High Density 65nm CLN65GP SRAM Compiler User Guide

Revision: r0p0

Confidential

December 2009

ARM PUG 0086A1a

ARM PUG 0086A1a

Proprietary Notice

Confidentiality Status

Table of Contents

Preface

Revision History

The following table provides the revision history for this manual.

Part Number	Updates (to template)
ARM PUG 0086A1a	• 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`).
<italic type> *italic type*	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.
Italic Type with Initial Capital Letters	Document, chapter, section and reference manual names.

1
Overview and Installation

This chapter contains the following sections:

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 SRAM compilers to create memory instances based on a variety of parameters, and the corresponding EDA tool support views. This manual provides information about single- and dual-port SRAM compilers.

Note

Parameters or specifications for your compiler may differ from the default high density compilers. Information about deviations from 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.

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

- This chapter provides basic information about high density SRAM compilers 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 SRAM Compiler Architecture,” **-** Lists the architectural details, physical characteristics of memory instances, and characterization/timing information.
- Chapter 4, “Compiler Views,” **-** Provides information about EDA tools support and provides instructions on generating specific views.

1.2 High Density SRAM Compiler Features

High density memory compilers include the following features:

- Aspect Ratio Control for Efficient Floor Planning
- Memory Operation and Retention at Low Frequency (Down to 0 MHz)
- Low Active Power and Leakage-Only Standby Power
- Timing and Power Models for Industry-Leading Design Tools
- Configurable Word-Write Mask Option
- Extra Margin Adjustment™ (EMA) Option
- Integrated BIST MUX Option
- Soft Error Repair (SER) Option
- Flex-Repair™ Redundancy Option
- ArtiGrid™ Over-the-Cell Power Routing Option
- Maximum Static Power Dissipation Corner
- Back Biasing
- Power Gating
- Advanced Test Feature (Weak Bit Test and Read Disturb Test)
- Dual Voltage Support

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

Standard and optional tool support is listed below.

Standard Support	**Optional Support**
Verilog	FastScan TetraMAX
Flex-Repair & SER Verilog	
Synopsys Liberty	
VCLEF Footprint	
GDSII Layout LVS netlist	
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
```

where <*install_files*> represents the compiler installation files in your installation directory.

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.
`corners/`	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.

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.

Compiler	Product Name	Executable
High Density Single-Port SRAM	sram_sp_hdc_svt_rvt_hvt	sram_sp_hdc_svt_rvt_hvt
High Density Dual-Port SRAM	sram_dp_hdc_svt_rvt_hvt	sram_dp_hdc_svt_rvt_hvt

2
Using the Compiler

This chapter contains the following sections:

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.

Note

Not all features and capabilities may apply to your specific compiler.

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

The ARM 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 instance with Verilog view, mux = 8, words = 256, type:

```
sram_sp_hdc_svt_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 2-16 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 2-13, "Generating Views from the Command Line" on page 2-16, and "Compiler Options" on page 2-21 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 2-1. Views and Output Files

Standard Support		
View	**Output Files**	**Note(s)**
Verilog model	*<instance_name>*.v	
Flex-Repair & SER Verilog	*<instance_name>*rtl.v	1
Synopsys (Liberty) for each corner	*<instance_name>*_<*corner*>_syn.lib	2, 3, 4
VCLEF footprint	*<instance_name>*.vclef *<instance_name>*_ant.lef *<instance_name>*_ant.clf	
GDSII layout file	*<instance_name>*.gds2	
Optional Support		
View	**Output Files**	
Mentor FastScan	*<instance_name>*.fastscan	
Synopsys (Liberty) TetraMAX	*<instance_name>*.tv	5
Datasheets		
View	**Output Files**	
PostScript datasheet	*<instance_name>*.ps	
ASCII datatable	*<instance_name>*.dat	

1 The word-write mask option should be set to ‘off’ when SER is selected as either ‘1db1bc’ or ‘2bd1bc.’

2 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, ff_1p10v_1p10v_125c, ff_1p10v_1p10v_0c, ff_1p10v_1p10v_m40c, tt_1p00v_1p00v_25c, ss_0p90v_0p90v_m40c, and ss_0p90v_0p90v_125c) is inserted into the output filename (for example, sram_sp_hdc_svt_rvt_hvt_<corner>_syn.lib).

[3] 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.

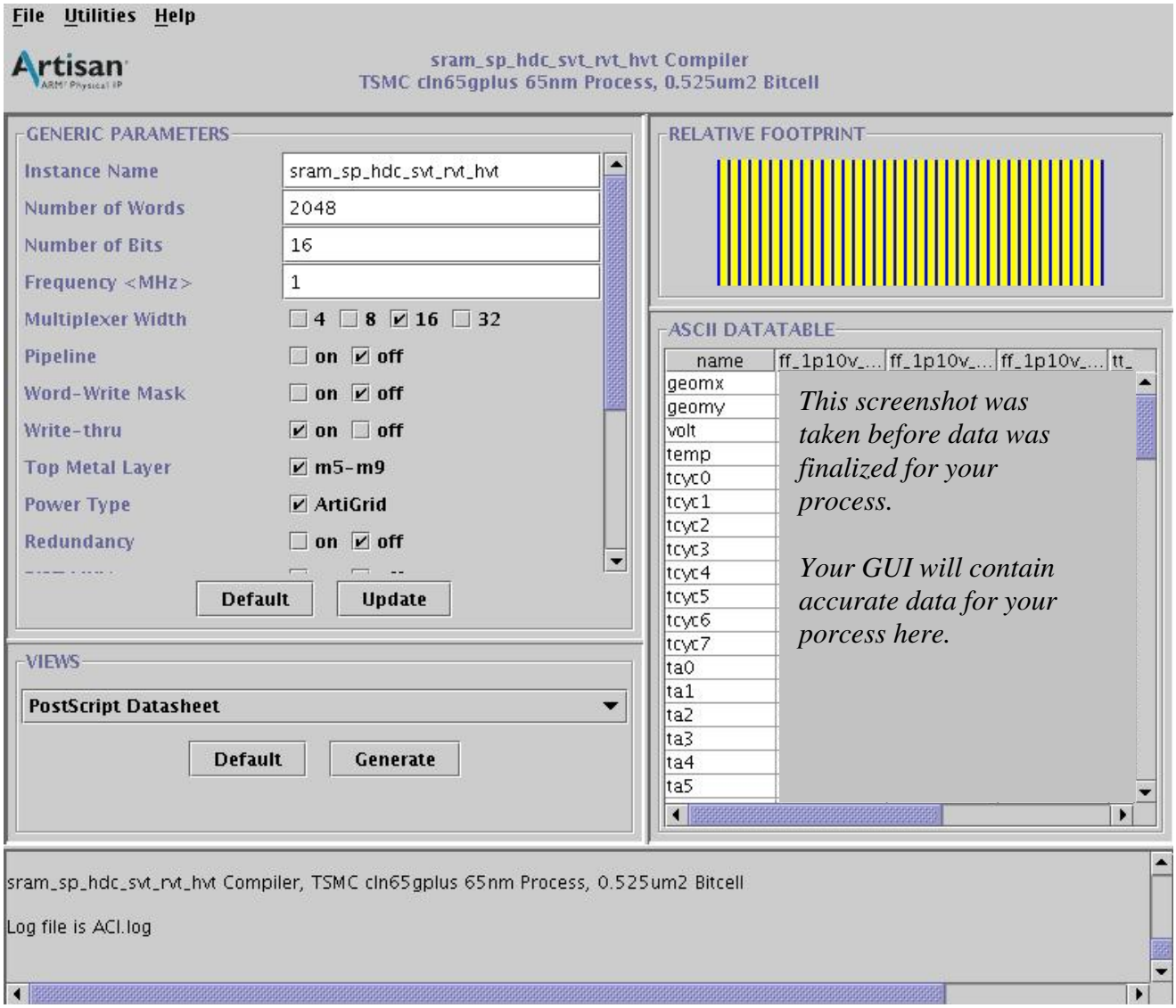
[4] Synopsys models are generated with maximum alternate current (AC) values for each supported corner. Depending on chip design, overall chip level worst case power conditions can occur under the fast corner (PVT conditions) or under the “Maximum Static Power” corner condition. The worst case static 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.

[5] 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 SRAM 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: SRAM 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. 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 also generate a single view at a time from the *Views* pane in the compiler GUI. Be aware that your Views Pane options may differ from examples in this guide. 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 2-1 "Views and Output Files" on page 2-5. For detailed information about using these views refer to Chapter 4, "Compiler Views".

You can generate multiple views at one time. Refer to "Generating Multiple Views" on page 2-14. You can also 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 SRAM changes as the words, bits, and MUX parameters are varied. Refer to Figure 2-1 "Example: SRAM 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. The instance without a power ring is shown in darker color. The power ring is shown in lighter color.

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 compiler GUI. Information about minor 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.

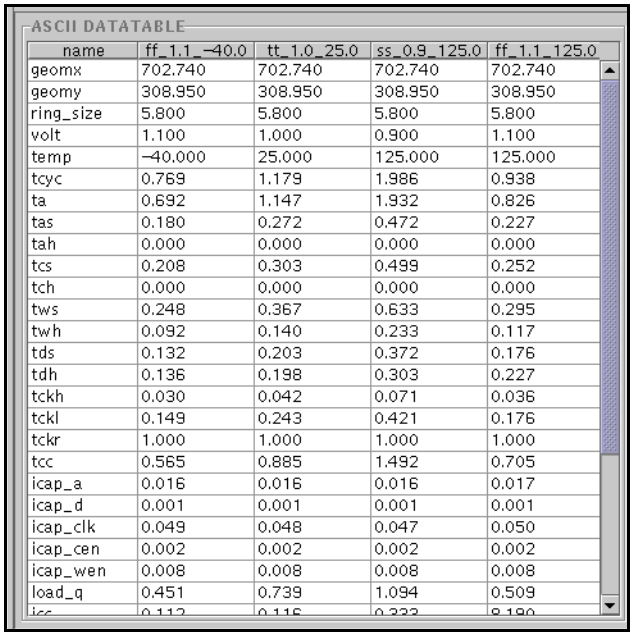
In the ASCII datable and postscript data sheets, non-power values are displayed in decimal format. Current/power values in datatables and datasheets may be shown in scientific notation or in decimal format.

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.

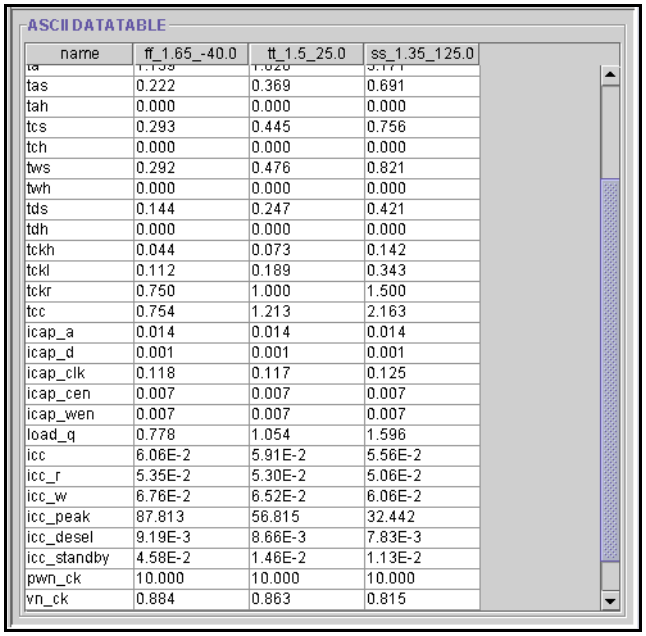
Figure 2-2. Example: ASCII Datatable Pane



ASCII DATATABLE

name	ff_1.1_-40.0	tt_1.0_25.0	ss_0.9_125.0	ff_1.1_125.0
geomx	702.740	702.740	702.740	702.740
geomy	308.950	308.950	308.950	308.950
ring_size	5.800	5.800	5.800	5.800
volt	1.100	1.000	0.900	1.100
temp	-40.000	25.000	125.000	125.000
tcyc	0.769	1.179	1.986	0.938
ta	0.692	1.147	1.932	0.826
tas	0.180	0.272	0.472	0.227
tah	0.000	0.000	0.000	0.000
tcs	0.208	0.303	0.499	0.252
tch	0.000	0.000	0.000	0.000
tws	0.248	0.367	0.633	0.295
twh	0.092	0.140	0.233	0.117
tds	0.132	0.203	0.372	0.176
tdh	0.136	0.198	0.303	0.227
tckh	0.030	0.042	0.071	0.036
tckl	0.149	0.243	0.421	0.176
tckr	1.000	1.000	1.000	1.000
tcc	0.565	0.885	1.492	0.705
icap_a	0.016	0.016	0.016	0.017
icap_d	0.001	0.001	0.001	0.001
icap_clk	0.049	0.048	0.047	0.050
icap_cen	0.002	0.002	0.002	0.002
icap_wen	0.008	0.008	0.008	0.008
load_q	0.451	0.739	1.094	0.509
icc	0.112	0.116	0.333	8.190

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



ASCII DATATABLE

name	ff_1.65_-40.0	tt_1.5_25.0	ss_1.35_125.0
ta	1.139	1.820	3.171
tas	0.222	0.369	0.691
tah	0.000	0.000	0.000
tcs	0.293	0.445	0.756
tch	0.000	0.000	0.000
tws	0.292	0.476	0.821
twh	0.000	0.000	0.000
tds	0.144	0.247	0.421
tdh	0.000	0.000	0.000
tckh	0.044	0.073	0.142
tckl	0.112	0.189	0.343
tckr	0.750	1.000	1.500
tcc	0.754	1.213	2.163
icap_a	0.014	0.014	0.014
icap_d	0.001	0.001	0.001
icap_clk	0.118	0.117	0.125
icap_cen	0.007	0.007	0.007
icap_wen	0.007	0.007	0.007
load_q	0.778	1.054	1.596
icc	6.06E-2	5.91E-2	5.56E-2
icc_r	5.35E-2	5.30E-2	5.06E-2
icc_w	6.76E-2	6.52E-2	6.06E-2
icc_peak	87.813	56.815	32.442
icc_desel	9.19E-3	8.66E-3	7.83E-3
icc_standby	4.58E-2	1.46E-2	1.13E-2
pwn_ck	10.000	10.000	10.000
vn_ck	0.884	0.863	0.815

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 2-2.

Table 2-2. ASCII Datatable Units

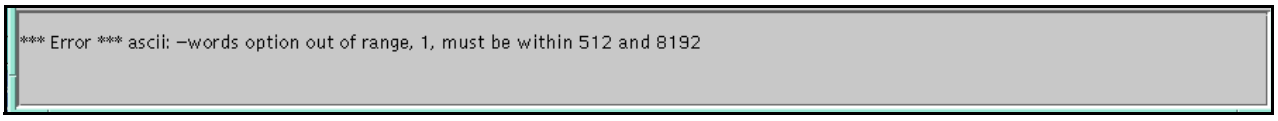
Parameters	Units
Geometry	Microns
Current-consumption	Milliamperes
Timing	Nanoseconds

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.

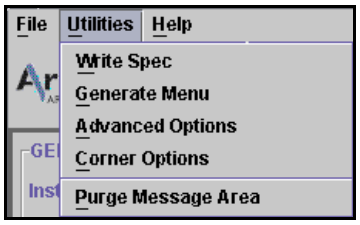
2.3.6 File Menu (Exiting the GUI)

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



Note

Corner options may not be available in your GUI.

Refer to "Using Specification Files" on page 2-18, "Generating Multiple Views" on page 2-14, "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 the 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.

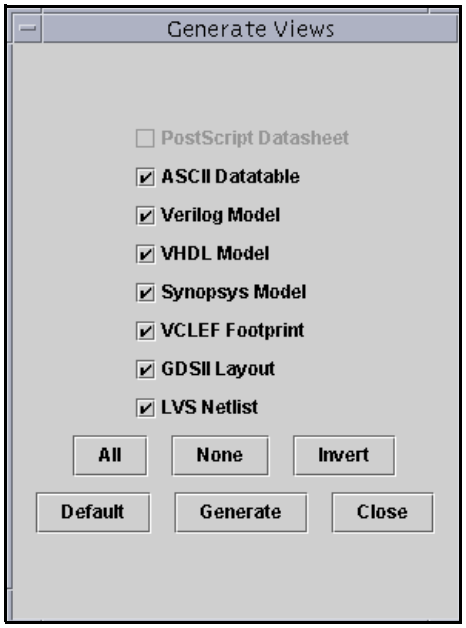
For instance, if you want to create datasheets for multiple instances, you can start by changing the generic parameters shown in the Generic Parameters section or in the Advanced Options pull-down menu of the GUI. Click on Update in the GUI. Select “Postscript Datasheet” from the Views pull-down menu 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 2-12. 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 2-30 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, "Compiler Views" 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 2-3. View Commands

Standard Support	
View	**View-Command**
Verilog model	verilog
Flex-Repair & SER Verilog	verilog_rtl
Synopsys (Liberty)	synopsys
VCLEF footprint	vclef-fp
GDSII layout file	gds2
Optional Support	
View	**View-Command**
Mentor FastScan	fastscan
Synopsys TetraMAX	tmax
Datasheets	
View	**View-Command**
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 2-21 for details about writing commands to run the compiler correctly.

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

For example:

```
% sram_sp_hdc_svt_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:

```
% sram_sp_hdc_svt_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 instance, the `inst2ring` and `site_def` options only apply 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` and `-inst2ring` options, and supply a view-specific value for each option, type:

```
% ra1shsynopsys vclef-fp -synopsys.libname <syn_userlib>
-vclef-fp.diodes on
```

For low-power single-port SRAM memories, replace `ra1sh` with `ra1sl` in the statement above.

