

# CoreLink™ DMC-400 Dynamic Memory Controller Cycle Model

Version 9.1.0

**User Guide**



# CoreLink DMC-400 Dynamic Memory Controller

## User Guide

Copyright © 2017 ARM Limited. All rights reserved.

### Release Information

The following changes have been made to this document.

Change History			
Date	Issue	Confidentiality	Change
February 2017	A	Non-Confidential	Restamp release.

### Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM Limited (“ARM”). **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, ARM makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version shall prevail.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to ARM’s customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement specifically covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. You must follow the ARM trademark usage guidelines <http://www.arm.com/about/trademarks/guidelines/index.php>.

Copyright © ARM Limited or its affiliates. All rights reserved.  
ARM Limited. Company 02557590 registered in England.  
110 Fulbourn Road, Cambridge, England CB1 9NJ.

In this document, where the term ARM is used to refer to the company it means “ARM or any of its subsidiaries as appropriate”.

**Confidentiality Status**

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

**Product Status**

The information in this document is final, that is for a developed product.

**Web Address**

<http://www.arm.com>



# Contents

## Preface

About This Guide .....	3
Audience .....	3
Conventions .....	4
Further reading .....	5
Glossary .....	6

## Chapter 1.

### Using the Cycle Model in SoC Designer

DMC-400 Memory Controller Cycle Model Functionality .....	2
Supported Features .....	2
Unsupported Hardware Features .....	2
Features Additional to the Hardware .....	2
Adding and Configuring the SoC Designer Component .....	3
SoC Designer Component Files .....	3
Adding the Cycle Model to the Component Library .....	4
Adding the Component to the SoC Designer Canvas .....	4
Available Component ESL Ports .....	5
Setting Component Parameters .....	6
Debug Features .....	8
Register Information .....	8
DMC Configuration Registers .....	8
Memory Registers .....	10
Memory Information .....	11
Available Profiling Data .....	12



# Preface

A Cycle Model component is a library developed from ARM intellectual property (IP) that is generated through Cycle Model Studio™. The Cycle Model then can be used within a virtual platform tool, for example, SoC Designer.

## About This Guide

This guide provides all the information needed to configure and use the Cycle Model in SoC Designer.

## Audience

This guide is intended for experienced hardware and software developers who create components for use with SoC Designer. You should be familiar with the following products and technology:

- SoC Designer
- Hardware design verification
- Verilog or SystemVerilog programming language

## Conventions

This guide uses the following conventions:

Convention	Description	Example
<code>courier</code>	Commands, functions, variables, routines, and code examples that are set apart from ordinary text.	<code>sparseMem_t SparseMemCreateNew();</code>
<i>italic</i>	New or unusual words or phrases appearing for the first time.	<i>Transactors</i> provide the entry and exit points for data ...
<b>bold</b>	Action that the user performs.	Click <b>Close</b> to close the dialog.
<text>	Values that you fill in, or that the system automatically supplies.	<platform>/ represents the name of various platforms.
[ text ]	Square brackets [ ] indicate optional text.	\$CARBON_HOME/bin/modelstudio [ <filename> ]
[ text1   text2 ]	The vertical bar   indicates “OR,” meaning that you can supply text1 or text 2.	\$CARBON_HOME/bin/modelstudio [<name>.symtab.db   <name>.ccfg ]

Also note the following references:

- References to C code implicitly apply to C++ as well.
- File names ending in .cc, .cpp, or .cxx indicate a C++ source file.



## Further reading

This section lists related publications. The following publications provide information that relate directly to SoC Designer:

- *SoC Designer Installation Guide*
- *SoC Designer User Guide*
- *SoC Designer Standard Component Library Reference Manual*

The following publications provide reference information about ARM® products:

- *AMBA 3 AHB-Lite Overview*
- *AMBA Specification (Rev 2.0)*
- *AMBA AHB Transaction Level Modeling Specification*
- *Architecture Reference Manual*

See <http://infocenter.arm.com/help/index.jsp> for access to ARM documentation.

The following publications provide additional information on simulation:

- IEEE 1666™ SystemC Language Reference Manual, (IEEE Standards Association)
- SPIRIT User Guide, Revision 1.2, SPIRIT Consortium.

