero.

——— **Note** ———

ARM recommends that you avoid using implicit netlisting because it is an error prone methodology.

4.3.3 Using the Synopsys (Liberty) Model to Generate SDF with Cadence Encounter

Use the following steps to generate SDF with Cadence Encounter.

Invoke the Cadence tool:

```
ets
```

Once inside `ets`, execute the following commands:

```
read_lib <name>.lib
read_verilog <file>.v
set_top_module <top_module>
write_sdf -splitsetuphold -min_period_edges none -version 2.1
<out>
exit
```

where `<name>.lib` is the Synopsys file, `<file>.v` is the top-level netlist, `<top_module>` is top level module in netlist and `<out>` is the output file name.

If you are using multiple Synopsys libraries, you can read the `.lib` files by including all of them in `read_lib` step between `{ }`.

All Synopsys models are generated with both maximum and minimum delays. ARM recommends that critical path, setup, and hold analysis be performed for all corners. In the Synopsys model, data from SPICE characterization is used for setup and hold times. In SDF, if the hold time is a negative number, it is set to zero.

4.3.4 Using the RTL Compiler for Timing Analysis.

Use the following steps to perform timing analysis.

1. Use the command `rc` to invoke the RTL compiler
2. Once inside the RTL compiler, execute the following commands where:
 - a. `<name>.lib` is the Synopsys file for memory
 - b. `<standard_cells>.lib` is the Synopsys file containing cells used in the netlist
 - c. `<file>.v` is the top-level netlist

```
# -> Read Target Libraries
set_attribute library {<name>.lib <standard_cells>.lib}
# -> Read HDL designs
read_hdl <file>.v
# -> Elaborate Design
elaborate
# -> Set timing Constraints
set_attribute force_wireload [find /libraries -wireload zerowlm] /
    designs/netlist_top
set_clock [define_clock -period 10000 -name CLK [clock_ports]]
set_attribute fixed_slew_fall 10 [find /designs -port ports_in/*]
set_attribute fixed_slew_rise 10 [find /designs -port ports_in/*]
set_attribute slew_fall 10 $clock
set_attribute slew_rise 10 $clock
set_attribute external_pin_cap <load_value> [find /designs -port
    ports_out/*]
# -> Verify timing constraints
report_timing -lint
report_timing -paths [specify_paths -to_rise_clock CLK]
# Check a design for undriven and multi-driven ports and pins
    unloaded sequential
# elements and ports, unresolved references, constant connected
    ports and pins and any
# assign statements using the following command
check_design -all
quit
```

If you use multiple Synopsys libraries, you can read the .lib files into the Synopsys file and include each file in your `link_library` and `target_library` paths. You can write a script or manually remove the `lu_table_template` descriptions, `power_lut_template` descriptions, type descriptions, and `cell()` block for each .lib file. You should include all the descriptions and block into a single .lib file and append the global library information found at the beginning of the generated .lib files to the beginning of your consolidated .lib file. Attach a closing bracket `"}`" to the end of the new .lib file.

All Synopsys models are generated with maximum and minimum delays. ARM recommends that critical path, setup, and hold analysis be performed for all corners. In the Synopsys model, data from SPICE characterization is used for setup and hold times. In SDF, if the hold time is a negative number, the hold time is set to zero.

4.3.5 Loading the VCLEF Description into SOC Encounter

Load VCLEF into SOC Encounter using the following steps.

Invoke SOC Encounter.

Bring up the “Design Import” window.

Perform either task “a” or “b” described below.

a. Enter the VCLEF filename and the LEF technology header filename along with other required inputs, such as Verilog netlist of the design.

b. Prepare a config file as shown in the following example and type

```
load config <config_file> 1
```

in the command window.

Example config file:

```
global_rda_Input
set rda_Input(ui_netlist) testRoute.v
set rda_Input(ui_netlisttype) {Verilog}
set rda_Input(ui_settop) {1}
set rda_Input(ui_topcell) {testRoute}
set rda_Input(ui_leffile) <tech>.lef <name>.vclef
set rda_Input(ui_pwrnet) {VDD}
set rda_Input(ui_gndnet) {VSS}
set rda_Input(ui_pg_connections) [list {PIN:VDD:} {PIN:VSS:}]
set rda_Input(PIN:VDD:) {VDD}
set rda_Input(PIN:VSS:) {VSS}
```

This creates the SOC Encounter database necessary for floor planning, placement, and routing; <tech> is the technology LEF and <name> is the memory instance name.