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

Commands (in sequence)	Example
%<*executable*>	%sram_sp_hdc_svt_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
write_mask=off
```

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 2-21.

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 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 Generator
*              Artisan xxnm Process
* version:     2008Q3V1
* comment:     080922_6:38_03
* configuration: -instname INSTANCExxx -words 4096 -bits 16
-frequency 1 -ring_width 2 -mux 16 -drive 6 -write_mask off
-wp_size 8 -top_layer met8 -cust_comment "080922_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 “080922_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*>] [-write_mask *on*|*off*]…

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 2-18.

`-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 2-16 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 `ra1sh`“ascii `-help`" to view the help information related to the ASCII datatable, such as compiler parameters. For low-power single-port SRAM memories, replace `ra1sh` with `ra1sl` in the statement.

`-instname` *name*

The default instance name is the same as the executable name, in capital letters. For instance, if the executable is sram_sp_hdc_svt_rvt_hvt, the default instance name is sram_sp_hdc_svt_rvt_hvt. Refer to "Directory Structure and Executables" on page 1-8, regarding the executable name for your compiler. For low-power single-port SRAM memories, replace `ra1sh` with `ra1sl` in the statement.

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.

`-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.

`-write_mask` *on/off*

The compiler GUI shows the default value for the Word-Write Mask option and any choices for this option. When it is selected, the companion option, Word Partition Size, is displayed in the GUI.

`-write_thru` *on/off*

This option allows for data latching. When write-thru (also referred to as write-through) is selected on (enabled), data to be written [D(i)] can appear on the output pins [Q(i)] if the BWEN(i) pin is set accordingly. When write-thru is set to off (disabled), output data remains latched to the last read value. The default value is off.

-wp_size *number*

The default value is 8. The range for word partition size is 1 to min (36, bits-1) increment = 1.

-top_layer *number*

The compiler GUI shows the default value for the top metal layer and any choices for this option.

-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" on page 3-92.

-power_type *ArtiGrid*

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

-redundancy *on|off*

When Flex-Repair or redundancy is turned on, there are zero or two columns and zero, two, or four rows of integrated redundancy available. See Figure 2-9.

Figure 2-9. Redundancy Options

The compiler can also create associated RTL, which sets memory pins needed to access the redundant locations, depending on external inputs. For detailed instructions on how to use the Flex-Repair option, refer to the “High Density Memory Compiler Test and Repair Features Application Note (Change title to reflect update).”

`-rcols` *0|2*

This option is enabled when Redundancy is set to *on* in the generic parameters pane. You can select the number of redundant columns through the `-rcols` option. The default is 0, when redundancy is on. See the “High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note” for more details.

`-rrows` *0|2|4*

This option is enabled when Redundancy is set to *on* in the generic parameters pane. You can select the number of redundant rows through the `-rrows` option. The default is 0, when redundancy is on. See the “High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note” for more details.

`-bmux` *on/off*

When this option is turned on, the compiler adds MUX at most input and output pins. The MUX can be used by BIST engines to access the memory in test mode. The default value is off. For more details, see the “High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note”

`-ser` *none/1bd1bc/2bd1bc*

This option allows you to store error correcting codes by adding extra bits to each word of a memory. Along with the repair RTL created by the compiler, this option prevents soft-errors due to external particles such as neutron or alpha particles.

By default, this option is disabled (set to “none”). You can set it to 1bd1bc to enable one bit error detection and one bit error correction. Setting it to 2bd1bc, provides two bits of error detection and one bit of error correction. This option is only available for memory instances when the word-write mask option is off (deselected).

`-back_biasing` *on|off*

This option allows the you to select (on) the back bias pins in order to reduce leakage. When back biasing is selected on, 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 'on.' The default value is off.

`-atf` *on|off*

When enabled, the advanced test features (ATF) option invokes both weak bit test (WBT) and read disturb test (RDT).

The WBT feature detects a weak bit in a core array and is asserted high (1) when ATF is selected.

The RDT option allows certain weak bit cells to fail during a read operation and helps find marginal cells that might otherwise pass a conventional test.

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.

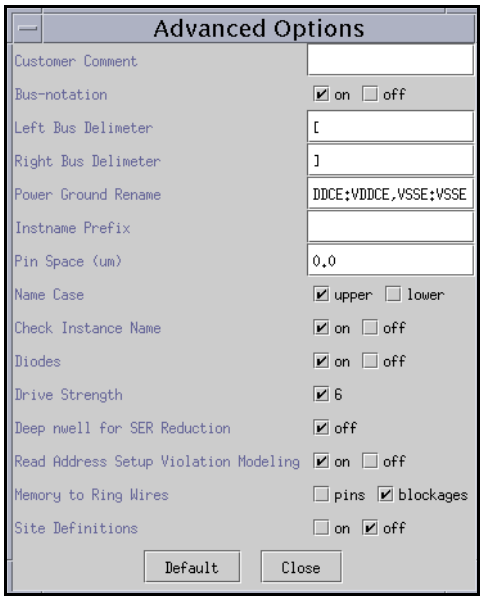
2.7.3.1 Setting Advanced Options from the GUI

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 2-10. Enter a string or number in the fields, or select the check boxes as needed.

Note

The number of advanced options may vary and depends on your specific compiler. On some compilers, you may see Dual Port Clock Collision Modeling (DPCCM) and Read Address Setup Violation Modeling (ASVM) options or just a single option for ASVM.

Figure 2-10. 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 2-21 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 2-20.

`-bus_notation` *on*

By default, all bus pins are represented by bus notation in the interface of models (for example, 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."

`-name_case` *upper|lower*

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

`-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`.

`-se_immunity` *on|off*

This option allows you to add deep NWELL implants and/or p-taps in the memory, providing better immunity to soft errors caused by external particles such as neutrons or alpha particles. The default value is off.

`-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.

`-dpccm` *on|off*

This option enables you to select front-end (FE) model behavior, for dual-port compilers, when clock collisions occur during Read and Write operations. Refer to your compiler GUI for the default setting for your compiler.

During clock collision, if `dpccm` is `off`, Read operation fails and the output lines show X. Also, Write operation fails and the memory location and write-through (if applicable) show X.
If `dpccm` is `on` during clock collision, Read operation fails and the output lines show X, but in this case Write operation to the memory location succeeds.

`-asvm` *on|off*

This option enables you to select front-end (FE) model behavior when address setup violations occur during a Read operation. This option allows you to choose memory FE model behavior when address setup violations occur. Refer to your compiler GUI for the default setting for your compiler.

During an address setup violation, if `asvm` is `off` during Read operation, Read fails, and the output lines show X and are invalidated (X-ed-out) at all memory locations.
If `asvm` is `on` during Read operation, Read fails and the output lines show X, but in this case all memory locations preserve their state.

2.7.4.1 Advanced View-Specific Options

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

`-libname` *userlib*

If you are using the Synopsys model, the default library name is *userlib*. Use this option to specify your choice for `-libname`. This option applies only to the Synopsys model.

`-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.

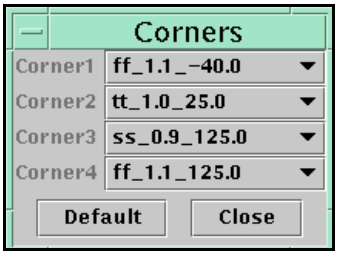
2.7.5 Selecting Characterization Corners

You can set characterization corners from the compiler GUI or by entering the option on the command line. ARM recommends that critical path, setup, and hold analyses be performed for all applicable corners.

2.7.5.1 Selecting Corners from the GUI

From the Utilities pull-down menu in the compiler GUI, select the "Corners Option." The Corners dialog box contains several corner options; Corner1, Corner2, and Corner3, and so on. The number of corner options may vary per compiler. Figure 2-11 shows a Corners dialog box with four fields.

Figure 2-11. Example: Corners Dialog Box



Note

Corner options may not be available in your GUI.

Each field contains predetermined PVT selections. The first item in each selection is the process indicator, such as fast (ff), typical (tt), and slow (ss). There may be more than one fast corner, each with differing combinations of voltages and temperatures. After the process indicator, the first value is the voltage for that corner. The second value is the temperature for that corner. In the example above, for the first selection in Corner 1, the process is fast (ff), the first value is 1.1V, and the second value is -40^{o}C.

2.7.5.2 Maximum Static Power Dissipation Corner

The maximum static power dissipation corner feature uses the maximum power value and minimum timing value.

2.7.5.3 Selecting Corners from the Command Line

`-corners` “string, string, string, {string}”

You can specify up to four PVT corners for characterization at one time. For more information about PVT corners, see "Characterization Environments" on page 3-93. All timing models, PostScript datasheet, and ASCII datatable have delays set at the chosen corners. Timing model filenames have a corner name embedded in them.

3
Synchronous SRAM Compiler Architecture

This chapter contains the following sections:

3.1 Overview

This chapter describes the architecture, features, timing characterization, and physical characteristics for synchronous single- and dual-port SRAM compilers.

Note

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

Where applicable, separate sections are provided for single- and dual-port SRAM compilers. The following table lists the compiler names, product names, and executable names for the compilers described in this section. Check your compiler GUI; the names provided in your compiler GUI always supercede those in the table below.

Table 3-1. SRAM Compiler Naming Conventions

Compiler	Product Name	Executable
High Density Single-Port SRAM	sram_sp_hdc_svt_rvt_hvt	sram_sp_hdc_svt_rvt_hvt
High Density Dual port sram	sram_dp_hdc_svt_rvt_hvt	sram_dp_hdc_svt_rvt_hvt

3.2 Synchronous Single-Port SRAM Architecture and Timing Specifications

This section describes the synchronous single-port SRAM compiler architecture, which includes pin descriptions, logic tables, 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 Single-Port SRAM Description

Descriptions for the basic functionality of this compiler are provided. In addition, descriptions of features you can use to enable the test and repair functions of your compiler are provided. You can obtain further repair and test details and scenarios from the “High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note.”

3.2.1.1 Basic Functionality

The synchronous single-port SRAM is produced by a parameterized block compiler which allows great flexibility in the SRAM organization.

If the word-write mask feature is not implemented, via the Compiler, a write cycle is initiated if both the write enable, WEN, and the chip enable, CEN, are asserted at the rising-edge of CLK. Input data, D, is written at the address, A.

If the word-write feature is implemented, data on the data input bus is partitioned via the compiler to the write enable bus, WEN[p:0]. Each WEN pin has a distinct latched value, making each partition individually selectable. When the latched value of a write enable pin, WEN[i], is low the corresponding data partition is selected, and its data is written to the memory location specified on the address bus. The written data is available at the output bus. In addition to the WEN bus, there is also a global write enable pin (GWEN). If any of the WEN[i] signals is low, the GWEN signal must also be low to enable a write operation.

A read cycle is initiated in the RAM if CEN is asserted and WEN is de-asserted at the rising-edge of the clock, CLK. The contents of the RAM location specified by the address, A, are driven on the data output bus, Q. The SRAM is allowed to access non-existing physical addresses, but the outputs will be unknown.

The read address for any given memory cycle can be identical to the write address of the previous memory cycle with the read data being identical to the data that was written from the previous memory write cycle.

A standby mode is provided for periods of non-operation (CEN=1). While in standby mode, address and data inputs are disabled; data stored in the memory is retained, but the memory cannot be accessed for reads or writes.