## Glossary

AMBA	<i>Advanced Microcontroller Bus Architecture.</i> The ARM open standard on-chip bus specification that describes a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC).
AHB	<i>Advanced High-performance Bus.</i> A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol.
APB	<i>Advanced Peripheral Bus.</i> A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports.
AXI	<i>Advanced eXtensible Interface.</i> A bus protocol that is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.
Cycle Model	A software object created by the Cycle Model Studio (or <i>Cycle Model Compiler</i> ) from an RTL design. The Cycle Model contains a cycle- and register-accurate model of the hardware design.
Cycle Model Studio	Graphical tool for generating, validating, and executing hardware-accurate software models. It creates a Cycle Model, and it also takes a Cycle Model as input and generates a component that can be used in SoC Designer, Platform Architect, or Accellera SystemC for simulation.
CASI	<i>ESL API Simulation Interface</i> , is based on the SystemC communication library and manages the interconnection of components and communication between components.
CADI	<i>ESL API Debug Interface</i> , enables reading and writing memory and register values and also provides the interface to external debuggers.
CAPI	<i>ESL API Profiling Interface</i> , enables collecting historical data from a component and displaying the results in various formats.
Component	Building blocks used to create simulated systems. Components are connected together with unidirectional transaction-level or signal-level connections.
ESL	<i>Electronic System Level.</i> A type of design and verification methodology that models the behavior of an entire system using a high-level language such as C or C++.
HDL	<i>Hardware Description Language.</i> A language for formal description of electronic circuits, for example, Verilog.
RTL	<i>Register Transfer Level.</i> A high-level hardware description language (HDL) for defining digital circuits.
SoC Designer	High-performance, cycle accurate simulation framework which is targeted at System-on-a-Chip hardware and software debug as well as architectural exploration.
SystemC	SystemC is a single, unified design and verification language that enables verification at the system level, independent of any detailed hardware and software implementation, as well as enabling co-verification with RTL design.
Transactor	<i>Transaction adaptors.</i> You add transactors to your component to connect your component directly to transaction level interface ports for your particular platform.

# Chapter 1

## Using the Cycle Model in SoC Designer

This chapter describes the functionality of the Cycle Model, and how to use it in SoC Designer. It contains the following sections:

- [DMC-400 Memory Controller Cycle Model Functionality](#)
- [Adding and Configuring the SoC Designer Component](#)
- [Available Component ESL Ports](#)
- [Setting Component Parameters](#)
- [Debug Features](#)
- [Available Profiling Data](#)

## 1.1 DMC-400 Memory Controller Cycle Model Functionality

The DMC-400 Cycle Model is a high-performance, area-optimized SDRAM or Mobile SDR memory controller compatible with the Advanced Microcontroller Bus Architecture (AMBA) AXI protocol. For a detailed description of the AXI protocol refer to the *AMBA AXI Protocol Specification*.

This section provides a summary of the functionality of the Cycle Model compared to that of the hardware, and the performance and accuracy of the Cycle Model. For details of the functionality of the hardware that the Cycle Model represents, refer to the *ARM CoreLink DMC-400 Dynamic Memory Controller Technical Reference Manual*.

The following topics are discussed in this section:

- [Supported Features](#)
- [Unsupported Hardware Features](#)
- [Features Additional to the Hardware](#)

### 1.1.1 Supported Features

This Beta release of the ARM CoreLink DMC-400 Dynamic Memory Controller (PL443) supports the following:

- DDR3 memory
- AXI3 slave interface
- 1:1 or 1:2 clock ratio

### 1.1.2 Unsupported Hardware Features

The following hardware features have not been implemented in the Cycle Model:

- AXI low-power interface
- DFT
- ECC interfaces and registers
- QVN support

### 1.1.3 Features Additional to the Hardware

The following features that are implemented in the DMC-400 Cycle Model to enhance usability do not exist in the ARM DMC-400 hardware:

- The DMC-400 Cycle Model has the memory built into the Cycle Model, so you do not need to provide a memory.
- Debug and profiling features. For further information about debug and profiling features, refer to [“Available Profiling Data”](#) on page 1-12.