4.3.6 Using Astro with ARM Memory Instances

In order to use the ARM memory instances with the Synopsys Astro tool suite, you need to import the instance into the Milkyway database. Before you can place or route a design, you need to create a FRAM view for any memories in the design. This can be done by importing the VCLEF and running “read_lef” to create the FRAM. If you wish to stream out a full GDSII database you also need to import the GDSII into the Milkyway database to create a CEL view.

ARM does not recommend trying to create a FRAM view directly from the GDSII. You must use the following sequence when importing the instance into the Milkyway database to avoid creating a FRAM view from the GDSII.

1. Generate the VCLEF and GDSII
2. Import the VCLEF into Milkyway by running “read_lef” to generate a FRAM view
3. Import the GDSII into Milkyway

It is important to run “read_lef” before importing the GDSII.

Details on creating and importing VCLEF and GDSII are provided in the following sections. Consult the Synopsys documentation for more details on the commands mentioned below. In particular, you should be familiar with the Synopsys *Milkyway Data Preparation User Guide*.

4.3.6.1 Loading the VCLEF Description into the Milkyway Database

This action uses a file called `<name>.vclef`. The instance name for this file is `<name>`.

Start Milkyway. On the command line for Milkyway, use the Synopsys LEF reader `read_lef` or, from the menu, select `Cell Library > Lef In...`

A form is displayed. Fill this form with the appropriate entries for library name and .lef file name, then click on OK to create the CEL view and FRAM view.

4.3.6.2 Loading the GDSII Layout into the Milkyway Database

This action uses a file called `<name>.gds2`. The instance name for this file is `<name>`.

Create a file named `gds2Arcs.map`.

Example:

```
gdsMacroCell
<name>
```

Start Milkyway. The VCLEF should have already been read into the library created in the previous section. Read in the GDSII. This process overwrites the CEL view with the actual layout. On the command line, use `auStreamIn` or, from the menus, select `Cell Library > Stream in...` You need to update the tech file to support all memory `gds2` layers. Fill in the `Stream File Name` and the `Library Name`. Depending on your flow, the other fields may or may not need to be updated.

4.3.7 Loading the GDSII Layout into a DFII Library

You can load the GDSII layout into a DFII library using these steps.

Invoke DFII.

From the DFII CIW, select the *File* pull-down menu, then select the *Import* pull-down sub menu and click on *Stream*.

In the *Stream* pop-up window, type the name of the GDSII layout file in the *Input File* field, and type the instance name in the *Top Cell Name* field. Type the name of the library in the *Library Name* field. Click on *User-Defined Data*.

In the *User-Defined Data* pop-up window, type the path to the metal layer table file in the *Layer Map Table* field, and type the path to the text font file in the *Text Font Table* field.

4.3.8 Using the LVS Netlist

The LVS netlist may be used in conjunction with the GDSII file for verification.

Typically, you use a tool like Cadence Assura, Mentor Graphics Calibre, or Synopsys Hercules, to read a GDSII file and compare it with the LVS netlist. This test compares the layout and schematic to ensure there is no short- or open-circuit in the layout.

The LVS netlist is then added onto the chip level LVS netlist, and the same test is run when the chip is fully assembled. This process ensures that the chip is correctly assembled (that is, there is no short- or open-circuit caused by a place-and-route or other tool).

4.3.8.1 Using Hierarchical LVS

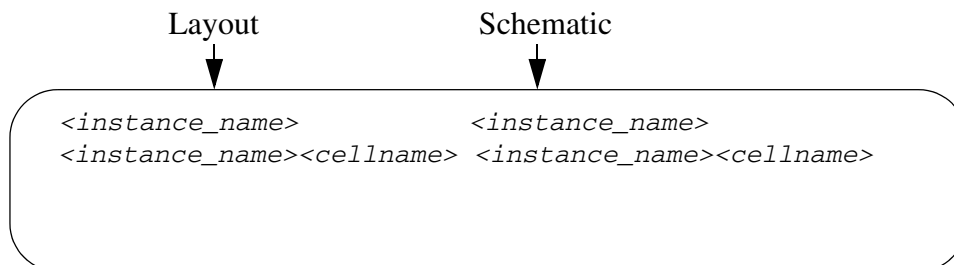
LVS (Layout vs. Schematic) performs an equivalence check between two different representations of a design. In this case, the physical (GDSII) and schematic (SPICE netlist) are compared to each other. The Calibre tool reports any discrepancies.