Memory is self initializing on Power Test and Repair Functionality

3.2.1.2 Test and Repair Functionality

The SRAM compiler includes features such as Flex-Repair/Redundancy, Soft Error Repair (SER), Built-In Self Test (BIST) MUX, and Extra Margin Adjustment (EMA), as described in this section. Feature options are described in the "Compiler Options" section on page 2-21. You can also obtain examples of how this functionality can be implemented in the "High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note."

3.2.1.2.1 Flex-Repair/Redundancy

Redundancy is a user-selectable option. A 4 row/16 column redundancy design is described. Any redundant row may replace any other row. Redundant columns are divided into two groups of 8. One group can be used for the lower half of the bits in the word and the other group used for the upper half of the bits in each word. The columns reuse the lower 3 bits of the address (A[2:0]) across all MUX values. The user chooses a subset of the repair capability and the repair and reconfigure RTL ensures that the user sees the appropriate number of redundant rows (0, 2, or 4) and columns (0 or 2). When redundancy is selected, BIST MUX are always delivered. However, BIST MUX selection can be obtained without redundancy.

Row Redundancy

Row redundancy is based on 4 independently addressable rows in the bottom banks (bank0 and bank1) that can be used to repair defective rows in any of the banks. For a 512Kb RA1 memory, redundancy consists of 4 spare rows placed at the bottom of the memory. When four redundant rows are added into an instance, new signals are added. These are FRA1[(n-1): 0], FRA2[(n-1): 0], FRA3[(n-1): 0], FRA4[(n-1): 0], and RRE1, RRE2, RRE3, RRE4.

Redundant rows are accessed if the regular address matches with any one set of FRAi[(n-1): 0] and the corresponding RREi is enabled. For 512Kb memory, FRAi is from 1 to 4, and RREi is also from 1 to 4. The maximum number of address bits for row address for 512Kb is ‘n.’

Column Redundancy

Column redundancy is implemented in blocks of 8 columns. Redundant columns are divided into two groups of 8, one of which can be used for the lower half of the bits in the word and the other for the upper half of the bits in each word. The spare columns reuse the lower 3 bits of the address (A[2:0]) across all MUX values. The column redundancy has the following characteristics:

- Two redundant column blocks will be used, one for each half of the word.
- This works as follows for each MUX value

  MUX 4: Full bit redundancy for both the lower and upper halves of each word. CREN signal is static.

  MUX 8: Full bit redundancy for both the lower and upper halves of each word. CREN signal is static.

  MUX 16: Bit redundancy for half the address space for the lower and upper halves of each word independently.

  MUX 32: Bit redundancy for one quarter of the address space for the lower and upper halves of each word independently.

  Various configurations are possible, but in no case can more than a single bit be replaced in each of the upper and lower halves of any given word.

- CREN operation changes from previous memories, because of the ability to control redundant bit shifting in the upper and lower half of the bits in a word independently.

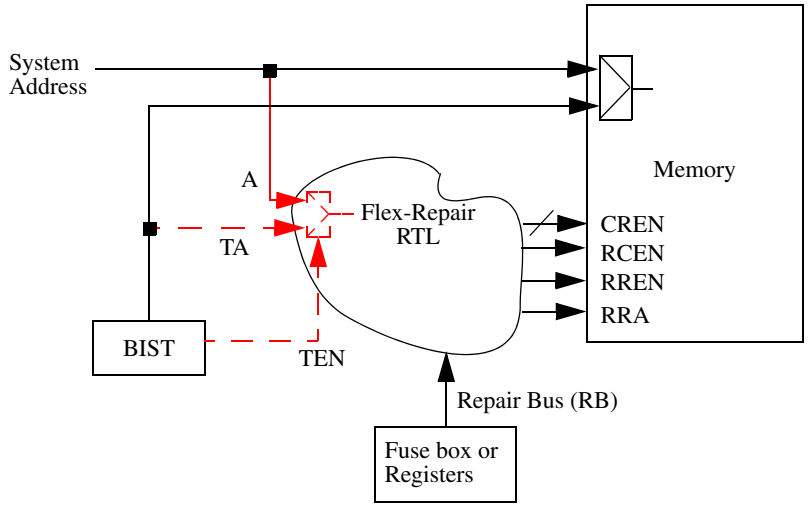
  Logically, the spare columns exist in the middle of the word.

  Lower order columns (blocks of 8) shift “up”

  Higher order bits (D/Q[N:N/2] shift “down” (Bit j is replaced by bit j-1)

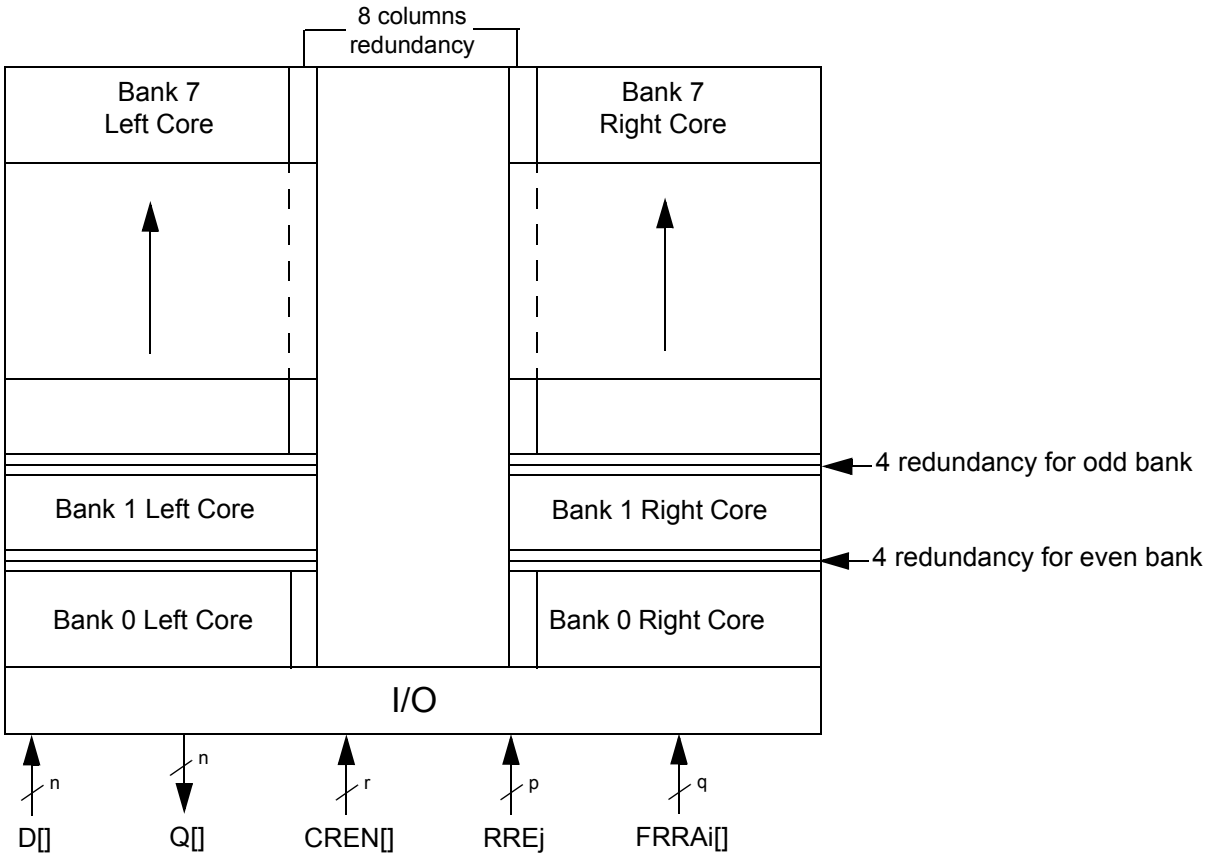
Figure 3-1 shows the redundancy RTL created by the compiler.

Figure 3-1. RTL for Redundancy Created by the Compiler



Refer to the “High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note” for more detailed information about RTL pins.

Figure 3-2. Architecture of a Typical Single-Port SRAM with Redundancy



RREj - present when redundancy is enabled; j = 1-4 for MUX rows ≤ 512, j = 1-8 for MUX rows > 512
FRRAi[] - present when redundancy is enabled; i = 1-4 for MUX rows ≤ 512, i = 1-8 for MUX rows > 512

3.2.1.2.2 Soft-Error Repair (SER)

SER automatically adds extra bits per word into the memory block and creates an accompanying RTL logic needed to implement Error Correction Codes (ECC) for SER. A composite RTL is created if both RTL and SER are enabled.

SER has two options based on your GUI option selection:

- One bit error detection and one bit error correction (1bd1bc)
- Two bit error detection and one bit error correction (2bd1bc)

3.2.1.2.3 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 and write operations by slowing down the memory access. The EMA feature is always enabled.

There are three input pins for each instance: EMA[2], EMA[1], EMA[0]. The access time and cycle times are progressively increased as the pins are driven from 000 to 111, respectively. The EMA[2:0] pins are always visible. 'Margin' sequentially increases as EMA sequentially increments from 000 through 111. Setting 000 is the fastest setting and 111 is the slowest setting. Minimum EMA setting for a given operating range is documented in the model .lib file.

3.2.1.2.4 Built-In Self Test (BIST) MUX

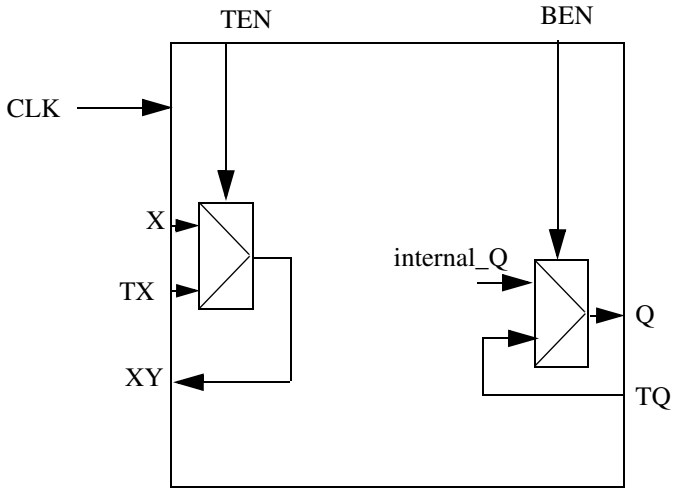
When this feature is enabled, MUX are added to critical input pins, CEN, WEN[ ], A[ ], & D[ ], and to all data output (Q) pins. Figure 3-3 shows the BIST MUX block diagram.

One of the inputs of the input BIST MUX is connected to system signals while other is connected to the BIST outputs. System or test inputs are selected, when the Test Enable pin TEN is high or low, respectively. The BIST MUX outputs are available as pins for testing. The address bus A has test address bus TA[ ], and MUX output is named AY[ ]. Similarly, CEN, WEN, and D[ ] have test inputs as TCEN, TWEN, TGWEN, and TD[ ], respectively and MUX outputs as CENY, WENY, GWENY, and DY[ ], respectively.

The bypass MUX are added before the output pins Q. These allow the test tools to have the direct controllability of the shadow logic, without going through the memory. On the single-port SRAM, inverted output of the DY test MUX is NORed together with the internal non-latched value of the write mask for each data bit. The output of these NOR gates drive one of the inputs of the bypass MUX and propagate to the output when BEN is active (or low). When BEN is high, the data from memory is output at Q[ ].

When the pipeline option is selected, a BIST MUX is added for the pipeline enable. Thus, the mission mode pin is called PEN and the test pin is TPEN. Like all of the other BIST MUX, the selection of this MUX is controlled by TEN. The output of this MUX drives a positive edge-triggered flip-flop. The output of the flip-flop is connected to the PENY pin.

Figure 3-3. Single-Port SRAM BIST MUX Block Diagram



X can be A, D, CEN and WEN.

3.2.1.2.5 Back Biasing Support

Back Biasing Support is a selectable option that allows the chip designer 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 may be slower than standard operating mode performance.

Back biasing support gives you access to drive the wells; VPW drives the P well and VNW drive the N well. It is your responsibility to ensure that the wells are properly driven in both regular mode and reduced leakage mode.

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

3.2.1.2.6 Single-Port SRAM Pipeline

When enabled, this feature places a flip-flop between the output of the memory and Q. The flip-flop is enabled when either TEN is low or the latched value from the previous cycle of PEN/TPEN is low. When the flip-flop is not enabled, it is not clocked and Q maintains its previous value. When the memory compiler switch pipeline is enabled, Q remains static as long as PEN is high.

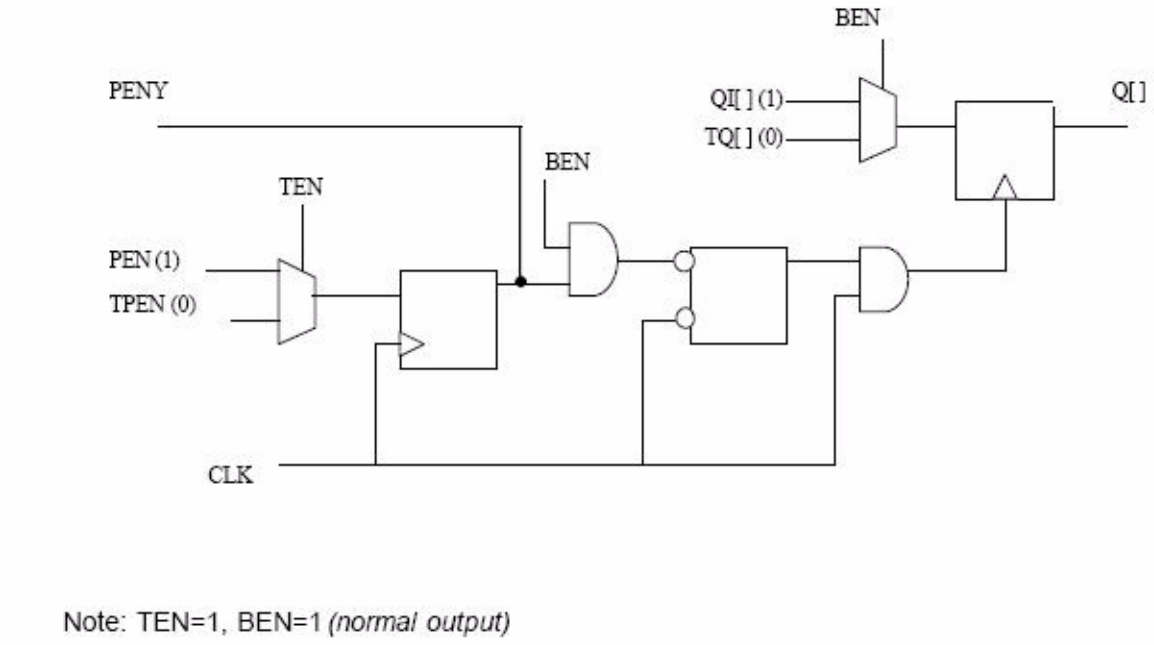
The BIST option is always available and BIST MUX is added when the pipeline option is selected. In order to ensure the testability of pipeline circuitry, PEN is set concurrently with CEN to determine if a particular read operation is available in the next cycle or if the read operation will be stalled.

Table 3-2. Pipeline Truth Table

Inputs (cycle n-1)			Inputs (cycle n)		Outputs (cycle n-1)	Outputs (cycle n)	Function
TEN	PEN	TPEN	BEN	TQ	QI	Q	
0	X	0	0	Data	X	Data	Pipeline enabled
0	X	0	1	X	QData	QData	Pipeline enabled
1	0	X	0	Data	X	Data	Pipeline enabled
1	0	X	1	X	QData	QData	Pipeline enabled
0	X	1	0	Data	X	Data	Pipeline enabled
1	1	X	0	Data	X	Data	Pipeline enabled
0	X	1	1	X	X	Q(n-1)	Pipeline disabled
1	1	X	1	X	X	Q(n-1)	Pipeline disabled

The circuit shown in Figure 3-4 is implemented for the pipeline register and assorted control logic. Flip-flops shown contain pairs of latches and do not share latches with other elements (for example, sense amp). CLK is the buffered system clock input; GTP is not. This keeps the circuit as a true “bolt-on.” The Q, QI, and PENY outputs must be separately buffered and inputs must not be connected directly to diffusion. The clock gating structure is needed for ATPG approaches that include the pipeline register in the scan chain.

Figure 3-4. Pipeline Schematic



Note: TEN=1, BEN=1 *(normal output)*

Pipeline related pin names and functions are as follows:

- PEN; pipeline enable (active low). Set concurrently with CEN to determine if a particular read operation is available in the next cycle, or if it will be stalled.
- TPEN; test version of pipeline enable. Mainly used for scan testing of flip-flop associated with PEN.
- PENY; observable output of pipeline enable flop for scan testing purposes.
- QI; non-pipelined memory output (logically equivalent to the Q port from the standard option).

Design specifics are as follows:

- The BIST option is required to be present when the pipeline option is present in order to ensure testability of pipeline circuitry.
- BEN = 0 forces clock gating of pipeline register to be disabled; for example, clock is always on.
- Figures 3-5 through 3-8 show correct pipeline operation.

Figure 3-5. Pipeline Timing - Mission Mode; Normal Operation

NOTE: TEN = 1, BEN = 1

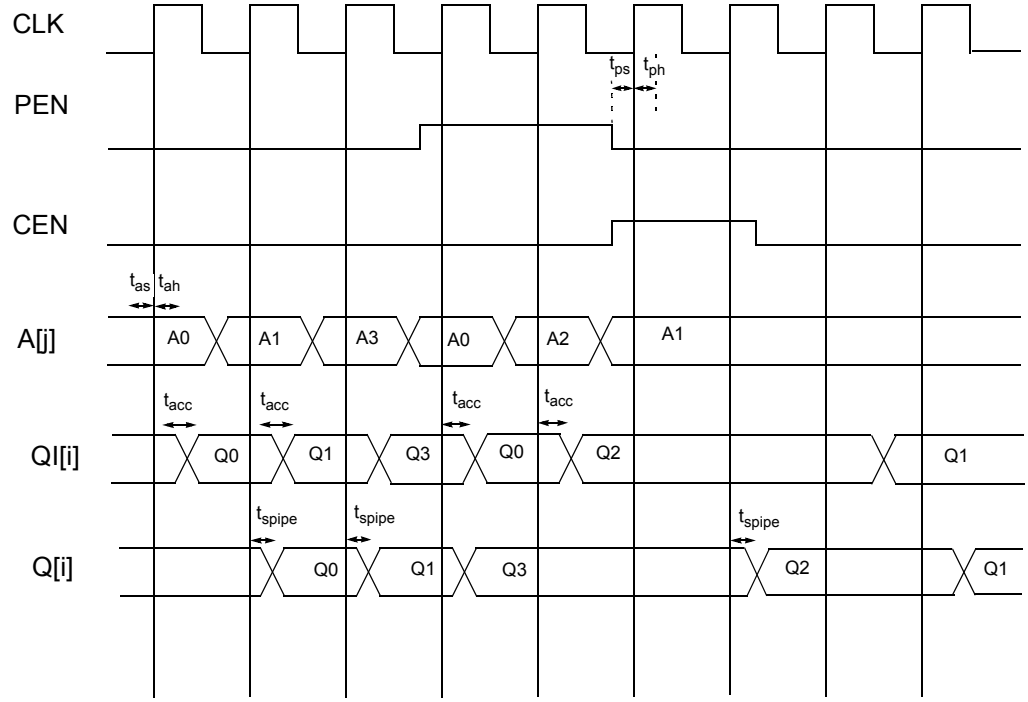


Figure 3-6. Pipeline Timing - ATPG Capture Mode

NOTE: TEN = 1, BEN = 0

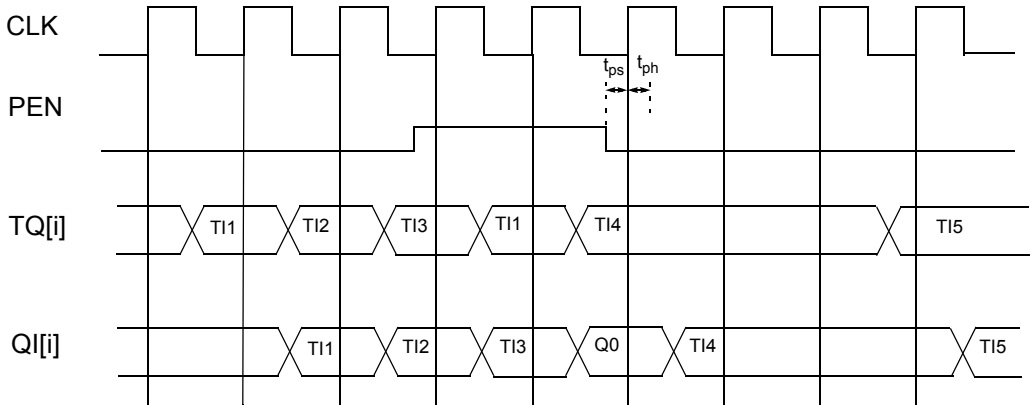


Figure 3-7. BIST Pipeline; Test Mode

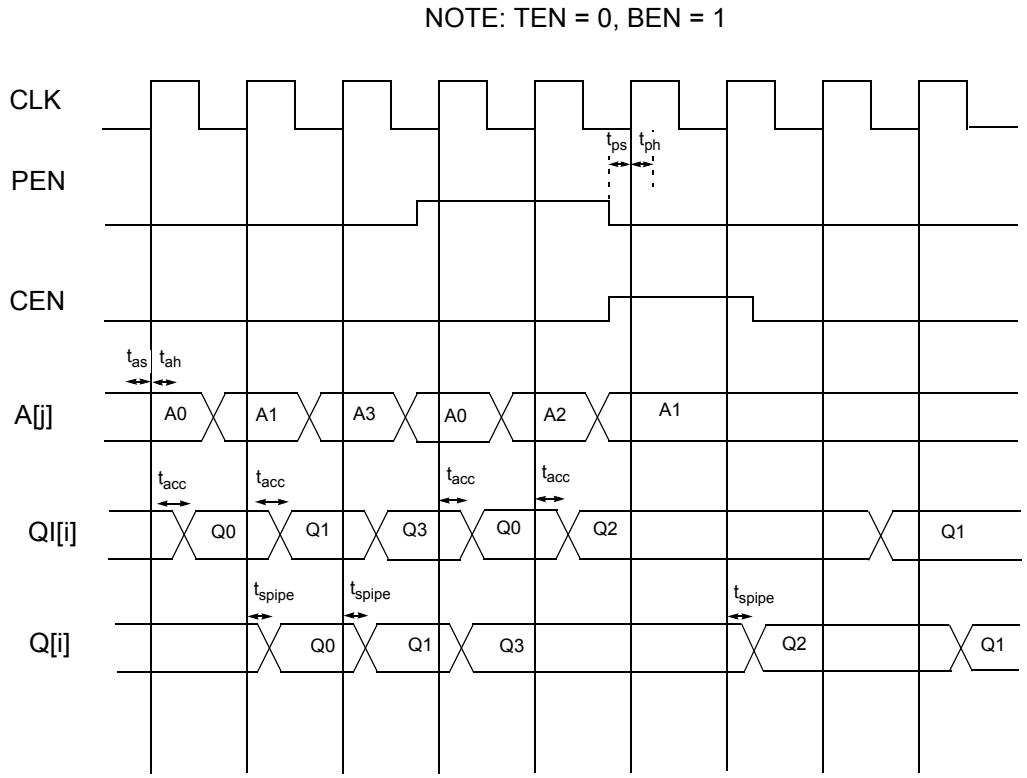


Figure 3-8. BIST Standard Test; ATPG Scan and BIST Shift Modes

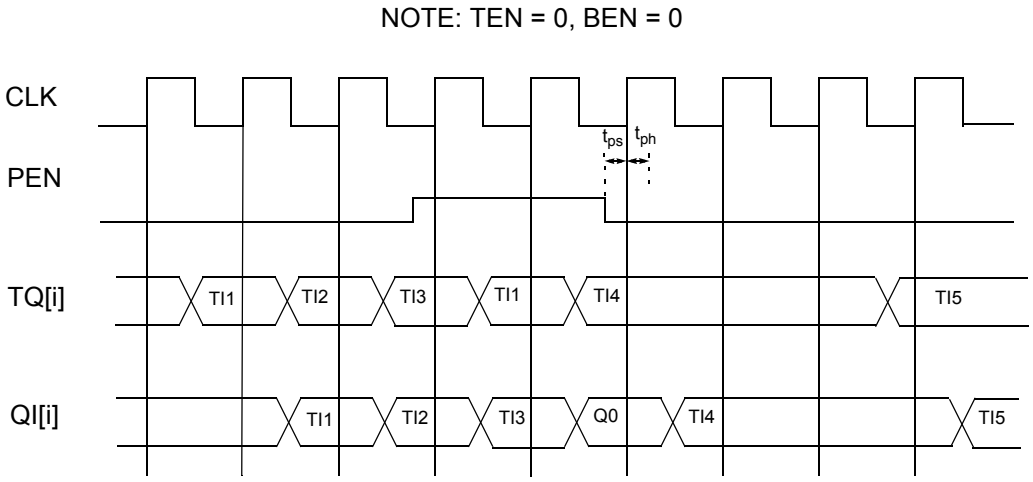


Table 3-3 lists the mode of operation for the various combinations of TEN and BEN.

Table 3-3. Pipeline Mode of Operation Options

TEN	BEN	Mode of Operation
0	0	BIST Standard Test; ATPG Scan; BIST Shift
0	1	BIST Pipeline Test
1	0	ATPG Capture
1	1	Mission Mode (Normal Operation)

Table 3-4 defines each mode of operation.

Table 3-4. Pipeline Mode of Operation Options

Mode of Operation	Description
Mission Mode	Normal Operation. All I/O to the memory should be accessed through non-test inputs and outputs.
ATPG Scan	Data is shifted through a stitched ATPG scan chain. The pipeline flip-flops and pipeline enable flip-flop are assumed to be on the chain. Inputs to these flip-flops must be driven by other flip-flops in the scan chain. Access to these flip-flops are provided through TQ and TPEN
ATPG Capture	Pipeline enable flip-flop captures data through the mission mode input PEN. The memory is bypassed so the pipeline flip-flops can be used to capture data through TQ. Typically, TQ is connected to DY.
BIST Standard Test	The BIST controls the memory and observes the results through the QI pin. Error information is stored in the pipeline registers so the BIST must have access to the pipeline registers through TQ.
BIST Shift	The BIST scans the pipeline register. BIST must have access to the pipeline register through TQA/B.
BIST Pipeline Test	The BIST tests to ensure the pipeline registers and pipeline enable circuitry work as intended. The BIST must obtain control of the pipeline enable through TPENA/B and the TPENA/B must have control of the pipeline register.

3.2.1.3 Write-Through (write-thru)

The write-through (also known as write-thru) feature allows for data latching and has an On/Off select option. When write-thru is selected on (enabled), data to be written [D(i)] can appear on the output pins [Q(i)] if the GWEN(i) pin is set accordingly. When write-thru is set to off (disabled), output data remains latched to the last read value and power consumption is reduced. Write-thru is typically selected through the GUI and the default value is off.

3.2.1.3.1 Advanced Test Feature (ATF)

When enabled, the Advanced Test Features (ATF) option invokes both Weak Bit Test (WBT) and Read Disturb Test (RDT) pins. In the compiler GUI, when the ATF option set to 1, the output has the Weak Bit Test (WBT) and Read Disturb Test (RDT) pins. With ATF set to 0, there will be no WBT or RDT pins; these will be tied low internally.

Weak Bit Test (WBT)

Weak Bit Test feature is used to detect a weak bit in a core array. This test is controlled by an external WBT pin. The WBT pin is available at the memory instance only when this option is selected in the GUI. When WBT is asserted high, the internal self-timing path is sped up. When this happens, the sense amp timing speeds up during a read cycle. A weak bitcell will develop a smaller differential at the input sense node and cause a read ‘0’ or read ‘1’ fail at the output. For a write cycle, time measured for a bit cell to flip to the word line falling is shortened. As a result, the write margin will be reduced slightly. The WBT pin needs to be tied low during regular operation.