## 1.2 Adding and Configuring the SoC Designer Component

The following topics briefly describe how to use the component. See the *SoC Designer User Guide* for more information.

- [SoC Designer Component Files](#)
- [Adding the Cycle Model to the Component Library](#)
- [Adding the Component to the SoC Designer Canvas](#)

### 1.2.1 SoC Designer Component Files

The component files are the final output from the Cycle Model Studio compile and are the input to SoC Designer. There are two versions of the component; an optimized *release* version for normal operation, and a *debug* version.

On Linux the *debug* version of the component is compiled without optimizations and includes debug symbols for use with gdb. The *release* version is compiled without debug information and is optimized for performance.

On Windows the *debug* version of the component is compiled referencing the debug runtime libraries, so it can be linked with the debug version of SoC Designer. The *release* version is compiled referencing the release runtime library. Both release and debug versions generate debug symbols for use with the Visual C++ debugger on Windows.

The provided component files are listed below:

**Table 1-1 SoC Designer Component Files**

Platform	File	Description
Linux	maxlib.lib<model_name>.conf	SoC Designer configuration file
	lib<component_name>.mx.so	SoC Designer component runtime file
	lib<component_name>.mx_DBG.so	SoC Designer component debug file
Windows	maxlib.lib<model_name>.windows.conf	SoC Designer configuration file
	lib<component_name>.mx.dll	SoC Designer component runtime file
	lib<component_name>.mx_DBG.dll	SoC Designer component debug file

Additionally, this User Guide PDF file is provided with the component.

## 1.2.2 Adding the Cycle Model to the Component Library

The compiled Cycle Model component is provided as a configuration file (*.conf*). To make the component available in the Component Window in SoC Designer Canvas, perform the following steps:

1. Launch SoC Designer Canvas.
2. From the *File* menu, select **Preferences**.
3. Click on **Component Library** in the list on the left.
4. Under the *Additional Component Configuration Files* window, click **Add**.
5. Browse to the location where the SoC Designer Cycle Model is located and select the component configuration file:
  - `maxlib.lib<model_name>.conf` (for Linux)
  - `maxlib.lib<model_name>.windows.conf` (for Windows)
6. Click **OK**.
7. To save the preferences permanently, click the **OK & Save** button.

The component is now available from the SoC Designer *Component Window*.

## 1.2.3 Adding the Component to the SoC Designer Canvas

In SoC Designer, locate the component in the *Component Window* and drag it out to the Canvas.

## 1.3 Available Component ESL Ports

The DMC-400 component has an APB transaction slave port, an AXI transaction slave port and additional signal slave ports as shown below. The APB port is for configuring the component, whereas the AXI port is for accessing the memory. Table 1-2 describes the ESL ports that are exposed in SoC Designer.

All pins that are not listed in this table have been either tied or disconnected for performance reasons. See the *DMC-400 Technical Reference Manual* for more information.

**Table 1-2 ESL Component Ports**

ESL Port	Description	Direction	Type
AceLite_<n> <sup>1, 2</sup>	ACE Lite port for memory accesses. Refer to section A.2 of the <i>DMC-400 Technical Reference Manual</i> for the ACE Lite signal list.	Input	AXI transaction slave
apb	APB port for memory mapped register accesses. Refer to section A.4 of the <i>DMC-400 Technical Reference Manual</i> for the APB signal list.	Input	APB transaction slave
arap_0	Read auto-precharge policy signal.	Input	Signal
awap_0	Write auto-precharge policy signal.	Input	Signal
AXI_s_<n> <sup>1, 3</sup>	AXI port for memory accesses.	Input	AXI transaction slave
clk-in	Clock slave port.	Input	Clock slave
cwakeup_s<n> <sup>1</sup>	Wakeup signal for the clock domain of the system interface.	Input	Signal
cwakeup_m<m> <sup>4</sup>	Wakeup signal for the clock domain of the memory interface.	Input	Signal
dmc_clk	Clock slave port.	Input	Clock slave
dmc_resetr	Reset slave port.	Input	Reset signal
ev_bus_valid_s<n> <sup>1</sup>	System Interface PMV event valid bus.	Output	Signal
ev_bus_valid_m<m> <sup>4</sup>	Memory Interface PMV event valid bus.	Output	Signal
ev_bus_payload_s<n> <sup>1</sup>	SystemInterface PMU event payload bus.	Output	Signal
ev_bus_payload_m<m> <sup>4</sup>	Memory Interface PMU event payload bus.	Output	Signal
MEM_CLK	Clock slave port.	Input	Clock slave
MEM_CLK2	Clock slave port for clock running at 2x speed.	Input	Clock slave
user_config0	User-defined output.	Output	Signal
user_config1	User-defined output.	Output	Signal
user_status	User-defined inputs.	Input	Signal