Executing LVS in a hierarchical mode is faster than using LVS in the flat mode. The blocks are checked only once, as opposed to multiple passes for various instantiations of the same block in the design.

To run Calibre LVS on memory instances in full hierarchical mode, execute the following steps:

1. Invoke the compiler GUI.
2. Generate the GDSII and LVS netlists to create the `<instance_name>.gds2` and `<instance_name>.cdl`.
3. Create the `.hcell` file.

Sample Hcell file format:



The left column corresponds to the layout instances and the right column corresponds to schematic instances where:

`<instance_name>` is the instance name you specified when the `.gds2` and `.cdl` views were created.

`<cellname>` is a hierarchical cell within the memory

4. Modify the rules file.

The rules file specifies the location and format of the following items:

```
LAYOUT PATH "<instname>.gds2"
LAYOUT PRIMARY <instname>
SOURCE PATH "<instname>.cdl"
SOURCE PRIMARY <instname>
LVS REPORT "<instname>.lvs"
```

5. Execute Calibre:

```
calibre -lvs -hier -spice <instname>.spc
        -hcell <instname>.hcell rulesfile
```

where the `.spc` file is output from calibre.

6. Examine the LVS report.

Troubleshooting Notes:

More information about hierarchical LVS can be found in Mentor Graphics' *Physical Extraction and Verification Application Note #21: What to look for in the Calibre LVS transcript*.

Specifically, review the section on the command "LVS SHOW SEED PROMOTIONS YES"

This information is available online at the Mentor Graphics web site.

4.3.9 Using the CeltIC Enablement Tool

CeltIC models for memories are created with the `make_cdb` utility from Cadence. The main design inputs are the CDL netlist and the SPICE mapping file. Details on these input files and the template script that are used to run `make_cdb` are explained in the following sections.

For the discussions below, it is assumed that the memory instance name (which is user supplied) is "MyMemInstName".

4.3.9.1 Design Inputs

4.3.9.1.1 CDL Netlist

This netlist is generated as part of the views that are generated by the compiler. The netlist name is "MyMemInstName.cdl". To make it compatible with spice models, we have to modify the model.

Diodes have different syntax between CDL and SPICE.

CDL style: D10 VSSE CLK tdndsx 1.024e-07U 1.28U

SPICE style: XD10 VSSE CLK tdndsx Area=1.024e-07U Perim=1.28U (X at the beginning is needed because of subckt call)

Transistors need X in the beginning because models are subckt based.

CDL style: M16 AI5N AI5 VSSE VSSE nfet W=1.12U L=0.06U M=1

SPICE style: XM16 AI5N AI5 VSSE VSSE nfet W=1.12U L=0.06U M=1

The `celtic.pl` script as shown in the following sections converts CDL to SPICE.

Run the following commands on the "MyMemInstName.cdl":

```
./celtic.pl "MyMemInstName.cdl" "MyMemInstName_mod.cdl"

mv "MyMemInstName_mod.cdl" "MyMemInstName.cdl"
```

4.3.9.1.2 SPICE Mapping File

This file is supplied as part of the compiler installation. The path to the hspice models and temperature and corner information is contained in `main.spi`.

4.3.9.2 Required Scripts and Files

4.3.9.2.1 main.spi

```
* title

.lib <path>/hspice/models tt

.temp 0
```

The path to the spice model files is denoted by `<path>`.

4.3.9.2.2 celtic.pl

Use this file for modification of cdl netlist

```
#!/usr/bin/perl

if($#ARGV < 1) {
    &showSyntax();
    exit(-1);
}

$inp_file = $ARGV[0];
print $inp_file;
shift;
$out_file = $ARGV[0];
shift;
```

```
#####
# Description : Show usage syntax
#####
sub showSyntax() {
    print "Command line: $0 <input>.cdl <output>.cdl\n";
    exit(-1);
}

open (INP_F, "< $inp_file") || die "Could not open
<input>.cdl";
open(OUT_F, "> $out_file") || die "FATAL : Fail to open file,
Exit.\n";
$i=0;
@inp_f = <INP_F>;
close(INP_F);

while ($i<=$#inp_f) {
#For M to XM
    if ($inp_f[$i] =~ /^M/) {
        $out_f= "X$inp_f[$i]";
        print OUT_F "$out_f";
    } elsif ($inp_f[$i] =~ /^D/) {
#For D to XD and adding Area and Perim
        $j=0;
        out_f= "X$inp_f[$i]";
        @varr = split(" ", $out_f);
        while ($j<=$#varr) {
            if(@varr[$j] =~ /[0-9][.][0-9]*e/) {
                @varr[$j] = "Area=@varr[$j]";
            } elsif(@varr[$j] =~ /[0-9][.][0-9]*/) {
                @varr[$j] = "Perim=@varr[$j]";
            }
        }
    }
}
```

```
    print OUT_F "@varr[$j] ";
    $j++;
  }
  print OUT_F "\n";
} else {
    $out_f= "$inp_f[$i]";
    print OUT_F "$out_f";
}
$i++;
}
close(OUT_F);
exit(1);
```