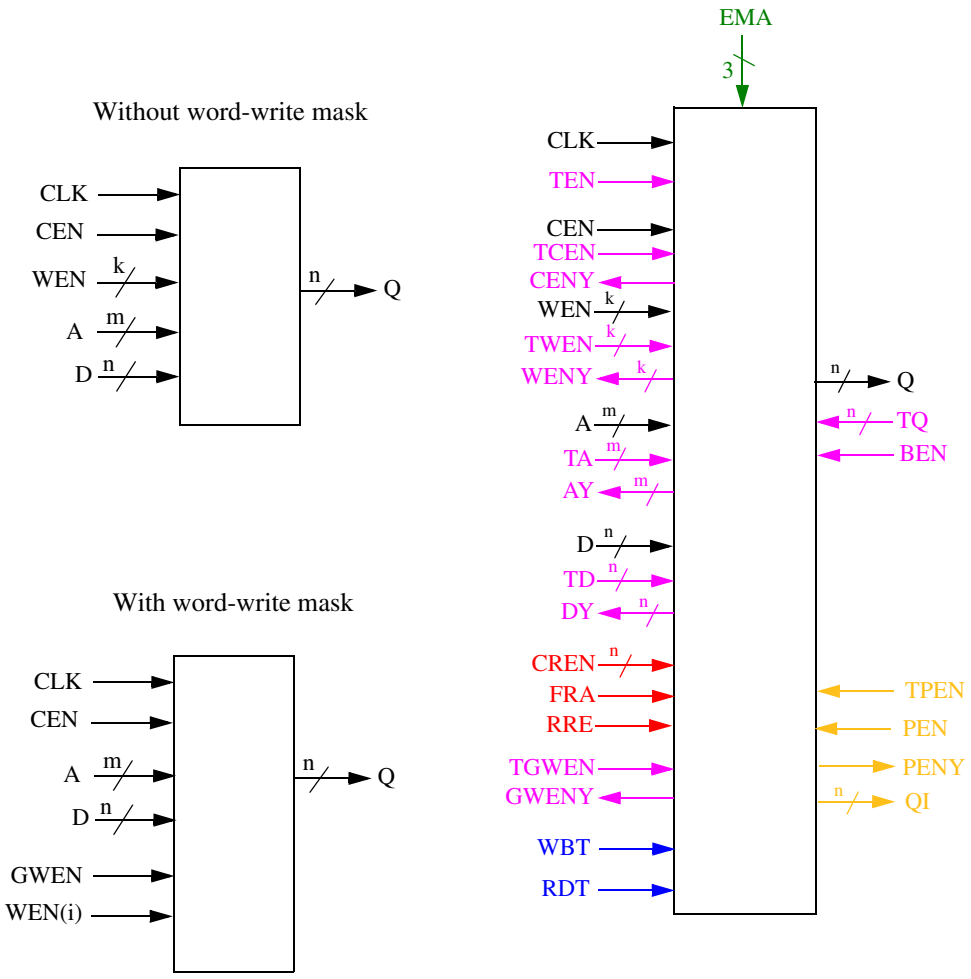
Read Disturb Test (RDT)

Read disturb test allows certain weak bit cells to fail during a read operation. This feature helps to find marginal cells that might otherwise pass a conventional test. Read disturb test is controlled by a pin called RDT, which is available at the memory instance only when this option is selected in the GUI. When RDT is set to 1, certain weak bit cells will fail read operation. RDT should be set to 0 for normal operation.

3.2.2 Single-Port SRAM Pins

Figure 3-9 shows basic and optional test/repair pins for the single-port SRAM compiler.

Figure 3-9. Single-Port SRAM Basic and Test/Repair Pins



Legend

Black	Basic
Green	Extra Margin Adjustment
Magenta	BIST MUX
Red	Redundancy
Blue	Supplementary Support
Maize	Pipeline Support

Table 3-5 provides the single-port SRAM compiler pin descriptions.

Table 3-5. Pin Descriptions for Single-Port SRAM Compilers

Name	Type	Description
Basic Pins		
A[m-1:0]	Input	Addresses (A[0] = LSB)
D[n-1:0]	Input	Data inputs (D[0] = LSB)
CEN	Input	Chip enable, active low
WEN or WEN [p-1:0]	Input	Write enable, active low; WEN[0] = LSB. *If word-write mask is enabled, this becomes a bus.
GWEN	Input	Global Write Enable; available if word-write mask is enabled
CLK	Input	Clock
Q[n-1:0]	Output	Data outputs (Q[0] = LSB)
Extra Margin Adjustment Pins		
EMA[2:0]	Input	Extra Margin Adjustment (EMA[0] = LSB)
Flex-Repair Pins		
CREN[n-1:0]	Input	Column Redundancy Enable, active low. Available when one or two column redundancy is enabled.
RRE[i]	Input	Row Redundancy Enable (active high), i=1 to 4 for up to 512Kb. Redundancy must be on.
FRAi [(n-1):0]	Input	Fuse Address. n= 1 to 8, i =1-4 for up to 512Kb. Redundancy must be on.
BIST MUX Pins		
TEN	Input	Test Mode Enable, active low. 0 = Test operation, 1 = Normal operation
TA[m-1:0]	Input	Address Test Input (TA[0] = LSB)
AY[m-1:0]	Output	Address MUX output (AY[0] = LSB)
TD[n-1:0]	Input	Test mode data inputs (TD[0] = LSB)
DY[n-1:0]	Output	Data MUX output (DY[0] = LSB)
TCEN	Input	Chip Enable Test Input, active low
CENY	Output	Chip Enable MUX output
TWEN or TWEN[p-1:0]	Input	Write Enable Test inputs, active low; TWEN[0] = LSB. *If word-write mask is enabled, this becomes a bus.
WENY or WENY[p-1:0]	Output	Write Enable MUX output; WENY[0] = LSB. *If word-write mask is enabled, this becomes a bus.
TGWEN	Input	Global Write Enable Test Input; available if write mask option is selected.
GWENY	Output	Global Write Enable MUX Output; available if write mask option is selected.
BEN	Input	Bypass Mode Enable, active low. 0 = Bypass operation, 1 = Normal operation
Pipeline Support Pins		
TPEN	Input	Pipeline test mode enable
PEN	Input	Pipeline enable (active low)
PENY	Output	Output of pipeline enable register; for scan testing
QI[n-1:0]	Output	Non-pipelined memory outputs

Table 3-5. Pin Descriptions for Single-Port SRAM Compilers (Continued)

Name	Type	Description
Power Down Mode Pins		
RETN	Input	Retention Mode Enable, active low Tie RETN to high when powers are renamed to the same name.
PGEN	Input	Power Down Mode Enable, active low
VPW	Input	P-well back biasing voltage supply pin
VNW	Input	N-well back biasing voltage supply pin
VDDPE	Input	Periphery power supply pin
VDDCE	Input	Core array power supply pin
VSSE	Input	Ground pin
Supplementary Support Pins		
WBT	Input	Weak Bit Test (Active High)
RTD	Input	Read Disturb Test (Active High)

3.2.3 Single-Port SRAM Logic Tables

This section provides logic tables for basic single-port SRAM functions and for the individual test and repair functions.

In following table, wen could be WEN when word-mask is off and WEN[ ] when it is on.

Logic functions for the basic single-port SRAM compiler features are shown in Table 3-6.

Table 3-6. Single-Port SRAM Basic Functions

CEN	WEN	GWEN	Mode	WEN[I]	Function
H	X	X	Standby		Address inputs are disabled; data stored in the memory is retained, but memory cannot be accessed for new reads or writes. Data outputs hold previous data [Q = Q(t-1)].
CEN	**Write Mask**	**Write Through**	**WEN**		**Function**
L	Off	On	0		Write to addressed memory location and read the same data
			1		Read only addressed memory location
		Off	0		Write to addressed memory location and output holds previous data; no read
			1		Read only addressed memory location
CEN	**Write Mask**	**Write Through**	**GWEN**	**WEN[I]**	**Function**
L	On	On	0	0	Write to addressed memory location and read the same data
			0	1	No write, no read; Q = D
			1	0	Read only addressed memory location
			1	1	Read only addressed memory location
		Off	0	0	Write to addressed memory location, no read; output holds previous data
			0	1	No write, no read; output holds previous data
			1	0	Read only addressed memory location
			1	1	Read only addressed memory location

Table 3-7 describes the behavior of BIST MUX at inputs to the memory.

Table 3-7. Single-Port SRAM BIST MUX at Memory Input

TEN	Mode	Function
H	Regular Mode	In this mode, the basic pins, A[], D[], CEN, and WEN[] are used for the internal operations described in Table 3-6. These inputs are also available at the MUX outputs: AY[], DY[], CENY, and WENY[], respectively.
L	Test Mode	In this mode, the test pins, TA[], TD[], TCEN, and TWEN[] are used for the internal operations described in Table 3-6. These inputs are also available at the MUX outputs: AY[], DY[], CENY, and WENY[], respectively.

Table 3-8 describes the behavior of BIST MUX at the output pins.

Table 3-8. Single-Port SRAM BIST MUX at Memory Output

BEN	Mode	Q	Function
H	Regular Mode	Last Data Read or Written	In this mode, the data from the last read or write operation is available at Q.
L	Data Bypass	Data at bypass pin	Data at the bypass pin (formed by the NOR of WENY and inverted DY) is available at output Q. This operation is not clocked. For basic features, all other operations described in Table 3-6 work when BEN=0. However, data does not appear at the output.

3.2.3.1 Redundancy

Redundancy is a selectable option. When the Repair RTL is generated, it is always generated for four row and two column redundancy. This RTL has a repair bus as an input. This Repair Bus (RB) should be connected to the fuse box. The fuse box bits are used to program the RTL in order to generate signals that correct the defect in the memory. You have the options to program the Repair Bus to set different redundancy configurations; for example, four rows and one column, as needed.

Check a GUI generated postscript datasheet to verify the delay and slew values for a particular instance.

Tables 3-9 and 3-10 show row and column redundancy logic tables, respectively.

Table 3-9. Single-Port SRAM Row Redundancy

RRE(i)	Mode	Function
H	Row redundancy enabled	If FRA(i) matches the row address of the address A, redundant row i is used
L	Row redundancy disabled	Normal operation

Table 3-10. Single-Port SRAM Column Redundancy

CREN	Mode	Function
H	Normal	The data is routed from pins (D[i], Q[i]) to the respective bit locations in the memory core.
L	Shift	For the low order bits in a word (0 to bit int(n/2)), the data is routed from pins (D[i], Q[i]) to the adjacent bits (i + 1) in the memory core. Bit int(n/2) is shifted to a redundant column. For the high order bit in a word, bit (int(n/2) + 1 to n - 1), the data is routed from pins (D[i}, Q[i] to the adjacent bits (j - 1) in the memory core. Bit int(n/2) + 1 is shifted to a redundant column.

3.2.4 Single-Port SRAM Parameters

The standard input and block parameters of a synchronous single-port SRAM are listed in Table 3-11. Refer to your compiler GUI for specific input ranges. 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 3-11. Single-Port SRAM Parameters

Input Parameters		
Parameter	**Ranges**	
number of words	MUX = 4	256 to 2048, increment = $mux \bullet 2$
	MUX = 8	256 to 4096, increment = $mux \bullet 2$
	MUX = 16	512 to 8192, increment = $mux \bullet 2$
	MUX = 32	1024 to 16384, increment = $mux \bullet 2$
number of bits[1]	MUX = 4	ECC = Off; 2 - 144, increment = 1 ECC = On; 2 - 128, increment = 1
	MUX = 8	ECC = Off; 2 - 144, increment = 1 ECC = On; 2 - 128, increment = 1
	MUX = 16	ECC = Off; 2 - 72, increment = 1 ECC = On; 2 - 64, increment = 1
	MUX = 32	ECC = Off; 2 - 36, increment = 1 ECC = On; 2 - 32, increment = 1
frequency (MHz)	1 to $1/t_{cyc} \bullet 1000$, increment = 1	
word partition size[2]	1 to min (36, bits-1) increment = 1	
multiplexer width (μm)	4, 8, 16, 32; default 16	
word-write mask	on, off; default off	
top compiler metal layer	m5 to top metal layer supported by design process	
power type	ArtiGrid	
extra margin adjustment	always on	
redundancy	on, off; default off	
redundant columns	0 or 2; redundancy must be on	
redundant rows	0, 2, or 4; redundancy must be on	
BIST MUX	on, off; default off	
soft error repair[3]	none, 1bd1bc, 2bd1bc Additional columns will be added to accommodate a modified hamming code. The maximum number of columns will not increase due to this.	
Back Biasing	on, off; default off	
Power Gating	on, off; default off	
Retention	always on	

Table 3-11. Single-Port SRAM Parameters (Continued)

Block Parameters		
Parameter	**Ranges**	
total memory bits	512 to 576 Kbit, total bits = words • bits	
rows in memory matrix	32 to 1024, increment = 2, rows = words / mux	
columns in memory matrix	16 to 1152, increment = mux; columns = bits • mux	
address lines	MUX = 4	8 to 11
	MUX = 8	8 to 12
	MUX = 16	9 to 13
	MUX = 32	10 to 14

1 Enabling the SER feature for soft-error detection and correction increases the limits beyond the specified number of bits.

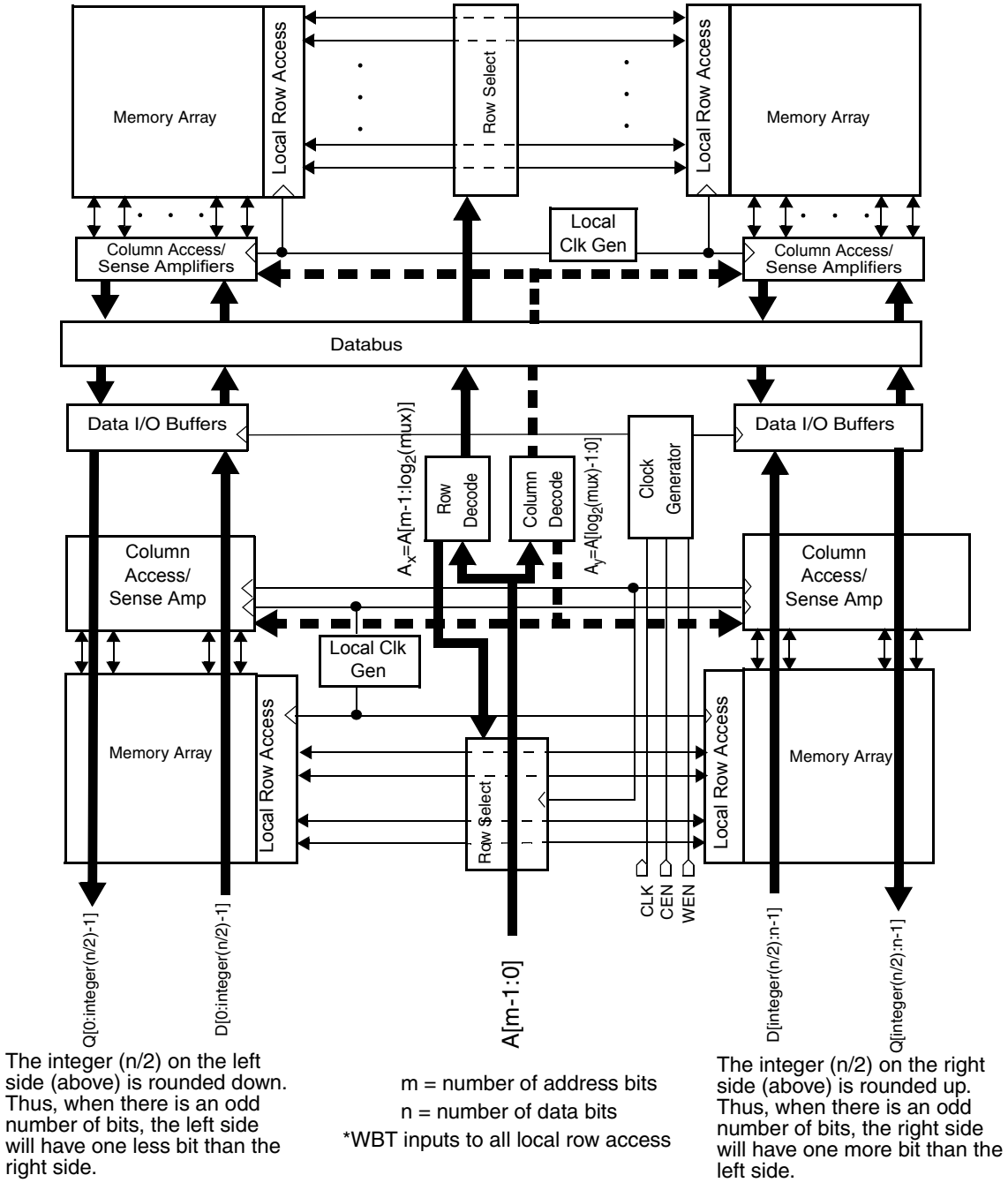
2 The input pin capacitance for each pin of the write enable bus is proportional to the size of the word partition. For example, an instance with bits=32 and wp_size=24 will have two partitions, one with 24 bits and one with 8 bits. The write enable pin for the 24 bit partition will have a significantly larger input pin capacitance than the write enable pin for the 8 bit partition. When modelling write enable timing, the write enable pin with the largest capacitance is used in the typical and slow corner timing models. The write enable pin with the smallest capacitance is used in the fast corner timing models. ARM recommends that the critical path, setup, and hold analysis be performed for all corners.

3 When the SER feature is enabled and any of the '1bd1bc' or '2bd1bc' options are selected, the actual memory generated will have more bits than specified in the GUI; these extra bits are used to store the error detection and correction code. For example, if you specify 120 bit memory in GUI and use '1bd1bc' SER option, then the generated memory will contain 127 bits. The extra 7 bits are needed for storing the error detection and correction code. So with SER option selected the range of bits specified in the GUI has to be lower then the maximum allowed for a given value of multiplexer. As an example the maximum number of bits allowed for multiplexer=8 with SER is 120 for a single-port memory instead of 128 with SER option 'none'.

3.2.5 Single-Port SRAM Block Diagrams

The synchronous high density single-port SRAM instance block diagram is shown in Figures 3-10 and 3-11.

Figure 3-10. Single-Port high density SRAM Basic Block Diagram



The integer (n/2) on the left side (above) is rounded down. Thus, when there is an odd number of bits, the left side will have one less bit than the right side.

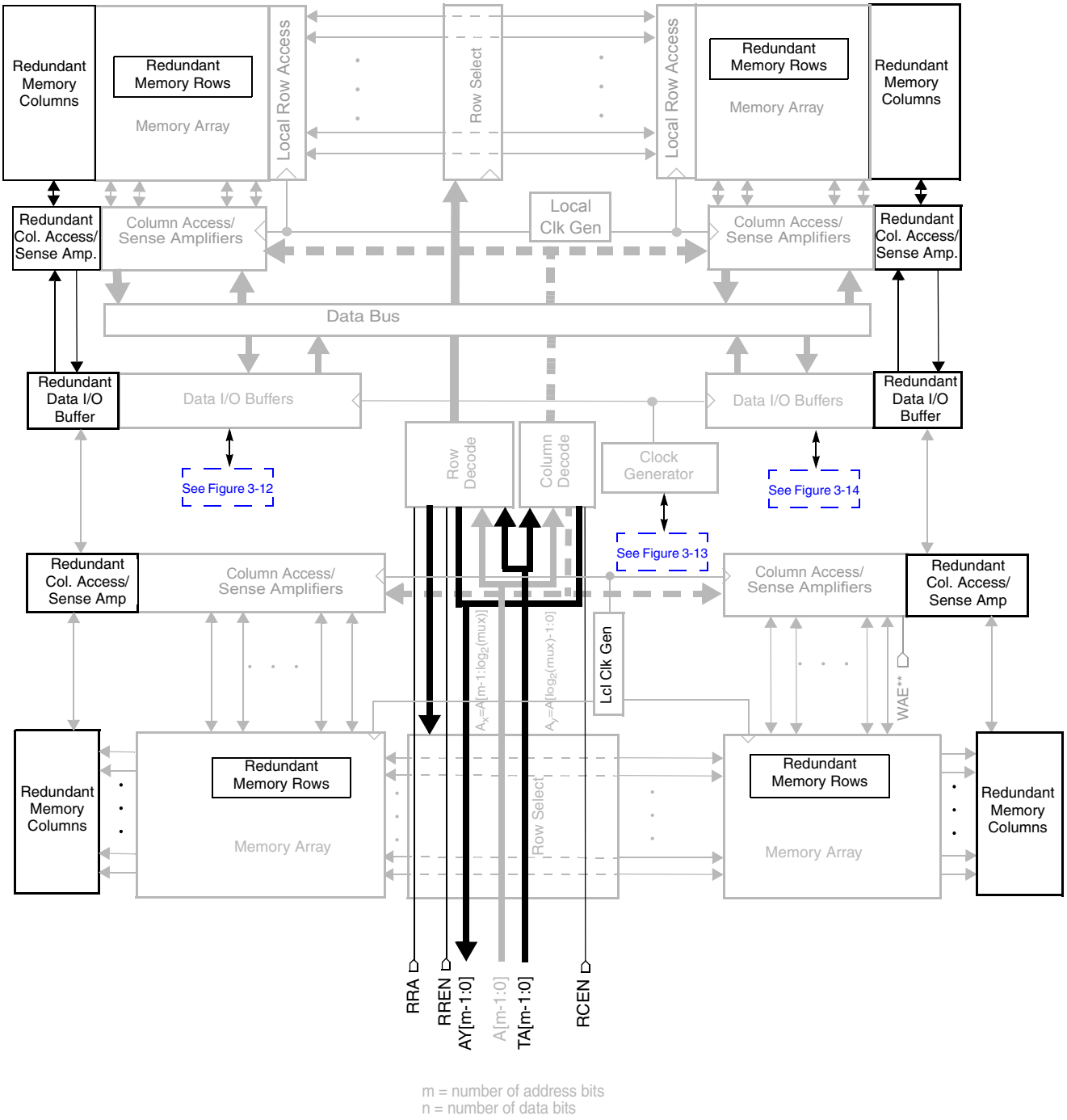
m = number of address bits
n = number of data bits
*WBT inputs to all local row access

The integer (n/2) on the right side (above) is rounded up. Thus, when there is an odd number of bits, the right side will have one more bit than the left side.

Notes:

Word-Write Mask
When the word-write mask option is turned on, the WEN pin is a bus signal and is located at the Data I/O Buffers (not shown in block diagram).
When the word-write mask option is turned off, the WEN pin is a signal pin and is located at the Clock Generator, as shown.

Figure 3-11. Single-Port high density SRAM Basic and Test/Repair Block Diagram



m = number of address bits
n = number of data bits

The integer (n/2) on the left side (above) is rounded down. Thus, when there is an odd number of bits, the left side will have one less bit than the right side.

*WBT inputs to all local row access
**WAE inputs to all column access

The integer (n/2) on the right side (above) is rounded up. Thus, when there is an odd number of bits, the right side will have one more bit than the left side.

Figure 3-12. Left Single-Port Data Buffer Detail

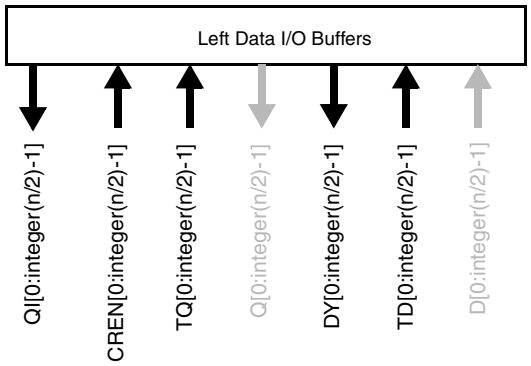


Figure 3-13. Single-Port Clock Generator Detail

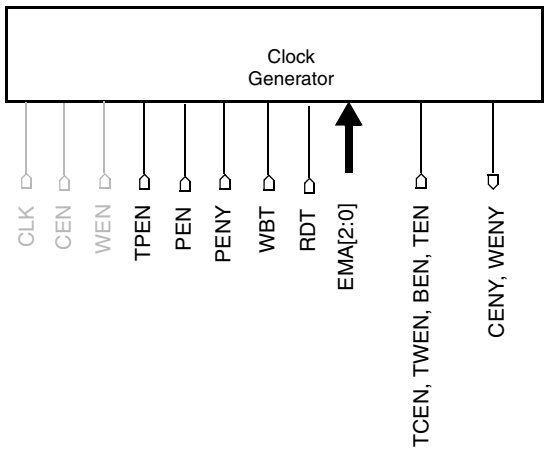
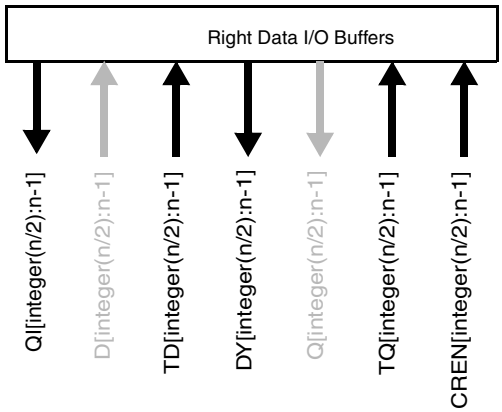


Figure 3-14. Right Single-Port Data Buffer Detail



Notes:

Word-Write Mask
When the word-write mask option is turned on, the WEN pin is a bus signal and is located at the Data I/O Buffers, (not shown in block diagram).