1. Where <n> indicates the slave interface (0 to 3).

2. Only for ACELite configurations.

3. Only for AXI3 configurations.

4. Where <m> represents memory interface 0 or 1.

## 1.4 Setting Component Parameters

You can change the settings of all the component parameters in SoC Designer Canvas, and of some of the parameters in SoC Designer Simulator. To modify the Cycle Model parameters:

1. In the Canvas, right-click on the Cycle Model and select **Component Information**. You can also double-click the component. The *Edit Parameters* dialog box appears.
2. In the *Parameters* window, double-click the *Value* field of the parameter that you want to modify.
3. If it is a text field, type a new value in the *Value* field. If a menu choice is offered, select the desired option. The parameters are described in Table 1-3.

**Table 1-3 Component Parameters**

Parameter Name	Description	Allowed Values	Default Value	Runtime <sup>1</sup>
ace_lite_<n> Enable Debug Messages <sup>2, 3</sup>	When set to <i>true</i> writes ACE-Lite debug messages onto the SoC Designer output window.	true, false	false	Yes
ace_lite_<n> axi_size[0-5] <sup>2, 3</sup>	These parameters are obsolete and should be left at the default values. <sup>4</sup>	n/a	size0 default is 0x100000000, size1-5 default is 0x0	No
ace_lite_<n> axi_start[0-5] <sup>2, 3</sup>			0x00000000	No
AXI_s_<n> Enable Debug Messages <sup>2, 5</sup>	When set to <i>true</i> writes AXI3 debug messages onto the SoC Designer output window.	true, false	false	Yes
AXI_s_<n> axi_size[0-5] <sup>2, 5</sup>	These parameters are obsolete and should be left at the default values. <sup>3</sup>	n/a	size0 default is 0x100000000, size1-5 default is 0x0	No
AXI_s_<n> axi_start[0-5] <sup>2, 5</sup>			0x00000000	No
Align Waveforms	When set to <i>true</i> , waveforms dumped from the component are aligned with the SoC Designer simulation time. The reset sequence, however, is not included in the dumped data.  When set to <i>false</i> , the reset sequence is dumped to the waveform data, however, the component time is not aligned with the SoC Designer time.	true, false	true	No
apb Base Address	APB Region base address. The address must be on a 4KB boundary.	0x0 - 0xFFFFFFFF	0x0	No
apb Enable Debug Messages	When set to <i>true</i> writes APB debug messages onto the SoC Designer output window.	true, false	false	Yes



**Table 1-3 Component Parameters (continued)**

Parameter Name	Description	Allowed Values	Default Value	Runtime <sup>1</sup>
apb Size	APB region size.	0x0 - 0xFFFFFFFF	0x100000000	No
Carbon DB Path	Sets the directory path to the database file.	Not used	empty	No
Dump Waveforms	Whether SoC Designer dumps waveforms for this component.	true, false	false	Yes
Enable Debug Messages	When set to <i>true</i> writes the debug messages onto the SoC Designer output window.	true, false	false	Yes
<mem>_Address_For_mat <sup>6</sup>	RBC specifies row, bank, column address. BRC specifies bank, row, column address.	RBC, BRC	RBC	No
<mem>_Bank_Bits	Number of bank bits in memory	value <sup>7</sup>	0	No
<mem>_Column_Bits	Number of column bits in memory	value <sup>3</sup>	0	No
<mem>_Row_Bits	Number of row bits in memory	value <sup>3</sup>	0	No
pclken	Clock enable for APB domain.	0, 1	0x1	Yes
Waveform File	Name of the waveform file. When enabled, SoC Designer writes accumulated waveforms to the waveform file in the following situations: when the waveform buffer fills, when validation is paused and when validation finishes, and at the end of each validation run.	string	arm_cm_DMC400_<component_name>.vcd	No
Waveform Timescale	Sets the timescale to be used in the waveform.	Many values in drop-down	1 ns	No