4.3.9.3 Writing the TCL Script for CeltIC

The following script may be copied and pasted as needed.

```
# Assumptions:
# Generated Memory Instance Name: "MyMemInstName"

# Pl specify the correct spice mapping file which
# contains the path to spice models
set spice_map_file "main.spi"

# Specify your memory netlists file here
set memory_netlist_files { MyMemInstName.cdl }

set_parm max_patterns 1000000

set_parm port_vdd_r 1
set_parm port_gnd_r 1
#
set_supply -vdd 1.52 -gnd 0.0
#
message_handler -set_msg_level ALL
```

```
# Now we use 'generate_cell_lib' to generate the
# noise library for the cells (in this case MyMemInstName)
# into the output file MyMemInstName.cdb

# Read .lib file for getting port directions
read_dotlib MyMemoryInstName.lib

#Put the correct Power Supply names related to the instance
#Please refer to the following comments in MyMemInstName.cdl
# Configuration: n -left_bus_delim "[" -right_bus_delim "]" -pwr_gn\
# Configuration: d_rename "VDDCE:VDDCE,VDDPE:VDDPE,VSSE:VSSE" -
pref\

generate_cell_lib \
  -vdd { VDDCE VDDPE \
} \
  -gnd { VSSE\
} \
  -cell_list MyMemInstName \
  -file_list "$spice_map_file $memory_netlist_files" \
  -file MyMemInstName.cdb \
  -text -ccc_print_large 100
#
validate_cell_lib -cdb { MyMemInstName.cdb }
```

4.3.9.4 Generating CeltIC Model (CDB)

Run the following command to generate the CeltIC model:

```
Load tool cadence/ets/7.1

make_cdb -64 <above tcl script>
```

4.3.9.5 Validation

Currently, only the ability to read the generated CDB file in the CeltIC tool without errors is assured. Only known warnings displayed during CDB generation that are cleared by Cadence Expert are ignored.

The following command used during the CDB view generation also covers the validation:

```
validate_cell_lib -cdb { MyMemInstName.cdb }
```

Results of the validation step are provided at the end of the log file.

4.3.9.5.1 Known Warnings/Actions

a. Warning: (SI-4594). Cannot find man pages for this product. You will not be able to find such pages.

Action: Ignore

b. Warning: Identified more than one `leakage_power` groups with same condition in cell (i_0). Last definition will be retained. <TECHLIB-359>.

Action: Ignore; this version of the CeltIC tool will generate this warning due to certain limitations and the leakage data has no influence on generated CeltIC data.

c. Warning: (SI-2054) The transistor I0.I2.I0.I0.I59.I9.M0 has the source net I0.TL0_0 connected to the drain. This indicates that the transistor is functioning as capacitive load. If this connection is not correct, check the netlist for errors. [load_spice]

Action: Ignore. The tool is detecting some transistors with drain and source connected to form a capacitive load. For RAM, it is common to balance the bit lines with transistor caps.

d. Warning: (SI-4543) Cell "i_0" contains 10518 transistors, only peripheral transistors will be processed. [generate_cell_lib].

Action: Ignore. Reports the names of all channel-connected components (CCCs) that have more than the specified number of transistors.

e. You may get some warning related to foundry models with CeltIC. Consult the foundry and/or Cadence to get a waiver and/or get the fix. For example some model warnings may be:

Warning: (SI-2197)<spice_model_path>/.../hspice/./model_files/./dgnfet.inc:219: SPICE card is not supported and will be ignored. If this SPICE card is required, correct any errors or replace it with an equivalent supported SPICE card. If the SPICE card is not required, comment it out. [make_cdb]

Warning: (SI-2157) The value of parameter NOIA in model srpdbnfet has a number larger than maximum supported value of 1e+37. This number will be reset to the maximum supported value

4.3.10 Using VoltageStorm Flow

The following sections describe the processes and files needed to generate the VoltageStorm view. Any process needs to be executed by the customer. The VoltageStorm view must be enenerated for each corner.