When the word-write mask option is turned off, the WEN pin is a signal pin and is located at the Clock Generator, as shown.

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, TD, TQ, TA, DY, and AY pins are available.
When the BIST MUX option is turned on and:
- the word-write mask option is turned on, the WEN, TWEN, and WENY pins are bus signals and are located at the Data I/O Buffers (not shown in block diagram).
- the word-write mask option is turned off, the WEN, TWEN, and WENY pins are signal pins and are located at the Clock Generator, as shown.

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

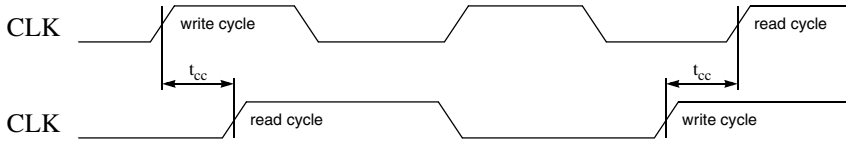
3.2.6 Single-Port SRAM Timing Specifications

This section contains the timing diagrams, timing parameters, and power parameters for the synchronous single-port SRAMs. For detailed pin descriptions, see Table 3-5 "Pin Descriptions for Single-Port SRAM Compilers" on page 3-19.

3.2.6.1 Single-Port SRAM Clock Timing Diagrams

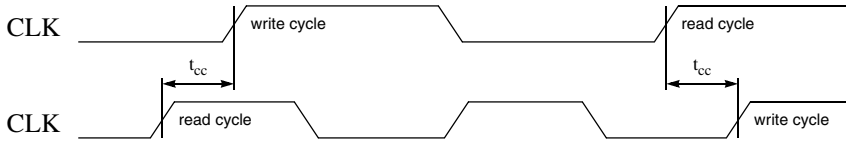
Figures 3-15 through 3-17 show clock timing diagrams for high density synchronous single-port SRAMs. Standard rising/falling delays and slews percentages are shown in these diagrams. Some compilers may be designed with different percentages. Check a GUI generated postscript datasheet to verify the delay values for a particular instance.

Figure 3-15. Single-Port SRAM Write-Read Clock Timing (Accessing Same Address)



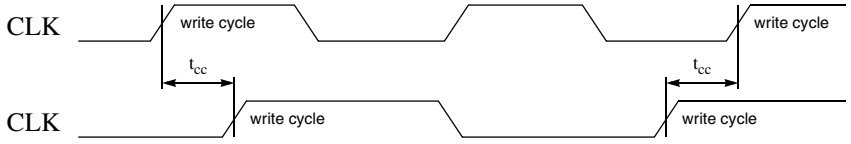
Rising delays are measured at 50% VDD and falling delays are measured at 50% VDD.
Rising and falling slews are measured from 10% VDD to 90% VDD.

Figure 3-16. Single-Port SRAM Read-Write Clock Timing (Accessing Same Address)



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.

Figure 3-17. Single-Port SRAM Write-Write Clock Timing (Accessing Same Address)

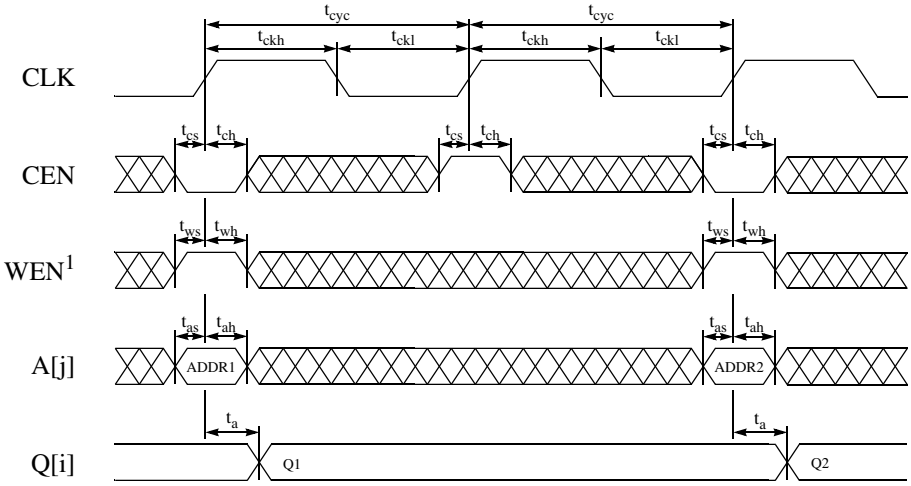


Rising delays are measured at 50% VDD and falling delays are measured at 50% VDD.
Rising and falling slews are measured from 10% VDD to 90% VDD.

3.2.6.2 Single-Port SRAM Timing Diagrams

Figures 3-18 and 3-19 show timing diagrams for high density synchronous single-port SRAMs. Standard rising/falling delays and slews percentages are shown in these diagrams. Some compilers may be designed with different percentages. Check a GUI generated postscript datasheet to verify the delay values for a particular instance.

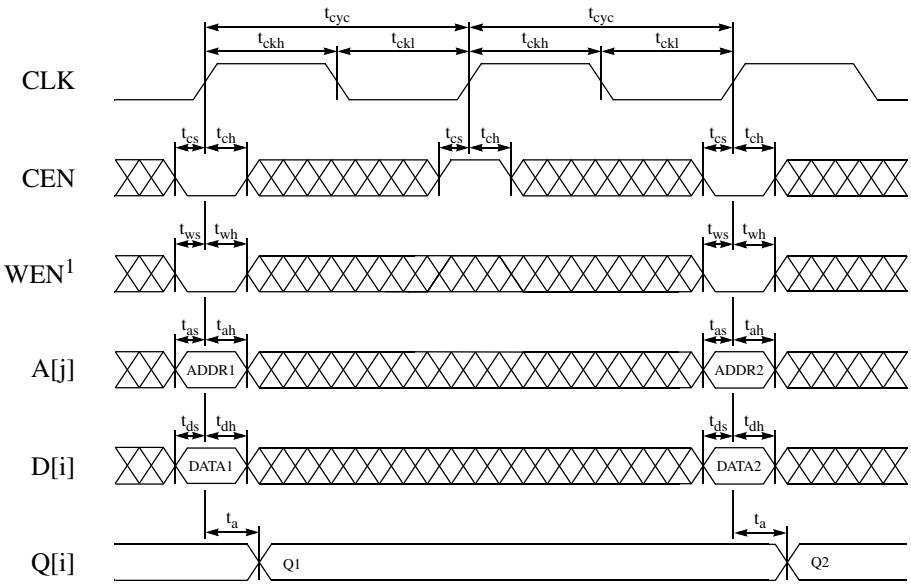
Figure 3-18. Single-Port SRAM Read-Cycle Timing



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.

[1] When word-write mask is turned off, WEN is a signal pin as shown in this diagram. When word-write mask is turned on, WEN is a bus.

Figure 3-19. Single-Port SRAM Write-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.

[1] When word-write mask is turned off, WEN is a signal pin as shown in this diagram. When word-write mask is turned on, WEN is a bus.

3.2.6.3 Single-Port SRAM Timing Parameters

The GUI generated ASCII datatable contains timing parameters listed in Table 3-12.

Table 3-12. Single-Port SRAM 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}
Write enable setup	t_{ws}
Write enable hold	t_{wh}
Data setup	t_{ds}
Data hold	t_{dh}
Clock high (minimum pulse width)	t_{ckh}
Clock low (minimum pulse width)	t_{ckl}
Clock rise slew (maximum transition time)	t_{ckr}
Access time and EMA enabled: eight numbers for eight values of EMA[1, 2]	$t_{a[0-7]}$
Extra margin enable pin setup	t_{emas}
Extra margin enable pin hold	t_{emah}
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}
Write enable setup, test pin	t_{tws}
Write enable hold, test pin	t_{twh}
Data setup, test pin	t_{tds}
Data hold, test pin	t_{tdh}
Test enable setup	t_{tens}

Table 3-12. Single-Port SRAM Timing Parameters (Continued)

Parameter	Symbol
Test enable hold	t_{tenh}
Propagation delay BEN to output	t_{benq}
Column redundancy enable setup	t_{crens}
Column redundancy enable hold	t_{crenh}
Redundant column address setup	t_{rcas}
Redundant column address hold	t_{rcah}
Row redundancy enable setup	t_{rrens}
Row redundancy enable hold	t_{rrenh}
Row redundancy address setup	t_{rras}
Row redundancy address hold	t_{rrah}
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 write enable MUX output (ns/pF)	load_weny
Load dependence factor on address MUX output (ns/pF)	load_ay
Load dependence factor on data MUX output (ns/pF)	load_dy

1 The ASCII datatable shows fixed delay values. These parameters have a load dependence (K_{load}), which is used to calculate: $\mathrm{TotalDelay} = \mathrm{FixedDelay} + (\mathrm{Kload} \times \mathrm{Cload})$, 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.

3.2.6.4 Single-Port SRAM Power Parameters

The GUI contains an ASCII datatable that provides characterization values for each corner. These values are also available in a generated postscript datasheet. AC current, AC read/write current, peak current and deselected current values include a DC leakage component equal to the standby current. Table 3-13 shows the single-port SRAM power parameters.

Table 3-13. Single-Port SRAM Power Parameters

Parameter	Symbol
AC Current [1, 2]	i_{cc}
Read AC Current [2]	i_{cc_r}
Write AC Current [2]	i_{cc_w}
Peak Current	i_{cc_peak}
Deselected Current [2, 3]	i_{cc_desel}
Standby Current [4]	$i_{cc\_standby}$
AC Current: eight numbers for eight values of EMA	$i_{cc[0:7]}$
Read AC Current: eight numbers for eight values of EMA	$i_{cc\_r[0:7]}$
Write AC Current: eight numbers for eight values of EMA	$i_{cc\_w[0:7]}$

1 Value assumes 50% read and write operations, where all addresses and 50% of input and output pins switch. This value is an average of the read and write current (i_{cc_r}, i_{cc_w}) values.

2 For sram_sp_hdc_svt_rvt_hvt, value shows dynamic current, without leakage (standby) component.

3 Value assumes the memory is deselected, all addresses switch, and 50% of data 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.

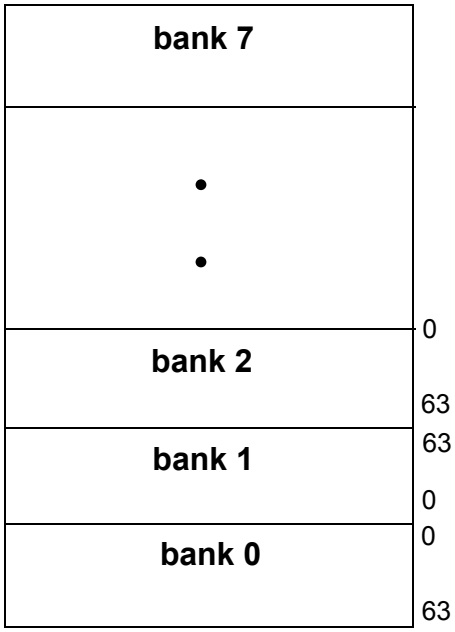
4 Value is independent of frequency and assumes all inputs and outputs are stable.

The current values shown in datasheets and datatables are based on certain assumptions. Refer to “Current Calculations” on page 3-85 for instructions on recalculating the current for a specific design. Refer to “Noise Limits” on page 3-88 for more information related to power considerations.

3.2.7 Single-Port SRAM Core Address Maps

Bit mapping for high density single-port SRAM is based on developing a bank map for each desired instance. A typical bank map consists of 8 or 16 banks and is shown in Figure 3-20.

Figure 3-20. Typical Single-Port Bank Map



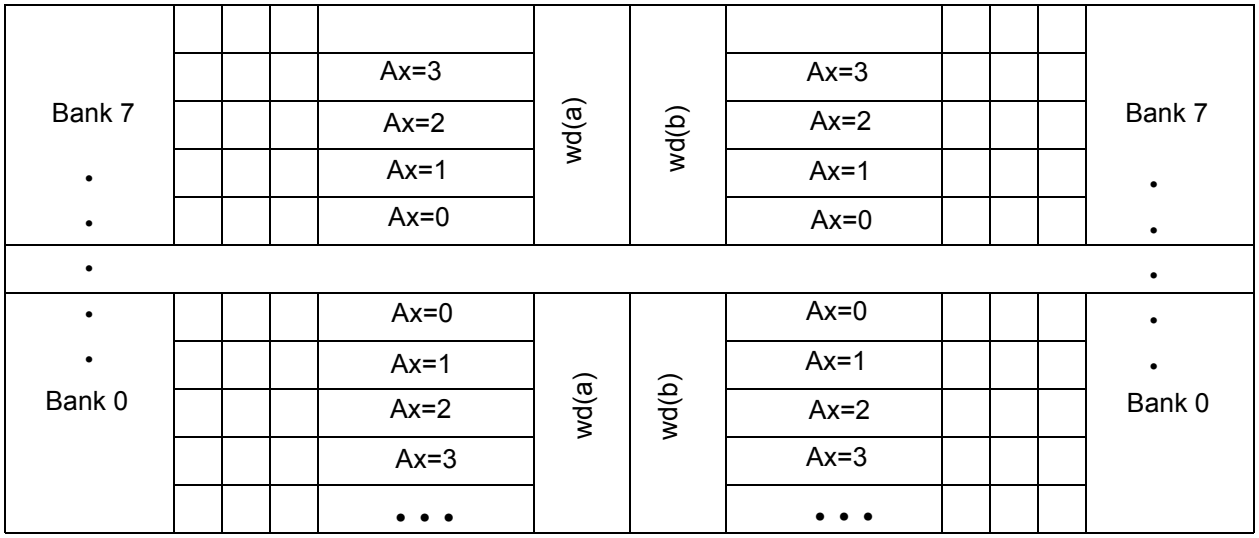
Two (2) to four (4) MSB bits are used to initially define/select a bank map.

Each compiler is mapped by completing a row or column in one bank then moving on to other banks in a given sequence.

Figure 3-21 shows the typical structure of rows in the banks. A column arrangement is similar but vertically arranged and not linear as is the row structure.

Banks are sequential from 0 through 7 (bottom to top) and 8 through 15 (top to bottom). Refer also to Figure 3-20.

Figure 3-21. Typical Single-Port Row Structure



An example of the standard physical core mapping for single-port SRAM MUX values is shown in Figures 3-22 to 3-25.

Figure 3-22. Single-Port SRAM MUX 4: Core Address Mapping

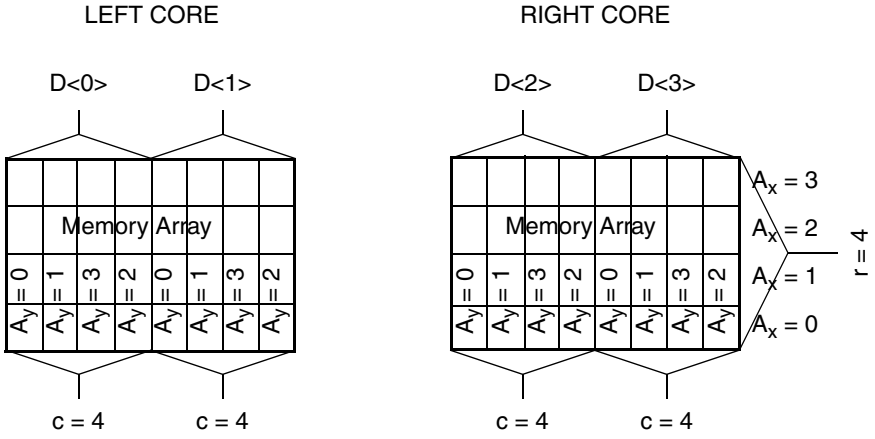


Figure 3-23. Single-Port SRAM MUX 8: Core Address Mapping

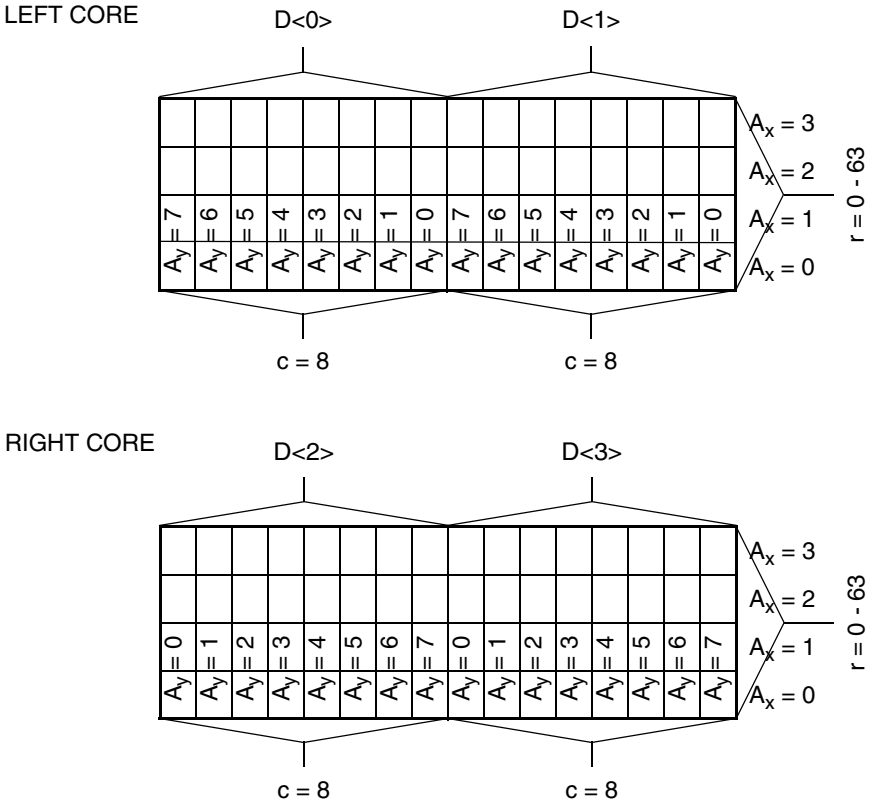


Figure 3-24. Single-Port SRAM MUX 16: Core Address Mapping

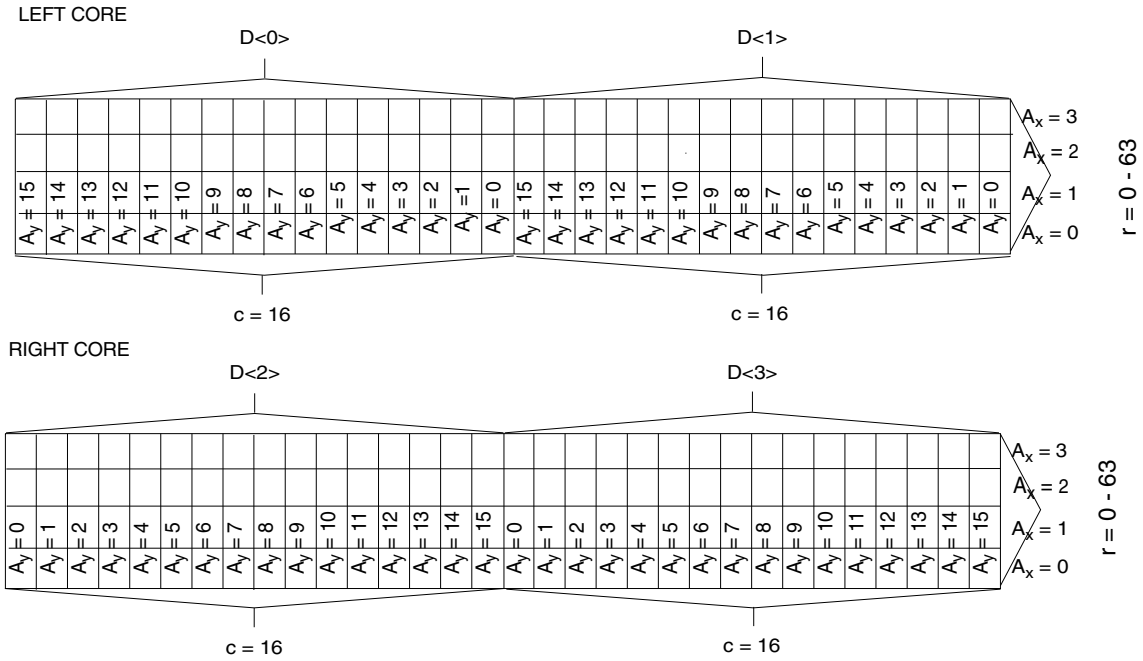
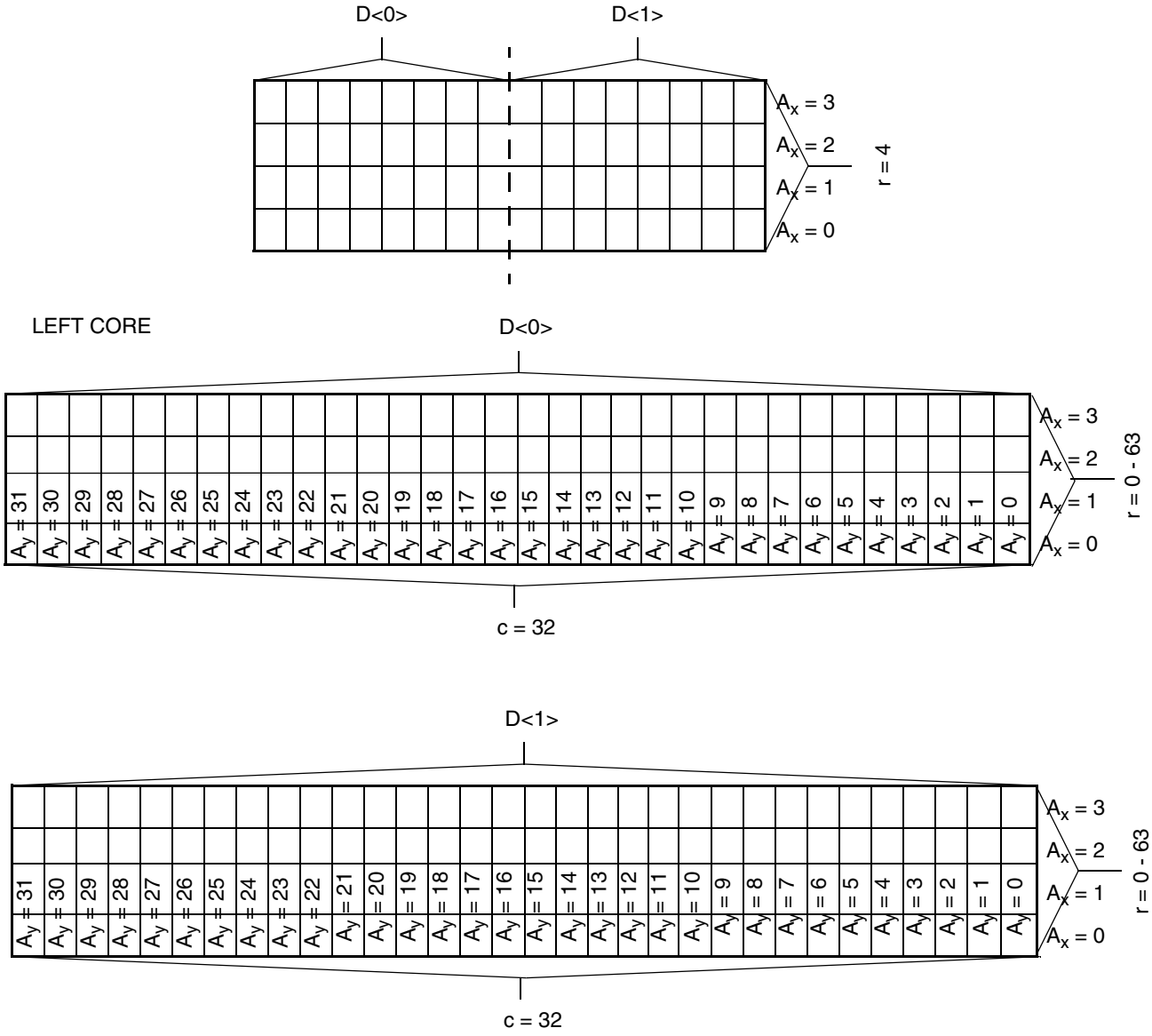
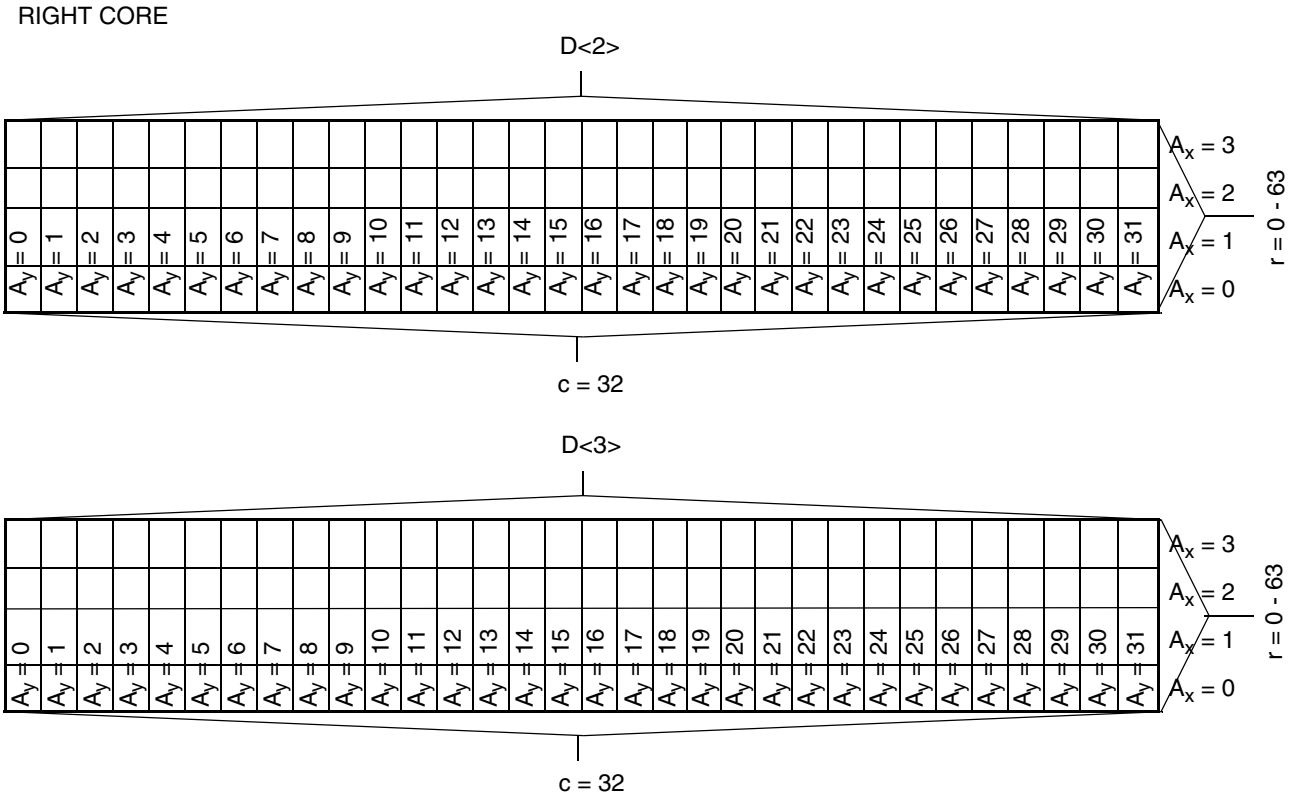


Figure 3-25. Single-Port SRAM MUX 32: Core Address Mapping (Part 1 of 2)



Single-Port SRAM MUX 32: Core Address Mapping (Part 2 of 2)



3.3 Synchronous Dual-Port SRAM Architecture and Timing Specifications

This section describes the synchronous dual-port SRAM compiler architecture, which includes pin descriptions, logic tables, block diagrams, core address maps, timing diagrams, and timing and power parameters. Executable names for applicable compilers are provided for your reference.

3.3.1 Dual-Port SRAM Description

Descriptions for the basic functionality of this compiler are provided. In addition, descriptions of features you can use to enable the test and repair functions of your compiler are provided. You can obtain further repair and test details and scenarios from the “High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note.”

3.3.1.1 Basic Functionality

SRAM access is synchronous and is triggered by the rising-edge of the clocks, CLKA and CLKB. Input addresses, input data, write enables and chip enables are latched by the rising edges of their respective clocks and respect individual setup and hold times. Both ports operate asynchronously in relation to each other.

The following descriptions apply to Port B in a manner similar to Port A

For Port A, if the word-write mask feature is not implemented, a write cycle is initiated via the compiler if the write enable, WENA, and the chip enable, CENA, are asserted at the rising-edge of CLKA. Input data, DA, is written at the address, AA. If the word-write feature is implemented, via the Compiler, data on the data input bus is partitioned to the write enable bus, WENA[x:0]. Each WENA pin has a distinct latched value, making each partition individually selectable. When the latched value of a write enable pin, WENA[i], is low; the corresponding data partition is selected and its data is written to the memory location specified on the address bus. In addition to the WENA bus, there is also a global write enable pin (GWENA). If any of the WENA[i] signals is low, the GWEN signal must also be low to enable a write operation.

For Port A, a read cycle is initiated in the RAM if CENA is asserted and WENA is de-asserted at the rising-edge of the clock, CLKA. The contents of the RAM location specified by the address, AA, are driven on the data output bus, QA. Identical operation occurs for Port B. The SRAM is allowed to access non-existing physical addresses, but the outputs will be unknown. Both ports can perform a read access to the same memory location simultaneously and will obtain correct results.

In the event of a write/read collision, the write is guaranteed and the read data is undefined. The read address for any given memory cycle can be identical to the write address of the previous memory cycle with the read data being identical to the data that was written from the previous memory write cycle.

A standby mode is provided for periods of non-operation (CENA=1 or CENB=1). The Ports A and B can enter standby mode independently. While in standby mode, address and data inputs are disabled. Data stored in the memory is retained, but the memory cannot be accessed for reads or writes.

3.3.1.2 Test and Repair Functionality

The SRAM compiler includes features such as Flex-Repair Redundancy, Soft Error Repair (SER), Built-In Self Test (BIST) MUX, and Extra Margin Adjustment (EMA), as described in this section. Feature options are described in the “Compiler Options” section on page 2-21. You can also obtain examples of how this functionality can be implemented in the “High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note.”

3.3.1.2.1 Flex-Repair/Redundancy

Redundancy is a user-selectable option. A 4 row/8 column redundancy design is described. Any redundant row may replace any other row. Redundant columns are divided into two groups of 4, one of which can be used for the lower half of the bits in the word and the other for the upper half of the bits in each word. The columns reuse the lower 2 bits of the address (A[1:0]) across all MUX values. The user can choose to use a subset of the repair capability, and the repair and reconfigure RTL will ensure that the user sees the appropriate number of redundant rows (0, 2 or 4) and columns (0 or 2). When redundancy is selected, BIST MUX are always delivered. BIST MUX election, however, can be obtained without redundancy.

Row Redundancy

Row redundancy consists of 4 independently addressable rows in the bottom bank (bank0) that can be used to repair defective rows in any of the banks. When four redundant rows are added to an instance, new signals are added. These signals are FRA1[(n-1): 0], FRA2[(n-1): 0], FRA3[(n-1): 0], FRA4[(n-1): 0], and RRE1, RRE2, RRE3, RRE4. Redundant rows are accessed if the regular address matches with any one set of FRAi[(n-1): 0] and the corresponding RREi is enabled. n is max number of address bits for row address of 256Kb.

Column Redundancy

Column redundancy is implemented in blocks of 4 columns. Redundant columns are divided into two groups of 4, one of which can be used for the lower half of the bits in the word and the other for the upper half of the bits in each word. The spare columns reuse the lower 2 bits of the address (A[1:0]) across all MUX values. Column redundancy has the following characteristics:

- Two redundant column blocks will be used, one for each half of the word.
- No change in local channel (column addressing) design (same as other MUX groups). This restricts the use of the spare blocks and requires the spare blocks use the same column addressing as the rest of the memory.