4.3.10.1 Compiler Generated Files

Generate GDS file and VCLEF files for a given instance.

4.3.10.2 Foundry Supplied Files

- XTC command file
- Extraction Tech file (qrcTechFile)
- ZX command file; this contains information such as temperature and special handling needs.
- Spectre models

4.3.10.3 ARM Supplied Files

A variety of standard cell LEF files are available in the arm_tech release for you to choose from.

4.3.10.4 Libgen Command File

This file contains the commands to be executed by the libgen tool. The file is described in Section 4.3.11.4.3.

4.3.11 VoltageStorm Flow Setup and Run

The main memory directory has four sub-directories that contain all the instances. These sub-directories are data, tech, libgen, and common.

The data and libgen directories contain instance specific information. For example, data/<instance>.gds2, data/<instance>.vclef, libgen/<instance>.cmd.

The VoltageStorm views are generated under common/<instance>.cl directory

4.3.11.1 Directory Files

4.3.11.1.1 data

- GDS file for different instances from the compiler
- VCLEF file for different instances from the compiler
- Technology LEF information the ARM-provided “ARM Routing Technology Kit

4.3.11.1.2 tech

- Extraction tech file (qrcTechFile) from the foundry
- Spectre SPICE models from the foundry

4.3.11.1.3 libgen

- Libgen command file: <instance>.cmd

4.3.11.1.4 common (Foundry Supplied Files)

- Layer map for LEF and GDS
- XTC command file

- ZX command file
- Thunder command file for temperature. User generated, thunder.cmd. See Section 4.3.11.4.2

4.3.11.2 Flow Run Tools

- VST: ANLS 7.1
- ULTRASIM: MMSIM 6.2
- QRC: EXT 7.1

4.3.11.3 Running the Flow

- Run inside the 'common' directory
- The following command runs the VoltageStorm view generation for instance i_0_0: libgen
../libgen/i_0_0.cmd

4.3.11.4 Command Files

All the command files are temperature and voltage specific for each corner. Temperature and voltage information is noted in *italics* in the following sections.

4.3.11.4.1 ZX Command File

```
# File: zx.cmd
#layer map P_SOURCE_DRAIN N_SOURCE_DRAIN
setvar layout_scale 0.9
#Add this temp information if the foundry supplied file is missing
this
setvar temperature 125
```

4.3.11.4.2 Thunder Command File

```
// File: thunder.cmd
// Should have the following line to specify the temperature
setenv SpectreOptions "temp=125"
```

4.3.11.4.3 Libgen Command File Template for <instance>

```
# File: libgen.cmd
# Library Name
setvar library_name <instance>

# Powerview Generation for cell
detailed_powerview_cell_list { <instance> }
cell_list { <instance> }

# Flow Type
input_type pr_lef

# Output Library Name

# Lef Files
lef_file_list { ../data/*.lef ../data/<instance>.vclef }

# GDS Files
gds_file_list { ../data/<instance>.gds2.gz }

# Extraction Tech File
setvar tech_file ../tech/qrcTechFile

# Extraction Include File (ZX)
setvar parasitic_extractor_command_file ../common/zx.cmd

# Thunder/Spectre cmd file for temperature
setvar thunder_command_file ../common/thunder.cmd

# View Generation
setvar generate_port_powerview true
setvar generate_detailed_powerview true
#setvar generate_reduced_powerview true
```

```
setvar assume_foreigns true
setvar generate_collapsed_powerview true

# PowerPins
generic_power_names 1.0 { VDD VDDE }
generic_ground_names 0 { VSSE }

# Layer Map
include ../common/lefdef.layermap
include ../common/gds.layermap

# Xtc/Calibre/PVS command file
setvar gds_extractor_command_file ../common/xtc.cmd

# Spice Models: on-the-fly tables using Spectre-SKI
setenv spice_model_file ../tech/*.scs { tt tt_hvt tt_lvt }
setvar default_frequency 100e+06
```

4.3.11.5 Verification

For verification, run the following commands inside the ‘common’ directory:

- `libgen -report <instance>.cl` and view the report for the information on the cell.
- `libgen -s <instance>.cl` for short summary on the cell data.
- `psviewer` will open gui, then open the cell library `<instance>.cl` and load the `<instance>` to view in the VoltageStorm GUI.

Appendix A- Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1. Issue A

Change	Location	Affects
No change, first release	-	-