- This works as follows for each MUX value

  MUX 4: Full bit redundancy for both the lower and upper halves of each word.

  MUX 8: Full bit redundancy for both the lower and upper halves of each word independently.

  MUX 16: Bit redundancy for half the address space for the lower and upper halves of each word independently.

  Various configurations are possible, but in no case can more than a single bit be replaced in each of the upper and lower halves of any given word.

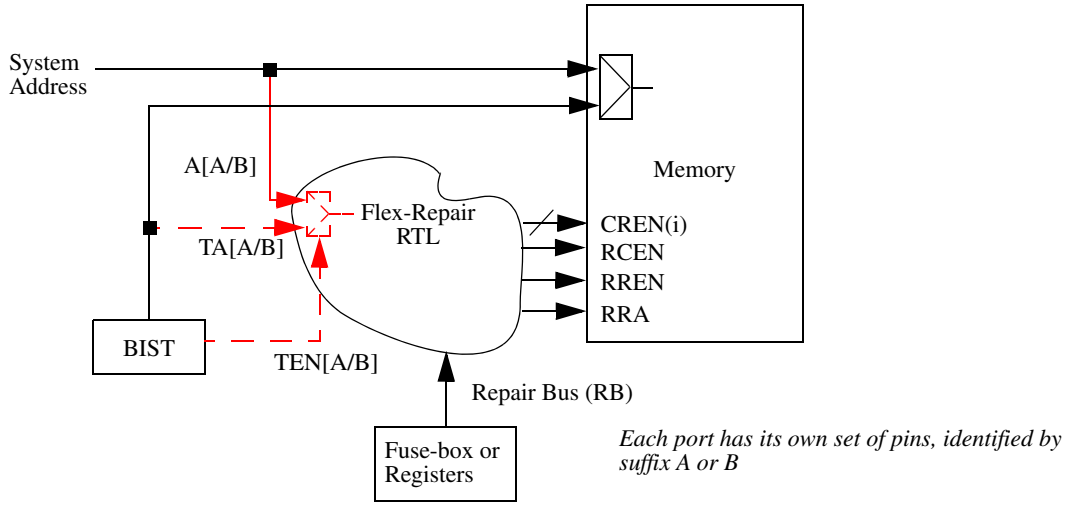
- Column shifting and muxing in accomplished in the global channel using MUX that are built into the memory instance.
- CREN operation changes from previous memories, because of the ability to control redundant bit shifting in the upper and lower half of the bits in a word independently. The number of CREN pins = (number of columns)/MUX number.

Logically, the spare columns exist in the middle of the word.

Lower order columns (blocks of 4) shift “up”

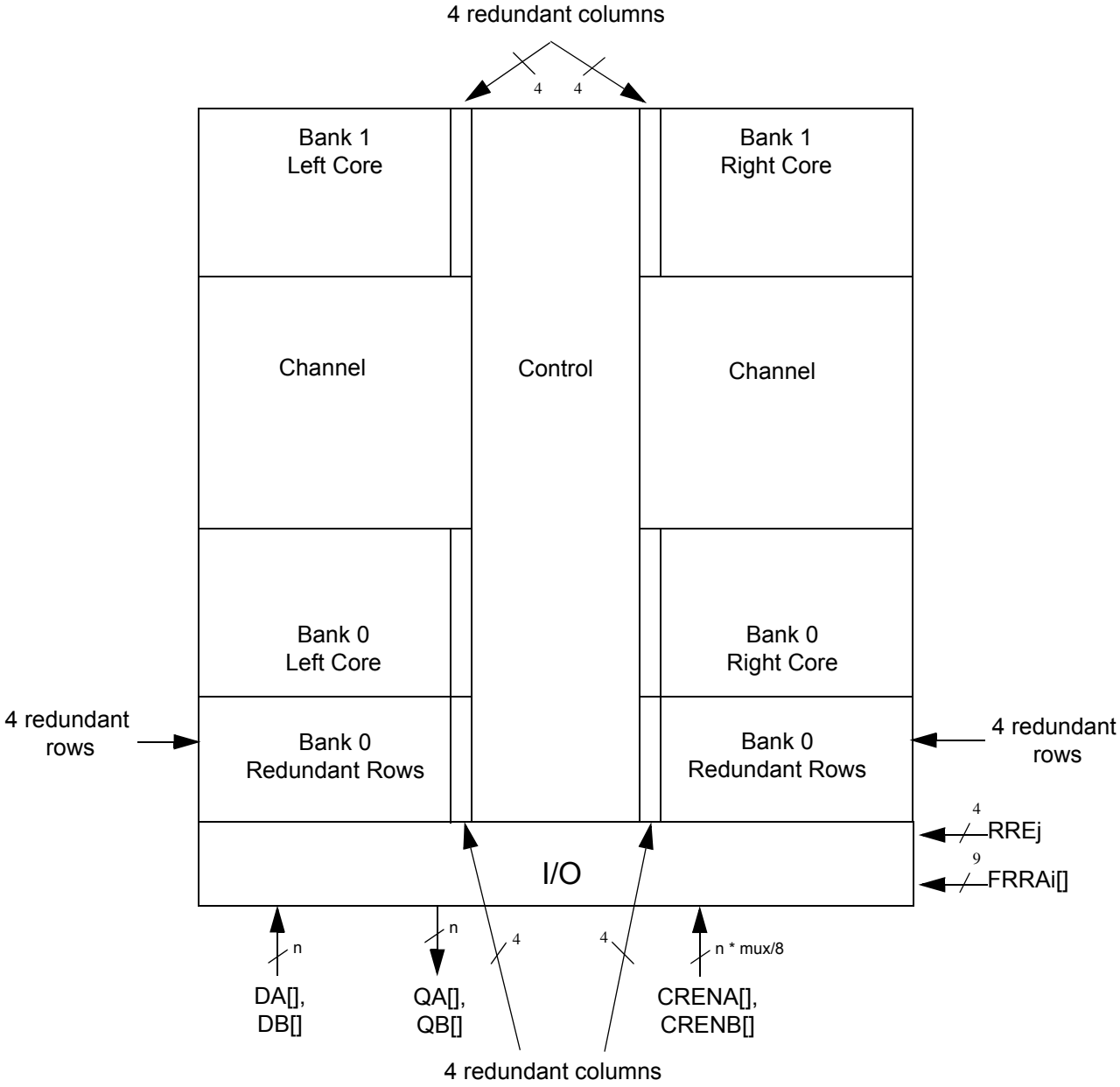
Higher order bits (D/Q[N:N/2] shift “down” (Bit j is replaced by bit j-1)

Figure 3-26. RTL for Redundancy Created by the Compiler



Refer to the “ High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note” for more detailed information about RTL pins.

Figure 3-27. Architecture of a Typical Dual-Port SRAM with Redundancy



RREj - present when redundancy is enabled; j = {1, 2, 3, 4}
FRRAi[] - present when redundancy is enabled; i = {1, 2, 3, 4}

3.3.1.2.2 Soft-Error Repair (SER)

SER adds extra bits per word into the memory block automatically and creates accompanying RTL logic needed to implement Error Correction Codes (ECC) for SER. A composite RTL is created if both RTL and SER are enabled.

SER has two options:

- Single bit error correction and single bit error detection
- Single bit error correction and double bit error detection

Refer to the “High Density 65nm and Low Power 90nm/130nm Test and Repair Features Application Note” for more detailed information about SER pins.

3.3.1.2.3 Extra Margin Adjustment (EMA)

Extra Margin Adjustment provides the option of adding delays into internal timing pulse. This delay provides extra time for memory read and write operations by slowing down the memory access. There are three input pins, named 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 111 respectively. The EMA[2:0] pins are always visible. 'Margin' sequentially increases as EMA sequentially increments from 000 through 111. Setting 000 is the fastest setting and 111 is the slowest setting. Minimum EMA setting for given operating range will be documented in the model .lib file.

3.3.1.2.4 Built-In Self Test (BIST) MUX

When this feature is enabled, MUX are added to critical input pins, CENA, WENA[ ], AA[ ], and DA[ ], and to all data output (QA) pins.

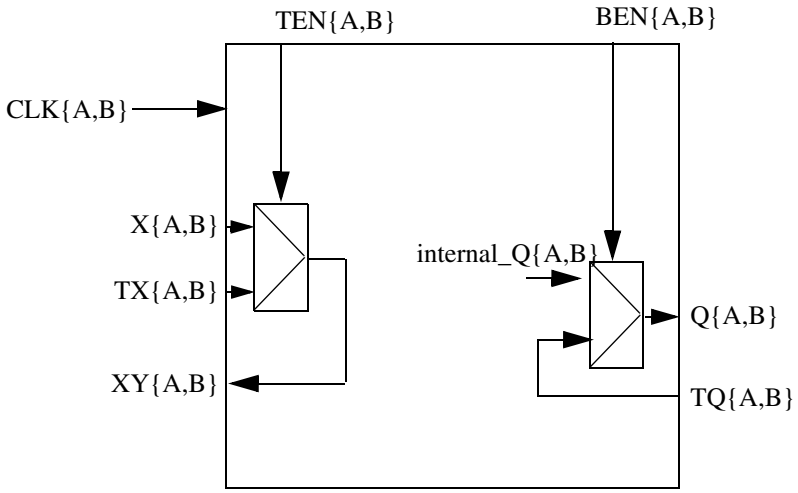
One of the inputs of the BIST MUX is connected to system signals while other is connected to the BIST outputs. System or test inputs are selected, when the Test Enable pin for that port (TENA or TENB) is high or low, respectively. The BIST MUX outputs are available as pins for testing. For Port A, the address bus AA has test address bus TAA[ ], and MUX output is named AYA[ ]. Similarly, CENA, WENA, DA[ ] have test inputs as TCENA, TWENA, TGWENA, and TDA[ ] respectively and MUX outputs as CENYA, WENYA, GWENYA, and DYA[ ], respectively. This arrangement is similar for Port B.

The bypass MUX are added before the output pins QA and QB. These MUX allow the test tools to have direct control of the shadow logic, without going through the memory. The inverted output of the DY test MUX is NORed together with the internal non-latched value of the write mask for each data bit. The output of these NOR gates drive one of the inputs of the bypass MUX and propagate to the output when BEN is active (or low). When BENA is high, the data from memory is output at QA[ ] pin. Similarly, BENB, and QB[ ] are used for Port B.

When the pipeline option is selected, a BIST MUX is added for the pipeline enable. The mission mode pin is called PENA and PENB, and the test pin is TPENA/B. Like all of the other BIST MUX, the selection of this MUX is controlled by TEN. The output of this MUX drives a positive edge-triggered flip-flop. The output of the flip-flop is connected to the PENYA pin for Port A and PENYB for Port B.

Figure 3-28 shows the BIST MUX block diagram, where each pin applies to ports A and B.

Figure 3-28. Dual-Port SRAM BIST MUX Block Diagram



X can be A, D, CEN and WEN.

3.3.1.2.5 Back Biasing Support

Back Biasing Support is a selectable option that allows the chip designer 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 will be 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.

3.3.1.2.6 Advanced Test Feature (ATF)

When enabled, the Advanced Test Features (ATF) option invokes both Weak Bit Test (WBT) and Read Disturb Test (RDT) pins. In the compiler GUI, when the ATF option set to 1, the output has the Weak Bit Test (WBT) and Read Disturb Test (RDT) pins. With ATF set to 0, there will be no WBT or RDT pins; these will be tied low internally.

Weak Bit Test (WBT)

The Weak Bit Test feature is used to detect a weak bit in a core array. This test is controlled by an external WBT pin. The WBT pin is available at the memory instance only when this option is selected in the GUI. When WBT is asserted high, the internal self-timing path is sped up. When this happens, the sense amp timing speeds up during a read cycle. A weak bitcell will develop a smaller differential at the input sense node and cause a read ‘0’ or read ‘1’ fail at the output. For a write cycle, time measured for a bit cell to flip to the word line falling is shortened. As a result, the write margin will be reduced slightly. The WBT pin needs to be tied low during regular operation.

Read Disturb Test (RDT)

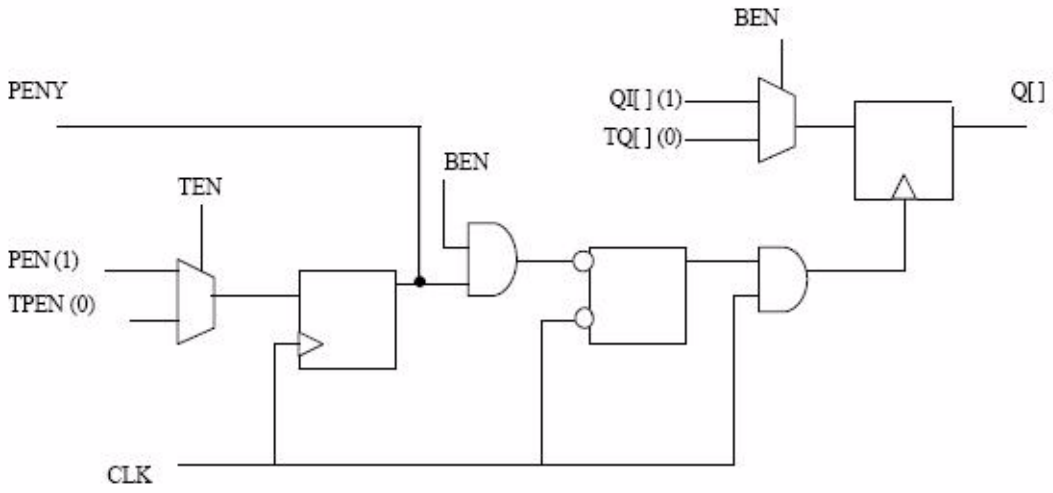
Read disturb test allows certain weak bit cells to fail during a read operation. This feature helps to find marginal cells that might otherwise pass a conventional test. Read disturb test is controlled by a pin called RDT, which is available at the memory instance only when this option is selected in the GUI. When RDT is set to 1, certain weak bit cells will fail read operation. RDT should be set to 0 for normal operation.

3.3.1.2.7 Pipeline

When the pipeline option is selected, a flip-flop is placed between the output of the memory and QA/B. The flip-flop is enabled when either TENA/B is low or the latched value from the previous cycle of PENA/B or TPENA/B is low. Otherwise, the flip-flop is not clocked and QA/B maintains its previous value. BIST option is required when the pipeline option is selected, in order to ensure testability of pipeline circuitry. PENA/B is set concurrently with CENA/B to determine if a particular read operation will be available in the next cycle, or if it will be stalled.

The circuit shown in Figure 3-29 is implemented for the pipeline register and assorted control logic. The figure applies to A and B ports. Flip-flops shown contain pairs of latches and do not share latches with other elements (for example, sense amp). CLK is the buffered system clock input; GTP is not. This keeps the circuit as a true “bolt-on.” The Q, QI, and PENY outputs must be separately buffered and inputs must not be connected directly to diffusion. The clock gating structure is needed for ATPG approaches that include the pipeline register in the scan chain.

Figure 3-29. Dual-Port SRAM Pipeline Schematic



Note: TEN=1, BEN=1 *(normal output)*

Pipeline related pin names and functions are as follows:

- PENA/B; pipeline enable (active low). Set concurrently with CEN to determine if a particular read operation is available in the next cycle, or if it will be stalled.
- TPENA/B; test version of pipeline enable. Mainly used for scan testing of flip-flop associated with PEN.
- PENYA/B; observable output of pipeline enable flop for scan testing purposes.
- QIA/B; non-pipelined memory output (logically equivalent to the Q port from the standard option).

Design specifics are as follows:

- BIST option is required to be present when the pipeline option is present in order to ensure testability of pipeline circuitry.
- BEN = 0 forces clock gating of pipeline register to be disabled; for example, clock is always on.
- Figures 3-30 through 3-33 show correct pipeline operation.
- Dual-port SRAM requires two pipeline registers, one for each port. Each register requires separate pipeline enable circuits. Port A uses CLKA and the Port B uses CLKB.

Figure 3-30. Dual-Port SRAM Pipeline Timing - Mission Mode; Normal Operation

NOTE: TEN = 1, BEN = 1

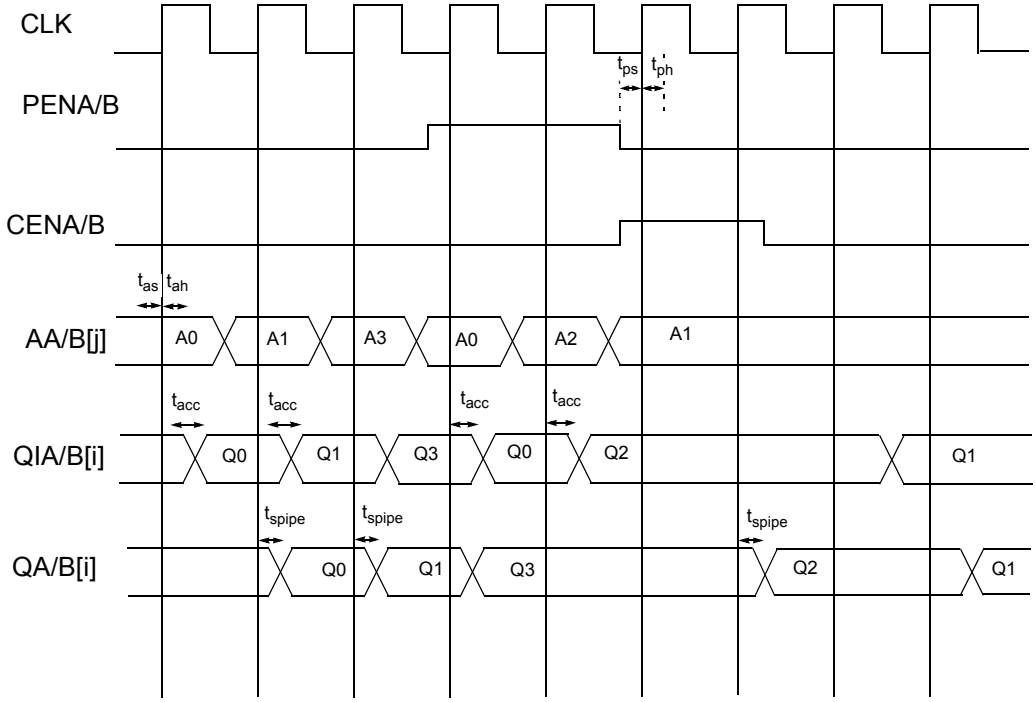


Figure 3-31. Dual-Port SRAM Pipeline Timing - ATPG Capture Mode

NOTE: TEN = 1, BEN = 0

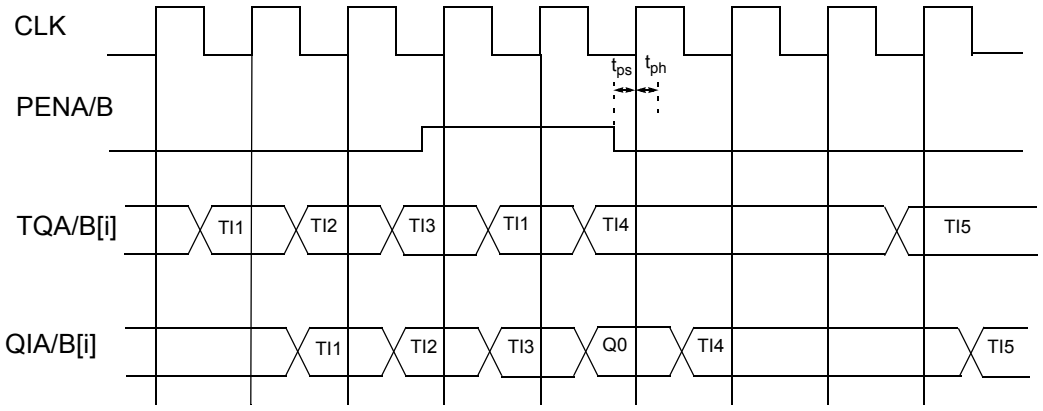


Figure 3-32. Dual-Port SRAM BIST Pipeline; Test Mode

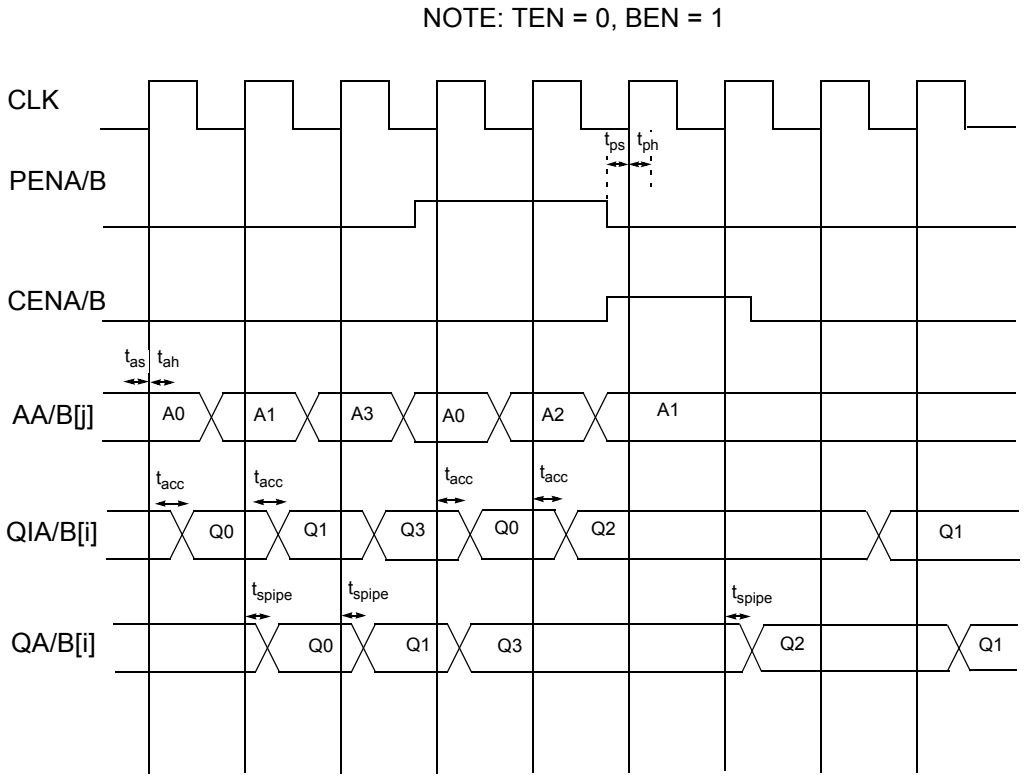


Figure 3-33. BIST Standard Test; ATPG Scan and BIST Shift Modes

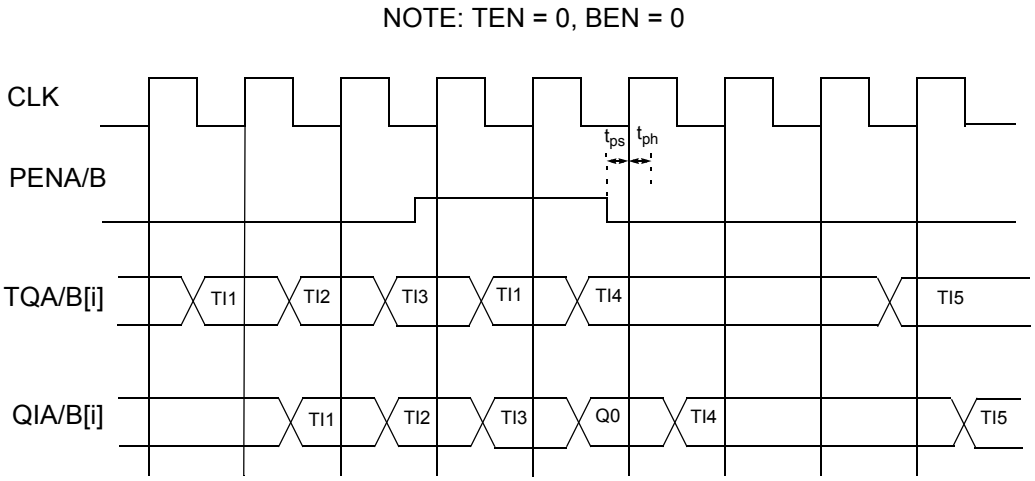


Table 3-14 lists the mode of operation for the various combinations of TEN and BEN.

Table 3-14. Dual-Port SRAM Pipeline Mode of Operation Options

TEN	BEN	Mode of Operation
0	0	BIST Standard Test; ATPG Scan; BIST Shift
0	1	BIST Pipeline Test
1	0	ATPG Capture
1	1	Mission Mode (Normal Operation)

Table 3-15 defines each mode of operation.

Table 3-15. Dual-Port SRAM Pipeline Mode of Operation Options

Mode of Operation	Description
Mission Mode	Normal Operation. All I/O to the memory should be accessed through non-test inputs and outputs.
ATPG Scan	Data is shifted through a stitched ATPG scan chain. The pipeline flip-flops and pipeline enable flip-flop are assumed to be on the chain. Inputs to these flip-flops must be driven by other flip-flops in the scan chain. Access to these flip-flops are provided through TQ and TPEN
ATPG Capture	Pipeline enable flip-flop captures data through the mission mode input PEN. The memory is bypassed so the pipeline flip-flops can be used to capture data through TQ. Typically, TQ is connected to DY.
BIST Standard Test	The BIST controls the memory and observes the results through the QI pin. Error information is stored in the pipeline registers so the BIST must have access to the pipeline registers through TQ.
BIST Shift	The BIST scans the pipeline register. BIST must have access to the pipeline register through TQ.
BIST Pipeline Test	The BIST tests to ensure the pipeline registers and pipeline enable circuitry work as intended. The BIST must obtain control of the pipeline enable through TPEN and the TPEN must have control of the pipeline register.

3.3.1.2.8 Write-Through (write-thru)

The write-thru feature allows for data latching. By default, write through is always on. [D(i)], the data to be written will appear on the output pins [Q(i)] if the GWEN(i) pin is set appropriately.

3.3.2 Dual-Port SRAM Pins

Figure 3-9 shows basic and optional test/repair pins for the dual-port SRAM compiler.

Figure 3-34. Dual-Port SRAM Basic and Test/Repair Pins

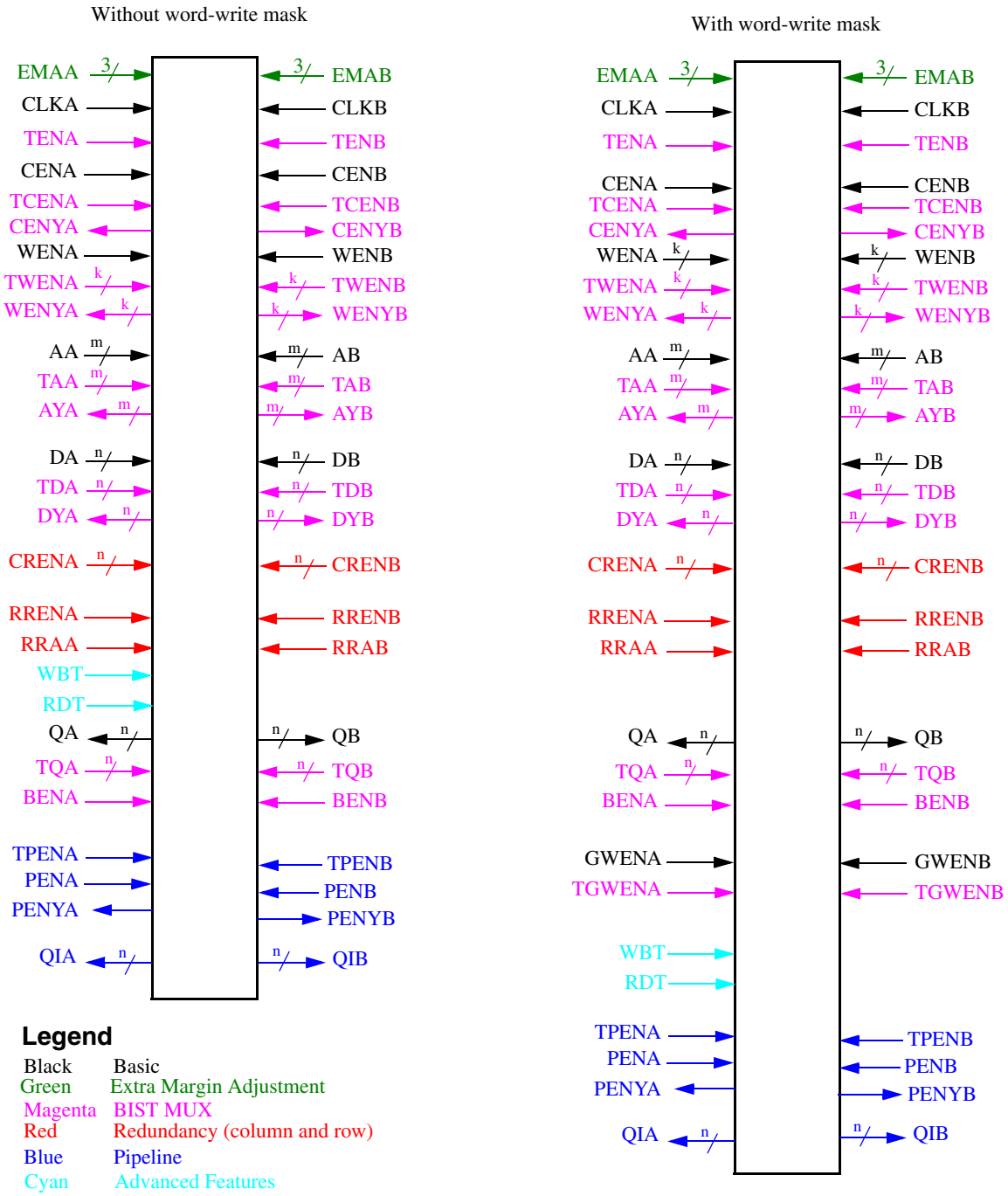


Table 3-16 provides the dual-port (ports A and B) SRAM compiler pin descriptions.

Table 3-16. Pin Descriptions for Dual-Port SRAM Compilers

Name	Type	Description
Basic Pins		
AA[m-1:0]. AB[m-1:0]	Input	Addresses (AA[0] = LSB), (AB[0] = LSB)
DA[n-1:0], DB[n-1:0]	Input	Data inputs (DA[0] = LSB), (DB[0] = LSB)
CENA, CENB	Input	Chip Enables, active low
WENA[*], WENB[*]	Input	Write Enables, active low. *If word-write mask is enabled, this becomes a bus.
GWENA, GWENB	Input	Global Write Enable; available if write mask option is selected.
CLKA, CLKB	Input	Clocks
QA[n-1:0], QB[n-1:0]	Output	Data Outputs (QA[0] = LSB), (QB[0] = LSB)
Extra Margin Adjustment Pins		
EMAA[2:0], EMAB[2:0]	Input	Extra Margin Adjustment (EMAA[0] = LSB), (EMAB[0] = LSB)
Flex-Repair Pins		
CRENA[n-1:0], CRENB[n-1:0]	Input	Column Redundancy Enables, active low (CRENA[0] = LSB), (CRENB[0] = LSB). Available when one or two column redundancy is enabled.
RRENA, RRENB	Input	Row Redundancy Enables (active low); CRENA/B[0] = LSB
RRAA[1:0], RRAB[1:0]	Input	Redundant Row Addresses.
BIST MUX Pins		
TENA, TENB	Input	Test mode Enables, active low: 0 = Test operation, 1 = Normal operation
TAA[m-1:0], TAB[m-1:0]	Input	Address Test Inputs (TAA[0] = LSB), (TAB[0] = LSB)
AYA[m-1:0], AYB[m-1:0]	Output	Address MUX outputs (AYA[0] = LSB), (AYB[0] = LSB)
TDA[n-1:0], TDB[n-1:0]	Input	Data Test inputs (TDA[0] = LSB), (TDB[0] = LSB)
DYA[n-1:0], DYB[n-1:0]	Output	Data MUX outputs (DYA[0] = LSB), (DYB[0] = LSB)
TCENA, TCENB	Input	Test Mode Chip Enable (TCEN) Test inputs, active low
CENYA, CENYB	Output	Chip Enable (CEN) MUX outputs
TWENA [*], TWENB [*], TGWENA, TGWENB	Input	Test Mode Write Enable Test inputs, active low. (TWENA[0] = LSB), (TWENB[0] = LSB). TWENA [*], TWENB [*],TGWENA, TGWENB -If word-write mask is on. TWENA, TWENB is write mask is off.
WENYA[*], WENYB[*], GWENYA, GWENYB	Output	Write Enable MUX outputs. (WENA[0] = LSB), (WENB[0] = LSB) *If word-write mask is enabled, this becomes a bus.
BENA, BENB	Input	Bypass Mode Enables, active low. 0 = Bypass operation, 1 = Normal operation
TQA[n-1:0], TQB[n-1:0]	Input	Bypass Q inputs in write mode (if BEN = 0, Q[n-1:0] = TQ[n-1:0])
Pipeline Pins		
TPENA, TPENB	Input	Pipeline test mode enable
PENA, BENB	Input	Pipeline enable (active low)
PENYA, PENYB	Output	Output of pipeline enable register, for scan testing
QIA[n-1:0]; QIB[n-1:0]	Output	Non-pipelined memory outputs

Table 3-16. Pin Descriptions for Dual-Port SRAM Compilers (Continued)

Name	Type	Description
Power Down Mode Pins		
RETN	Input	Retention Mode Enable, active low Tie RETN to high when powers are renamed to the same name.
PGEN	Input	Power Down Mode Enable, active low
VPW	Input	P-well back biasing voltage supply pin
VNW	Input	N-well back biasing voltage supply pin
VDDPE	Input	Periphery power supply pin
VDDCE	Input	Core array power supply pin
VSSE	Input	Ground pin
Supplementary Support Pins		
WBT	Input	Weak Bit Test (active high)
RDT	Input	Read Disturb Test (active high)

3.3.3 Dual-Port SRAM Logic Tables

This section provides logic tables for basic dual-port SRAM functions and for the individual test and repair functions. Information applies to both Port A and Port B. wen is WEN when word-mask is on and WEN[ ] when it is off.

Tables 3-17 shows the logic functions for basic dual-port SRAM compiler functions.

Table 3-17. Dual-Port SRAM Basic Functions

CEN	WEN	Q	Mode	Function
H	X	Last Data	Standby	Address inputs are disabled; data stored in the memory is retained, but the memory cannot be accessed for new reads or writes. Data outputs remain stable
L	L	Data In	Write	**Word Write**: Data on the data input bus, D[n-1:0] is written to the memory location specified by the address bus, A[m-1:0], and is driven through to the data output bus, Q[n-1:0]. **Word Write Mask**: The corresponding data partition is selected by the write enables, WEN[p-1:0] and that data is written to the memory location specified by the address bus, A[m-1:0], and is driven through to the data output bus, Q[n-1:0].
L	H	SRAM 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 3-7 describes the behavior of BIST MUX at inputs to the memory

Table 3-18. Dual-Port SRAM BIST MUX at Memory Input

TEN	Mode	Function
H	Regular Mode	In this mode, the basic pins, A[], D[], CEN, and WEN[] are used for the internal operations described in Table 3-17. These inputs are also available at the MUX outputs: AY[], DY[], CENY, and WENY[], respectively.
L	Test Mode	In this mode, the test pins, TA[], TD[], TCEN, and TWEN[] are used for the internal operations described in Table 3-17. These inputs are also available at the MUX outputs: AY[], DY[], CENY, and WENY[], respectively.

Table 3-8 describes the behavior of BIST MUX at the output pins.

Table 3-19. Dual-Port SRAM BIST MUX at Memory Output (Bypass Mode)

BEN	Mode	Q	Function
H	Regular Mode	Last Data Read or Written	In this mode, the data from the last read or write operation is available at Q.
L	Data Bypass	Data at bypass pin	Data at the bypass pin (formed by the NOR of WENY and inverted DY) is available at output Q. This operation is not clocked. Only data does not appear at output; all other operations described in Table 3-17 work when BEN=0.

3.3.3.1 Redundancy

When the Repair RTL is generated it is always generated for for rows and two column blocks with the repair bus (RB) as an input. The RB should be connected to fusebox; the fusebox bits are used to program the RTL and generate signals that correct any memory defect. You have the options to program the RB to obtain different redundancy configurations as needed.

Tables 3-9 and 3-21 show row (RREN) and column (CREN) redundancy logic tables.

Table 3-20. Dual-Port SRAM Row Redundancy

RREN	Mode	Function
H	Read or write	Operations are performed as defined in Table 3-17, for basic memory operations. Main memory locations are accessed as specified by A[m-1:0].
L	Redundant row access	Instead of the locations in the basic rows, the redundant rows are accessed as specified by RAA[1:0].

Note

For MUX values of 4, 8, and 16, the values for “p” are 2, 3, and 4, respectively.

Table 3-21. Dual-Port SRAM Column Redundancy

RCEN	Mode	Function
H	Normal	The data is routed from pins (D[i], Q[i]) to the respective bit locations in the memory core.
L	Shift	For the low order bits in a word (0 to bit int(n/2)), data is routed from pins (D[i], Q[i]) to the adjacent bits (i+1) in the memory core. Bit int(n/2) is shifted to a redundant column. For the high order bits in a word, bit (int(n/2)+1 to n-1), data is routed from pins (D[i], Q[i]) to the adjacent bits (j-1) in the memory core. Bit int(n/2)+1 is shifted to a redundant column.

3.3.4 Dual-Port SRAM Parameters

The standard input and block parameters of a synchronous dual-port SRAM are described in Table 3-22. You can also refer to your compiler GUI for the specific ranges for your compiler. If you enter an invalid value and update the GUI, the message pane at the bottom of the GUI displays an error message and the specific range for your compiler.

Table 3-22. Dual-Port SRAM Parameters

Input Parameters		
Parameter	**Ranges**	
number of words	MUX = 4	128 to 2048, increment = $mux \bullet 2$
	MUX = 8	256 to 4096, increment = $mux \bullet 2$
	MUX = 16	512 to 8192, increment = $mux \bullet 2$
number of bits [1]	MUX = 4	ECC = Off; 2 to 144, increment = 1 ECC = On; 2 to 128, increment = 1
	MUX = 8	ECC = Off; 2 to 72, increment = 1 ECC = On; 2 to 64, increment = 1
	MUX = 16	ECC = Off; 2 to 36, increment = 1 ECC = On; 2 to 32, increment = 1
frequency (MHz)	1 to $1/t_{cyc} \bullet 1000$, increment = 1	
word partition size [2]	1 to min (36, bits-1) increment = 1	
top compiler metal layer	m5 to top metal layer supported by design process	
power type	ArtiGrid	
extra margin adjustment	always on	
redundancy	on, off; default off	
redundant columns	0 or 2; redundancy must be on	
redundant rows	0, 2, or 4; redundancy must be on	
BIST MUX	on, off; default off	
Soft Error Repair [3]	none, 1bd1bc, 2bd1bc; default none.	
Back Biasing	on, off	
Power Gating	on, off	
Retention	always on	
Pipeline Register	on, off	
Block Parameters		
Parameter	**Ranges**	
total memory bits	256 to 294,912; total bits = $words \bullet bits$	

Table 3-22. Dual-Port SRAM Parameters (Continued)

rows in memory matrix	MUX = 4 2048/4=512; rows = word depth / mux MUX = 8 4096/8=512; rows = word depth / mux MUX = 16 8192/16=512; rows = word depth / mux	
columns in memory matrix	mux * 4 to 512; increment = mux	
address lines	MUX = 4	7 to 11
	MUX = 8	8 to 12
	MUX = 16	9 to 13

1 Enabling the SER feature for soft-error detection and correction increases the limits beyond the specified number of bits.

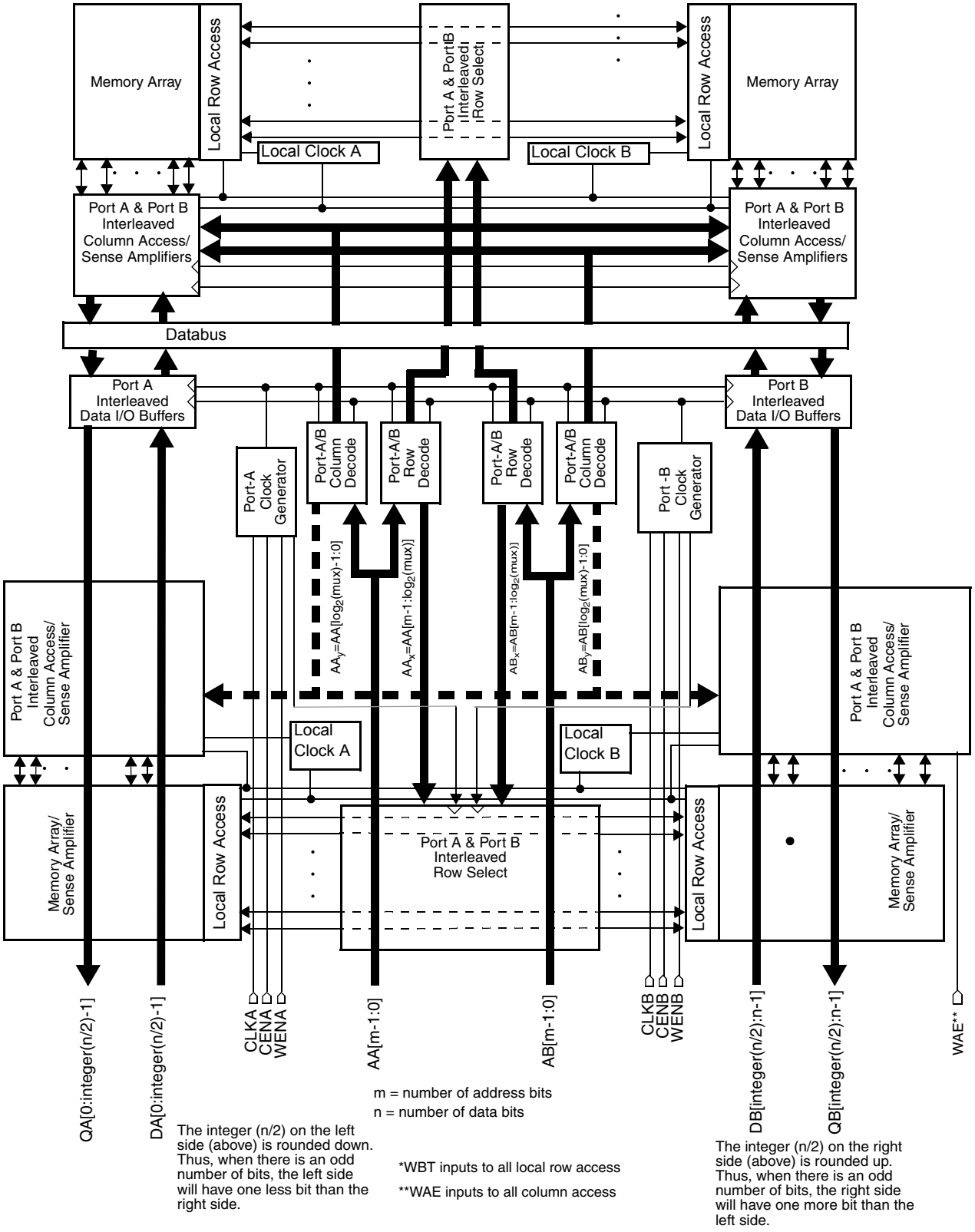
2 The input pin capacitance for each pin of the write enable bus is proportional to the size of the word partition. For example, an instance with bits = 32 and wp_size = 24 will have two partitions, one with 24 bits and one with 8 bits. The write enable pin for the 24 bit partition will have a significantly larger input pin capacitance than the write enable pin for the 8 bit partition. When modelling write enable timing, the write enable pin with the largest capacitance is used in the typical and slow corner timing models. The write enable pin with the smallest capacitance is used in the fast corner timing models. ARM recommends that the critical path, setup, and hold analysis be performed for all corners

3 When the SER feature is enabled and any of the '1bd1bc' or '2bd1bc' options are selected, the actual memory generated will have more bits than specified in the GUI; these extra bits are used to store the error detection and correction code. For example, if you specify 120 bit memory in GUI and use '1bd1bc' SER option, then the generated memory will contain 127 bits. The extra 7 bits are needed for storing the error detection and correction code. So with SER option selected the range of bits specified in the GUI has to be lower then the maximum allowed for a given value of multiplexer. As an example the maximum number of bits allowed for multiplexer = 8 with SER is 120 for a single-port memory instead of 128 with SER option 'none'

3.3.5 Dual-Port SRAM Block Diagrams

The SRAM has two ports for the same memory locations. Both ports can be independently accessed for read or write operations. The two ports function identically. Figures 3-35 and 3-36 show the block diagram for high density dual-port SRAMs.

Figure 3-35. Dual-Port high density SRAM Basic Block Diagram



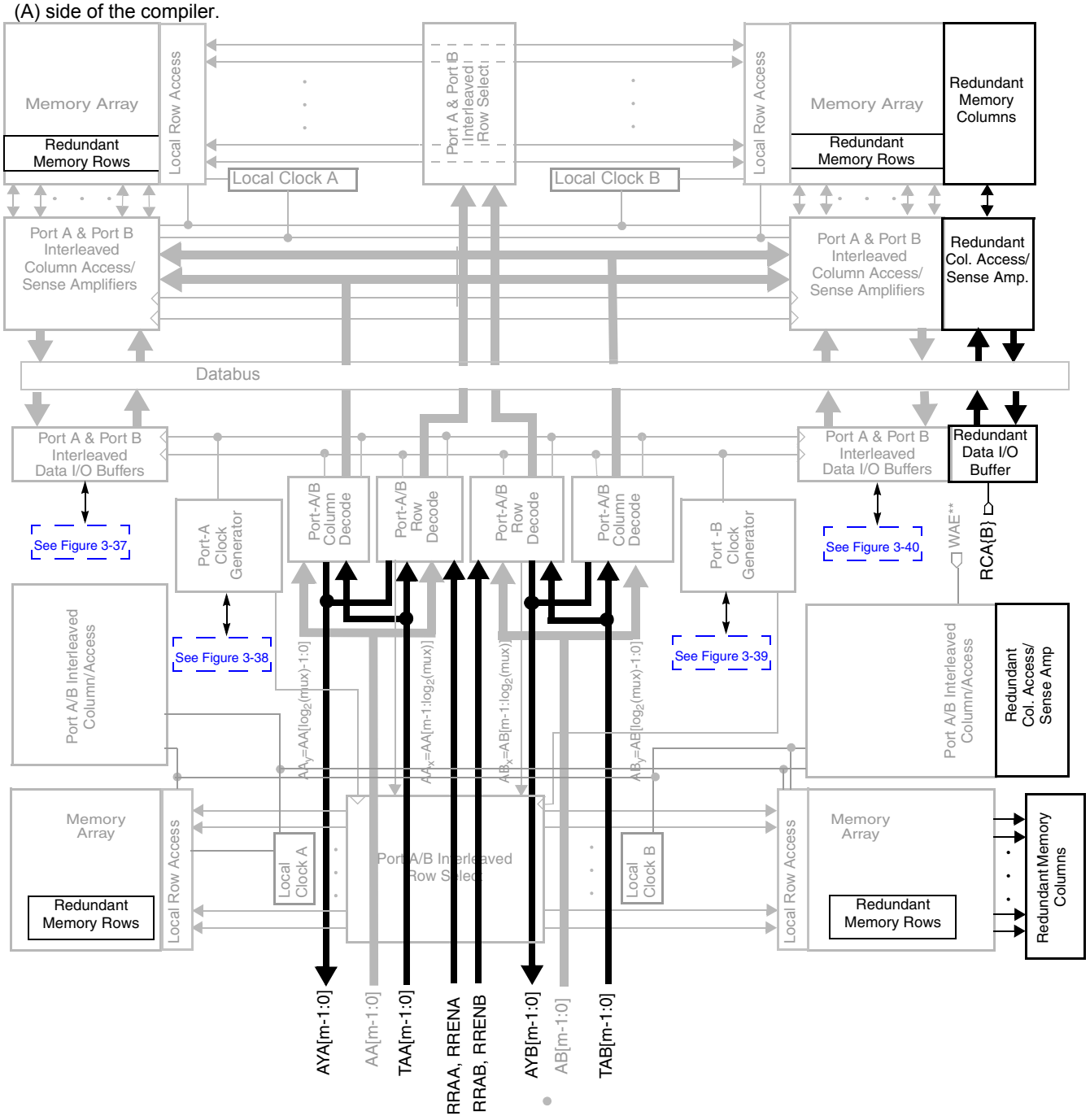
Notes:

Word-Write Mask
When the word-write mask option is turned on, GWENA and GWENB are located in the clock generator area besides the WENA(n:0) and WENB(n:0) bus; this is located at the Data I/O Buffers (not shown in block diagram).

When the word-write mask option is turned off, the WEN{A,B} pin is a signal pin and is located at the Clock Generator, as shown.

Figure 3-36. Dual-Port high density SRAM Basic and Test/Repair Block Diagram

In the actual compiler, the redundant elements illustrated on the far right (B) side of the diagram are also present on the left (A) side of the compiler.



The integer (n/2) on the left side (above) is rounded down. Thus, when there is an odd number of bits, the left side will have one less bit than the right side.

m = number of address bits
n = number of data bits

*WBT inputs to all local row access

**WAE inputs to all column access

The integer (n/2) on the right side (above) is rounded up. Thus, when there is an odd number of bits, the right side will have one more bit than the left side.

Figure 3-37. Left Interleaved Data I/O Buffers Detail

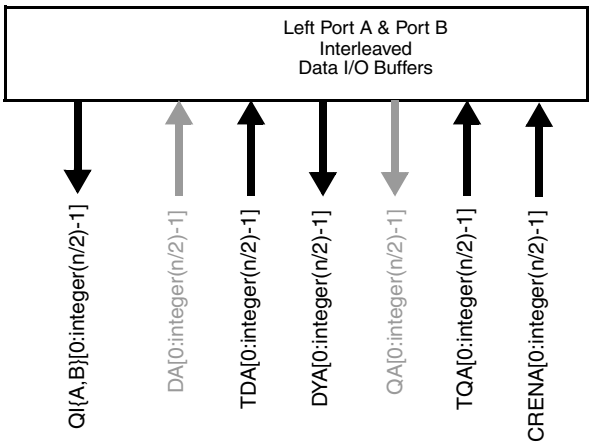


Figure 3-38. Port A Clock Generator Detail

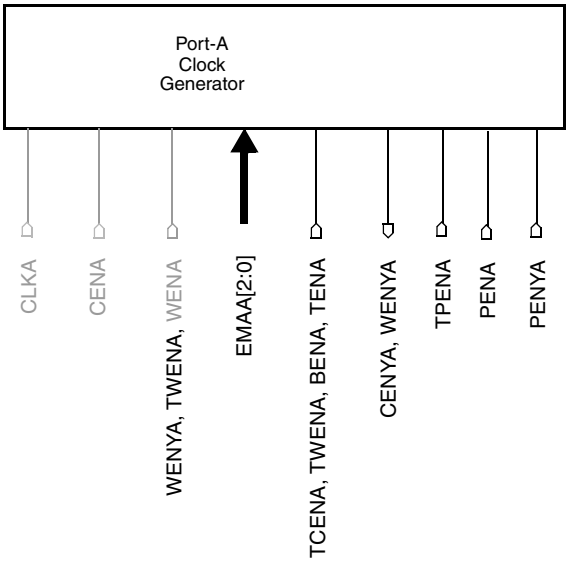


Figure 3-39. Port B Clock Generator Detail

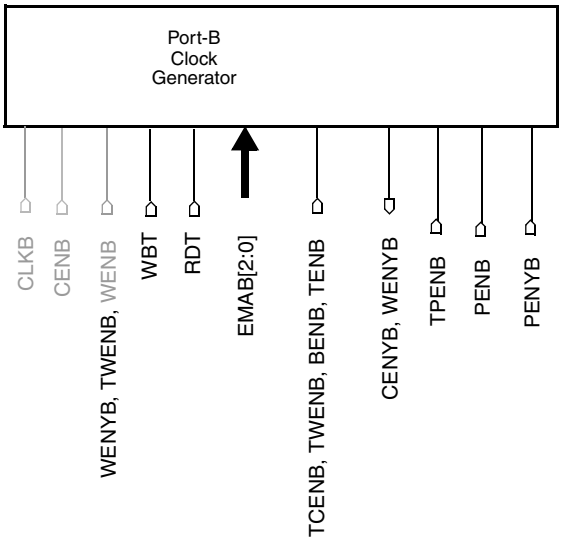
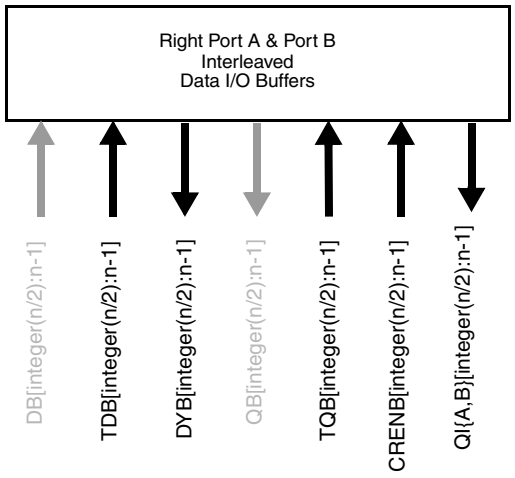


Figure 3-40. Right Interleaved Data I/O Buffers Detail



Notes:

Word-Write Mask
When the word-write mask option is turned on, GWENA and GWENB are located in the clock generator area besides the WENA(n:0) and WENB(n:0) bus; this is located at the Data I/O Buffers (not shown in block diagram).

When the word-write mask option is turned off, the WEN{A,B} pin is a signal pin and is located at the Clock Generator, as shown.

EMA
The EMA option is always turned on; the EMA{A,B} pin is an input bus signal.

BIST MUX
When the BIST MUX option is turned on, the TCEN{A,B}, TEN{A,B}, BEN{A,B}, CENY{A,B}, TD{A,B}, TQ{A,B}, TA{A,B}, DY{A,B}, and AY{A,B} pins are available.

When the BIST MUX option is turned on and:
- the word-write mask option is turned on, the WEN{A,B}, TWEN{A,B}, and WENY{A,B} pins are bus signals and are located at the Data I/O Buffers (not shown in block diagram).
- the word-write mask option is turned off, the WEN{A,B}, TWEN{A,B}, and WENY{A,B} pins are signal pins and are located at the Clock Generator, as shown.

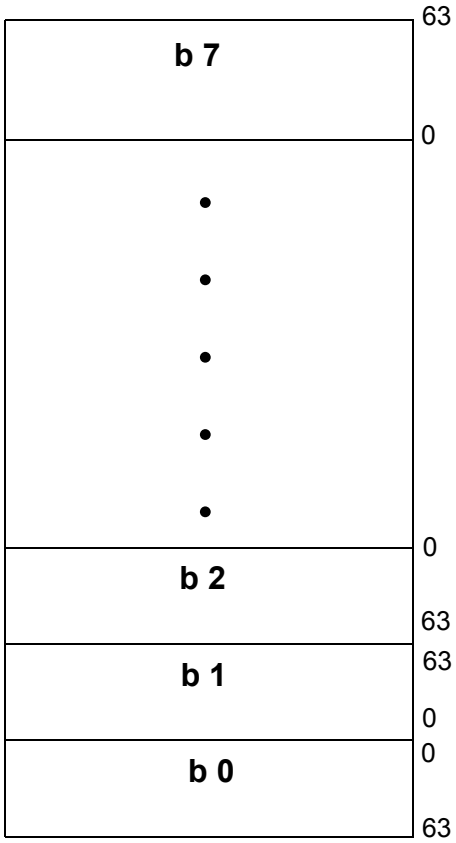
When the BIST MUX option is turned off, the TCEN{A,B}, TEN{A,B}, BEN{A,B}, CENY{A,B}, TD{A,B}, TQ{A,B}, TA{A,B}, DY{A,B}, AY{A,B}, TWEN{A,B}, and WENY{A,B} pins are unavailable.

3.3.6 Dual-Port SRAM Core Address Maps

This section describes the core address diagrams for high density synchronous dual-port SRAMs.

Bit mapping for high density dual-port SRAM is based on developing a bank map for each desired compiler. A typical bank map consists of eight (8) banks and is shown in Figure 3-41.

Figure 3-41. Typical Dual-Port SRAM Bank Map

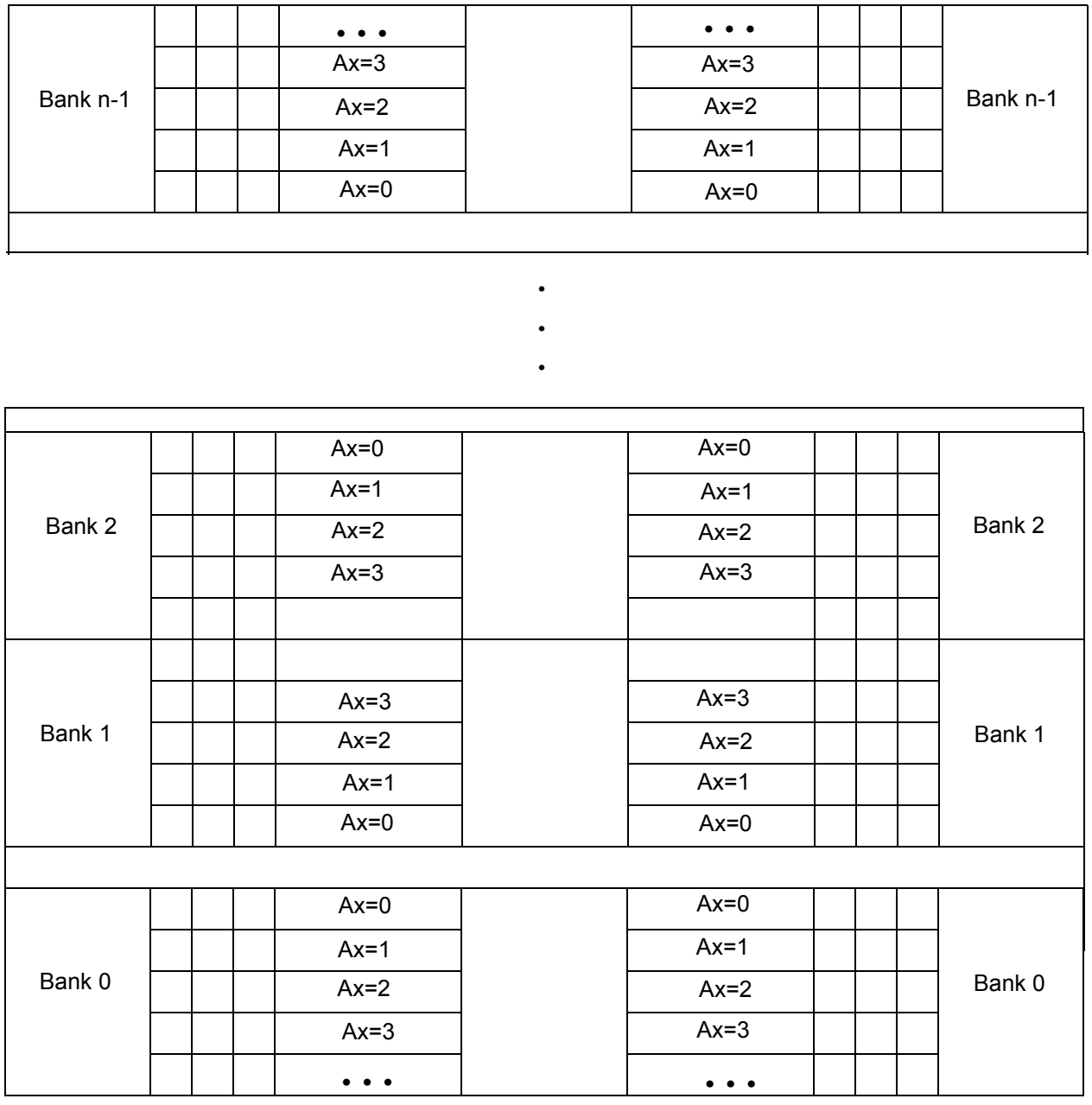


Two (2) to three (3) MSB bits are used to initially define/select a bank map.

Each compiler is mapped by completing a row or column in one bank then moving on to other banks in a given sequence.

Figure 3-42 shows the typical structure of rows in the banks. A column arrangement is similar but vertically arranged and not linear as is the row structure.

Figure 3-42. Typical Dual-Port SRAM Row Structure



An example of the standard physical core mapping for single-port SRAM mux values is shown in Figures 3-43 through 3-45.

Figure 3-43. Dual-Port SRAM Mux 4: Core Address Mapping

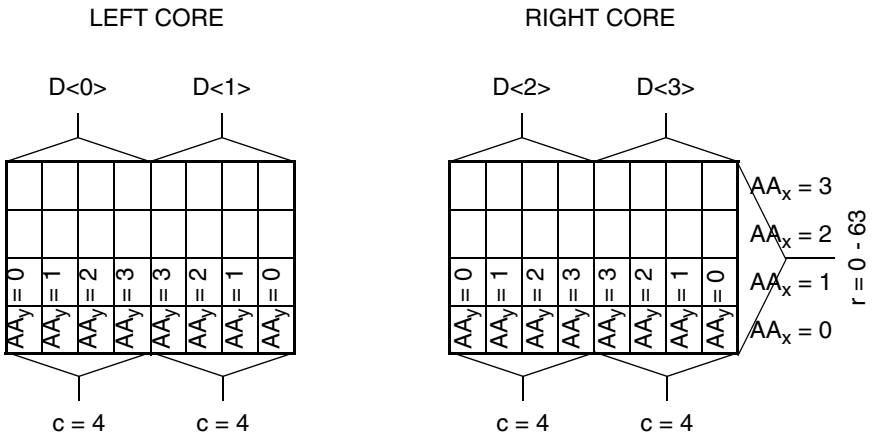


Figure 3-44. Dual-Port SRAM Mux 8: Core Address Mapping

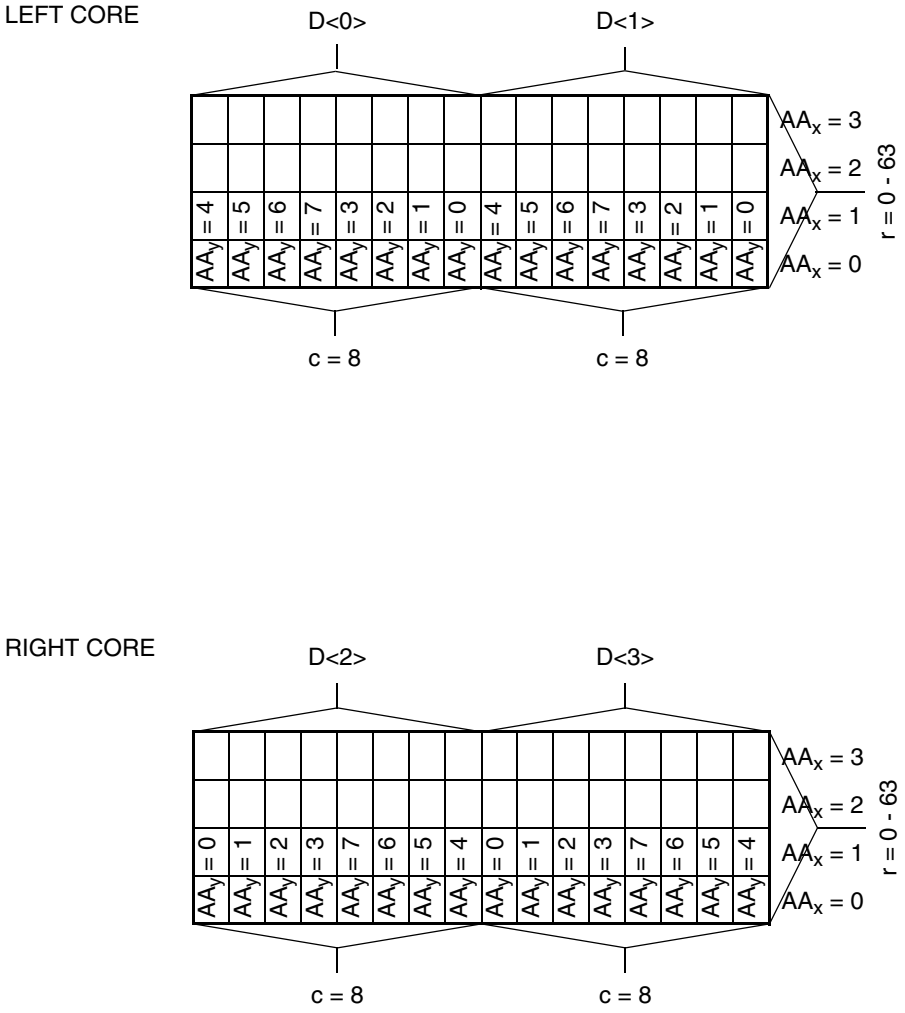
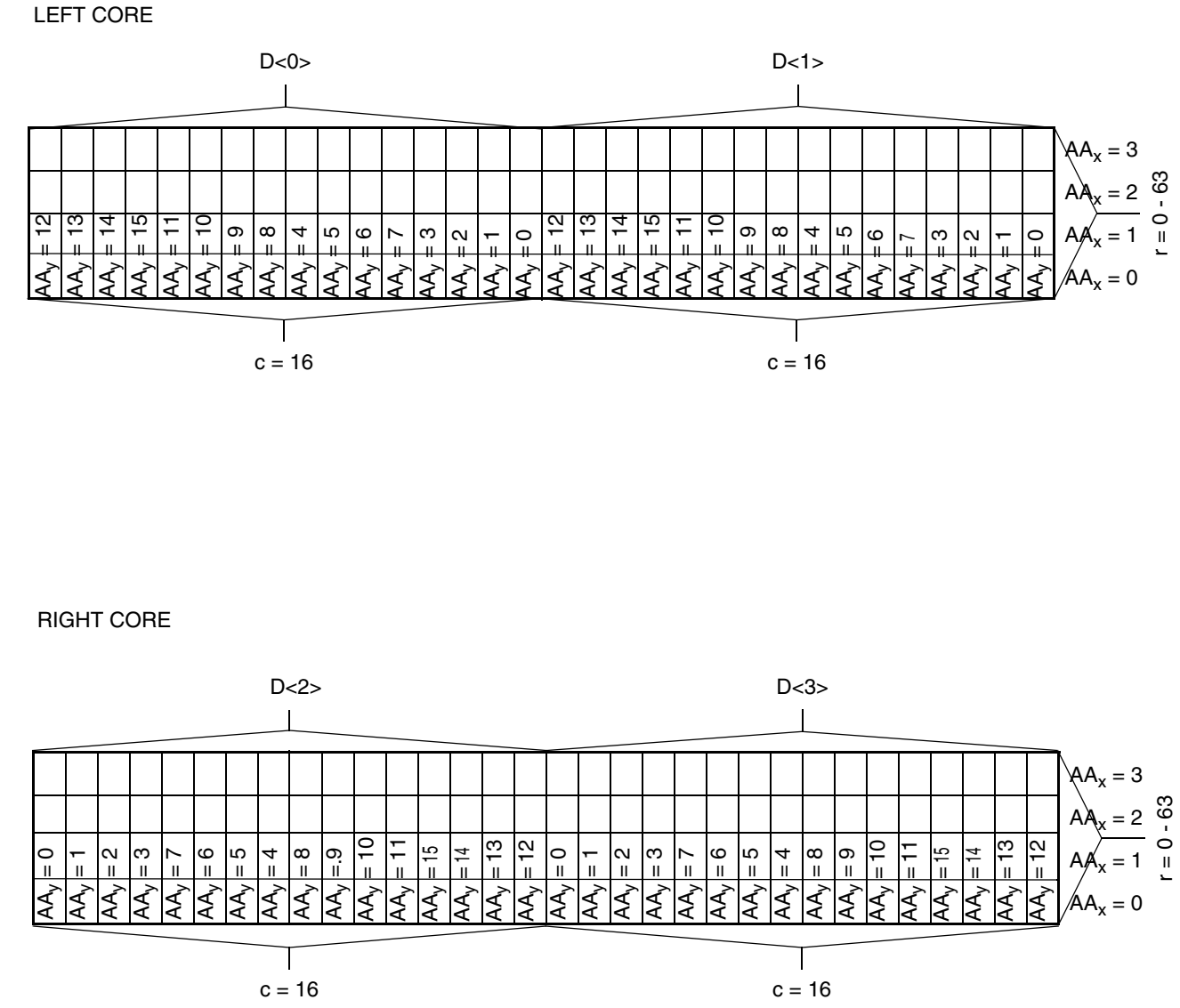


Figure 3-45. Dual-Port SRAM Mux 16: Core Address Mapping



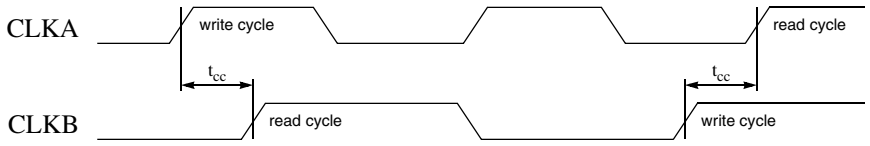
3.3.7 Dual-Port SRAM Timing Specifications

This section contains timing diagrams, timing parameters, and power parameters for the synchronous dual-port SRAMs.

3.3.7.1 Dual-Port SRAM Clock Timing Diagrams

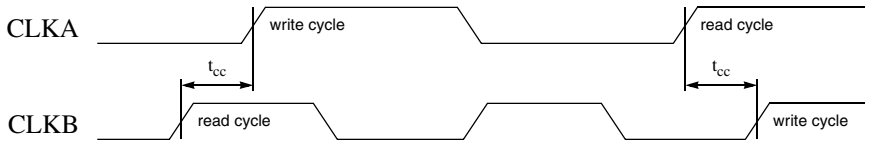
Figures 3-46 through 3-48 show clock timing diagrams for high density synchronous dual-port SRAMs. Standard rising/falling delays and slews percentages are shown in these diagrams. Some compilers may be designed with different percentages. Check a GUI generated postscript datasheet to verify the delay values for a particular instance.

Figure 3-46. Dual-Port SRAM Write-Read Clock Timing (Accessing Same Address)



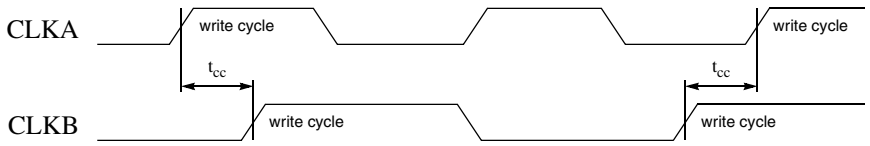
Rising delays are measured at 50% VDD and falling delays are measured at 50% VDD.
Rising and falling slews are measured from 10% VDD to 90% VDD.

Figure 3-47. Dual-Port SRAM Read-Write Clock Timing (Accessing Same Address)



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.

Figure 3-48. Dual-Port SRAM Write-Write Clock Timing (Accessing Same Address)



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.

Table 3-23 illustrates read and write behavior during clock contention, when both ports access the same address.

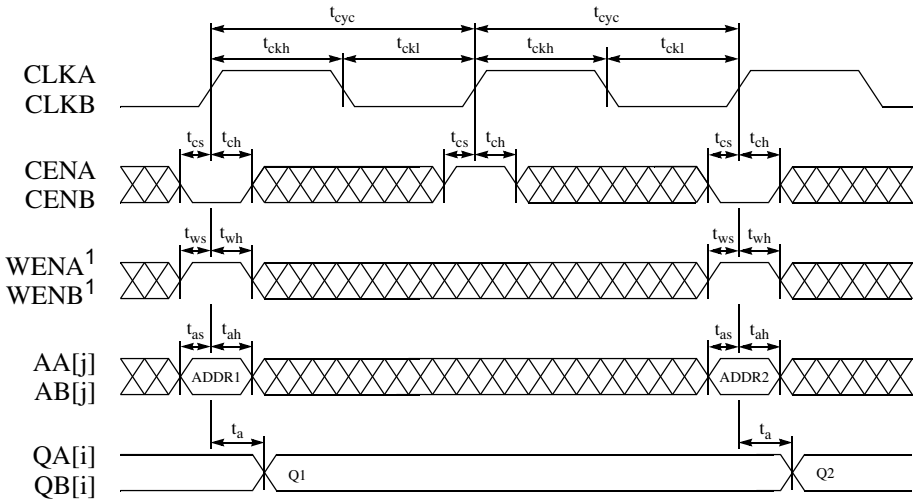
Table 3-23. Dual-Port SRAM Read and Write Behavior When Accessing Same Address

Action	Condition	Behavior
write from one port then read from the other port	t_{cc} is satisfied (see Figure 3-46)	write OK D-to-Q write through OK read (new data) OK
	t_{cc} is not satisfied (see Figure 3-46)	If DPCCM = ON write succeeds D-to-Q write through OK for write port read port produces an X at the output If DPCCM = OFF write fails, an X is placed in the memory location indicated by the address on the two ports D-to-Q write through OK for write port read port produces an X at the output
read from one port, followed by write from the other port	t_{cc} is satisfied (see Figure 3-47)	write OK D-to-Q write through OK read (old data) OK
	t_{cc} is not satisfied (see Figure 3-47)	If DPCCM = ON write succeeds D-to-Q write through OK for write port read port produces an X at the output If DPCCM = OFF write fails, an X is placed in the memory location indicated by the address on the two ports D-to-Q write through OK for write port read port produces an X at the output
write from one port then write from the other port	t_{cc} is satisfied (see Figure 3-48)	both writes OK (second write overwrites first write) D-to-Q write throughs OK
	t_{cc} is not satisfied (see Figure 3-48)	both writes fail, an X is placed in the memory location indicated by the address on the two ports D-to-Q write throughs OK
read from one port then read from the other port	no restriction	both reads OK

3.3.7.2 Dual-Port SRAM Timing Diagrams

Figures 3-49 and 3-50 show timing diagrams for high density dual-port SRAMs. Standard rising/falling delays and slews percentages are shown in these diagrams. Some compilers may be designed with different percentages. Check a GUI generated postscript datasheet to verify the delay values for a particular instance.

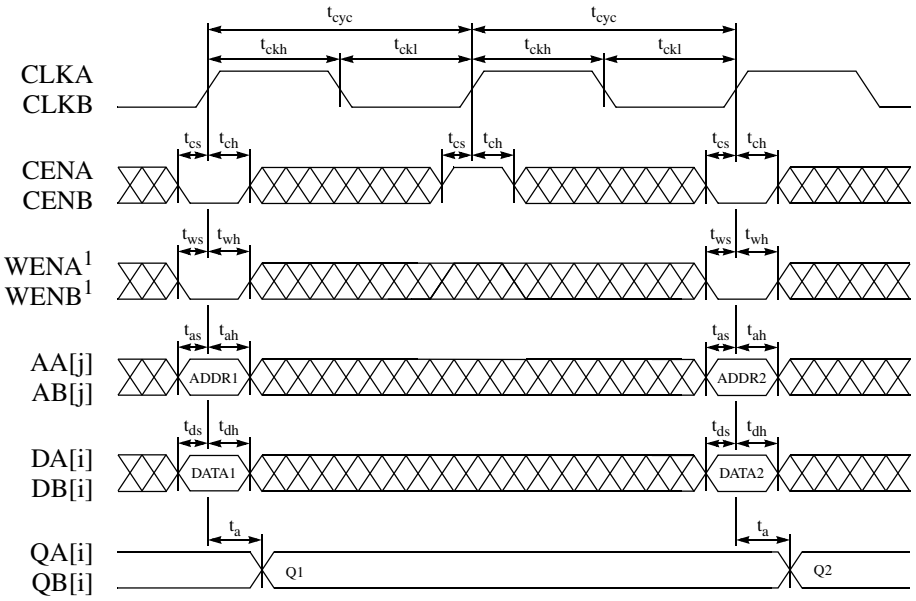
Figure 3-49. Dual-Port SRAM Read-Cycle Timing



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.

[1] When word-write mask is turned off, WEN is a signal pin as shown in this diagram.
When word-write mask is turned on, WEN is a bus that is ignored. GWEN is a signal pin as shown in the diagram.

Figure 3-50. Dual-Port SRAM Write-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.

[1] When word-write mask is turned off, WEN is a signal pin as shown in this diagram.
When word-write mask is turned on, WEN is a bus and GWEN is a signal pin as shown in the diagram.

3.3.7.3 Dual-Port SRAM Timing Parameters

The GUI generated postscript datasheets and the ASCII datatable contain timing parameters listed in Table 3-24.

Table 3-24. Dual-Port SRAM 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}
Write enable setup	t_{ws}
Write enable hold	t_{wh}
Data setup	t_{ds}
Data hold	t_{dh}
Clock high (minimum pulse width)	t_{ckh}
Clock low (minimum pulse width)	t_{ckl}
Clock rise slew (maximum transition time)	t_{ckr}
Clock collision	t_{cc}
Access time, EMA is enabled: eight numbers for eight values of EMA[1,2]	$t_{a[0-7]}$
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}
Write enable setup, test pin	t_{tws}
Write enable hold, test pin	t_{twh}
Data setup, test pin	t_{tds}
Data hold, test pin	t_{tdh}
Propagation delay BEN to output	t_{benq}

Table 3-24. Dual-Port SRAM Timing Parameters (Continued)

Parameter	Symbol
Bypass enable setup	t_{bens}
Bypass enable hold	t_{benh}
Extra margin enable pin setup	t_{emas}
Extra margin enable pin hold	t_{emah}
Column redundancy enable setup	t_{crens}
Column redundancy enable hold	t_{crenh}
Redundant column address setup	t_{rcas}
Redundant column address hold	t_{rcah}
Row redundancy enable setup	t_{rrens}
Row redundancy enable hold	t_{rrenh}
Row redundancy address setup	t_{rras}
Row redundancy address hold	t_{rrah}
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 write enable MUX output (ns/pF)	load_weny
Load dependence factor on address MUX output (ns/pF)	load_ay
Load dependence factor on data MUX output (ns/pF)	load_dy

1 The ASCII datatable and postscript datasheet shows fixed delay values. These parameters have a load dependence (K_{load}), which is used to calculate: $\mathrm{TotalDelay} = \mathrm{FixedDelay} + (\mathrm{Kload} \times \mathrm{Cload})$, 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.

3.3.7.4 Dual-Port SRAM Power Parameters

The GUI contains an ASCII datatable that provides characterization values for each corner. These values are also available in a generated postscript datasheet. AC current, AC read/write current, peak current and deselected current values include a DC leakage component equal to the standby current. Table 3-25 shows the dual-port SRAM power parameters.

Table 3-25. Dual-Port SRAM Power Parameters (changes pending)

Parameter	Symbol
AC Current [1, 2]	i_{cc}
Read AC Current [2]	i_{cc_r}
Write AC Current [2]	i_{cc_w}
Peak Current	i_{cc_peak}
Deselected Current [2, 3]	i_{cc_desel}
Standby Current [4]	$i_{cc\_standby}$
AC Current: eight numbers for eight values of EMA	$i_{cc[0:7]}$
Read AC Current: eight numbers for eight values of EMA	$i_{cc\_r[0:7]}$
Write AC Current: eight numbers for eight values of EMA	$i_{cc\_w[0:7]}$

1 Value assumes 50% read and write operations, where all addresses and 50% of input and output pins switch. This value is an average of the read and write current (i_{cc_r}, i_{cc_w}) values.

2 For sram_dp_hdc_svt_rvt_hvt, value shows dynamic current, without leakage (standby) component.

3 Value assumes SRAM is deselected, all addresses switch, and 50% of data 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.

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

The current values shown in datasheets and datatables are based on certain assumptions. Refer to “Current Calculations” on page 3-85 for instructions on recalculating the current for a specific design. Refer to “Noise Limits” on page 3-88 for more information related to power considerations.

3.4 SRAM Power Structure

The following sections explain the power structure options available with single- and dual-port, high density SRAM compilers.

3.4.1 Current Calculations

The average current, I_{avg}, in mA, for the SRAM instance may be calculated from data reported in the ASCII datatable, as well as the datasheet. The average current reported in the datasheet assumes 50% read and write operations where all addresses and 50% of input and output pins [unloaded] switch. You may choose to recalculate this number based on the percentage of reads and writes in a given design. This value is used to calculate the size of the power bus.

Given:

c = average capacitance of output port (pF);

n = number of ports;

From the datatable,

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

From the datasheet,

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

The peak current, I_p, in mA, for the instance is reported in the datasheet and ASCII datatable. This peak current is measured during read/write 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 memory 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

NOTE: When the SRAM is deselected, all addresses switch, and 50% of data 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 = Peak Current

3.4.2 Power Distribution Methodology

Your chip-level power distribution must ensure that the wire widths suppling power to the SRAM satisfy electromigration guidelines and limit the average and peak voltage drop in the power wires to an acceptable value. To ensure memory timing accuracy, the voltage supplied to the memory must be the same as the characterized voltage. The SRAM 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 (mA/μm);

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

I_p = peak current consumed by the SRAM (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 SRAM (µm);

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

W = connection width (µm);

we have:

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

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

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

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

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.

3.4.3 Noise Limits

The characterized clock noise limit, vn_ck, 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, pwn_ck, used in characterizing this limit is 10ns.

The power and ground noise limits, vn_pwr and vn_gnd respectively, are the maximum supply or ground voltage transition allowable without causing a memory failure. Power and ground noise limits are assured at 10% of the characterized voltage.

3.5 ArtiGrid Power Structure Options

Cadence Nanoroute and Synopsys Astro router verify that the horizontal M4 straps can be connected to from M5 and M6. ArtiGrid Over-the-Cell (OTC) has only the horizontal M4 straps over the instance. You must route VSS and VDD vertically in M5 or higher and drop vias down to M4 to connect to every power/ground strap seen. OTC parameter selections are shown in Table 3-26.

Table 3-26. OTC Parameter Selections

Parameter	Value
Minimum width and height for VDD and GND Metal4 straps	top metal via4 plus m4 overlap
Minimum connection/stripe to maintain power density	X (don’t care)

3.5.1 Over the Cell (OTC) Power Routing Diagrams

Figure 3-51. Over-the-Cell Power Routing - Standard Option

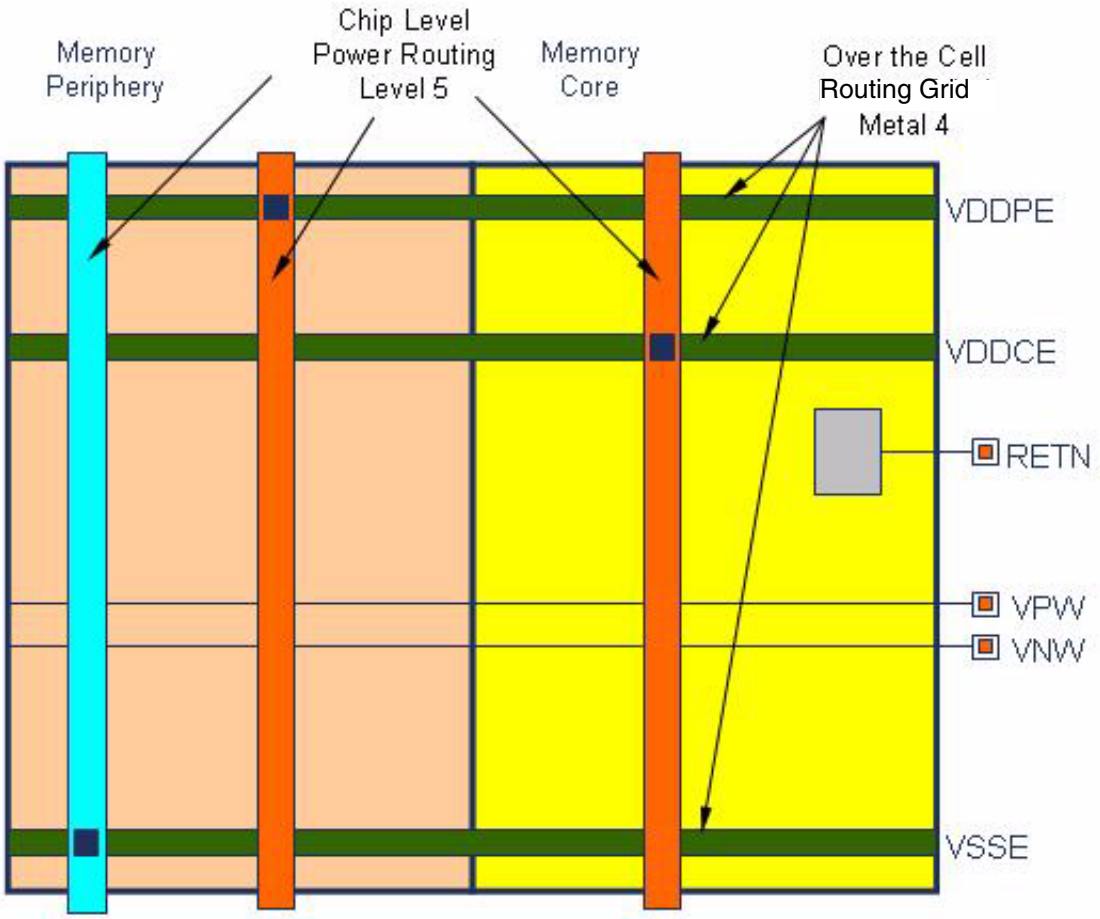
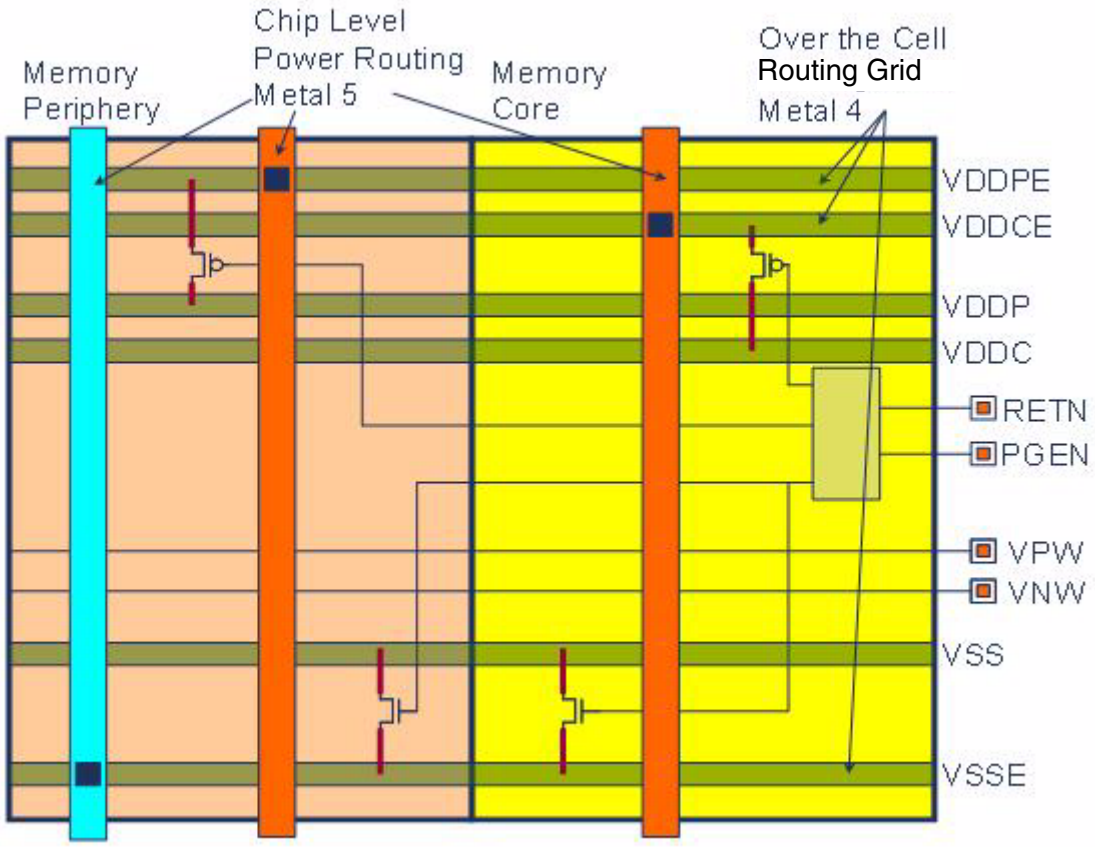


Figure 3-52. Over-the-Cell Power Routing - Power Gating Option



3.6 SRAM Physical Characteristics

This section provides physical design characteristics for single- and dual-port, high density SRAM compilers. Information such as top metal layer usage, I/O pin connections, ArtiGrid/Over-the-Cell (OTC) routing, and characterization environments is also provided in this section.

3.6.1 Top Metal Layer

The compiler has the capability of supporting different top-most metal layer designs. For example, a process may support a maximum of eight 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 including metal4 and below are used in the design, and therefore, are blocked for routing. All metal layers above metal4 can be routed over the memory.

3.6.2 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.

EMA pins are exception to the usual pin placement locations. These pins are located at the top edge of the memory.

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.

In some configurations it may not be possible to fit all the pins on metals1-4 using wide sizes. In these cases, wide pins are allowed on metals1-3. If neither of these cases are possible due to pin-density, the thin pins are created on metals1-3 with at least one routing pitch spacing between the pins.

3.6.3 Characterization Environments

By default, the compiler is characterized as fast (ff), typical (tt), and slow (ss) operating conditions or corners, and also at a high leakage corner. Your compiler may have more corners, or a different set of corners. Only four corners are visible in the ASCII datatable of the compiler GUI at one time. Similarly, only four corners are available at one time in the postscript datasheet.

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.

3.7 SRAM Timing Derating

Derating factors are coefficients that the characterization data is multiplied by in order to arrive at timing data that reflects different operating conditions. Standard ARM Artisan memories do not support a timing derating methodology. By default, timing is provided for three characterization environments: fast, 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
Compiler Views

This chapter contains the following sections:

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 SRAM compilers unless otherwise noted.

4.2 Tool Verification

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

Standard Support		
Verilog (.v)	NC-Verilog	Cadence
	VCS	Synopsys
	ModelSim (Verilog)	MTI/Mentor
Synopsys (Liberty) (.lib)	Library Compiler	Synopsys
	RTL Compiler	Cadence
	SOC Encounter	
LEF (.vclef)	SOC Encounter	Cadence
Optional Support		
TetraMAX (.tv)	TetraMAX Model	Synopsys
CDL	Calibre LVS	Mentor
	Hercules LVS	Synopsys
GDS II	Calibre LVS	Mentor
	Calibre DRC	Mentor
	Hercules LVS	Synopsys
Extended Views		
User-provided files	CeltIC Enablement	Cadence

4.3 Using the Memory Instance 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 Fast Cache 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

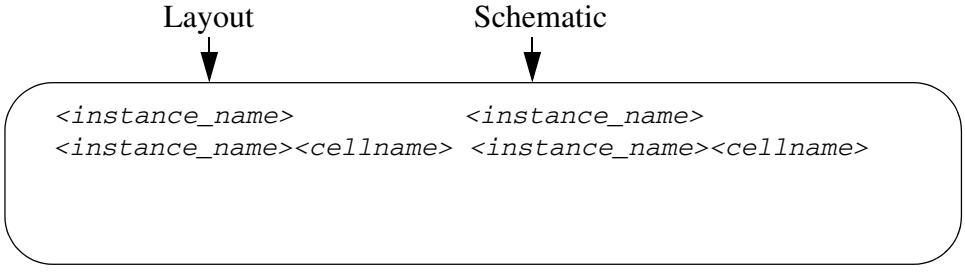
LVS (Layout vs. Schematic) performs an equivalence check between two different representations of a design. In this case, the physical (GDSII) and schematic (SPICE netlist) are compared to each other. The Calibre tool reports any discrepancies.

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 synatx
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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

Appendix A- Revisions

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

Table A-1. Issue A

Change	Location	Affects
First release for r0p0	-	-