1. *Yes* means the parameter can be dynamically changed during simulation, *No* means it can be changed only when building the system, *Reset* means it can be changed during simulation, but its new value will be taken into account *only* at the next reset.
2. <n> = 0 to 3.
3. Only for ACELite configurations.
4. ARM recommends using the Memory Map Editor (MME) in SoC Designer, which provides centralized viewing and management of the memory regions available to the components in a system. For information about migrating existing systems to use the MME, refer to Chapter 9 of the *SoC Designer User Guide*.
5. Only for AXI3 configurations.
6. <mem> is ddr3.
7. Sum of <mem>\_Bank\_Bits, <mem>\_Column\_Bits, and <mem>\_Row\_Bits must be less than the “maximum address bits” value set in AMBA Designer.

## 1.5 Debug Features

The DMC-400 Cycle Model has a debug interface (CADI) that allows the user to view, manipulate and control the registers and memory in the SoC Designer Simulator or any debugger that supports the CADI; for example, Model Debugger. You can access a view in SoC Designer Simulator, or an instance of the Model Debugger can be attached by right-clicking on the Cycle Model and choosing the appropriate menu entry.

- [Register Information](#)
- [Memory Information](#)

### 1.5.1 Register Information

The registers are described briefly in this section. See the *DMC-400 TRM* for complete information. In the SoC Designer Simulator Registers view, you can access sub-fields by clicking on the plus sign to the left of a register name. Registers are grouped into different sets according to functional area.

The following Register tabs are supported:

- [DMC Configuration Registers](#)
- [Memory Registers](#)

#### 1.5.1.1 DMC Configuration Registers

Table 1-4 shows the DMC Configuration registers, which appear on the Implementation tab. This tab shows the registers that configure the entire memory controller, including timing and ID registers.

**Table 1-4 DMC Configuration Registers**

Name	Description	Type
memc_status	Memory controller status	Read-Only
memc_config	Memory controller configuration	Read-Only
address_control	Memory address control	Read-Write
decode_control	Address decode control	Read-Write
format_control	Format control	Read-Write
low_power_control	DMC-400 low-power control	Read-Write
turnaround_priority	Read write arbitration control	Read-Write
hit_priority	DRAM traffic control	Read-Write
qos<n>_control	Qos <n> control where $n = 0$ to 15	Read-Write
timeout_control	Timeout control	Read-Write
queue_control	System interface queue control	Read-Write
write_priority_control	Write buffer priority control	Read-Write
write_priority_control2	Write buffer priority control	Read-Write
read_priority_control	Read buffer priority control	Read-Write
read_priority_control2	Read buffer priority control	Read-Write

**Table 1-4 DMC Configuration Registers (continued)**

Name	Description	Type
access_address_match	Access address match	Read-Write
access_address_mask	Access address mask	Read-Write
channel_status	Memory channel status	Read-Only
mr_data	Direct memory command	Read-Only
refresh_control	Memory refresh control	Read-Write
mode_control	Mode control	Read-Write
t_refi_reg	Refresh interval timing	Read-Write
t_rfc_reg	Post auto-refresh command timing	Read-Write
t_mrr_reg	Effective MRR burst length	Read-Write
t_mrww_reg	Post mode register write command delay timing	Read-Write
t_rcd_reg	Post activate command timing	Read-Write
t_ras_reg	Ras timing	Read-Write
t_rp_reg	Post precharge command timing	Read-Write
t_rpall_reg	Post precharge all command timing	Read-Write
t_rrd_reg	Activate to activate command timing	Read-Write
t_faw_reg	Four bank activate window control	Read-Write
read_latency_reg	Read latency timing	Read-Write
t_rtr_reg	Read to read timing	Read-Write
t_rtw_reg	Read to write timing	Read-Write
t_rtp_reg	Read to precharge timing	Read-Write
write_latency_reg	Write latency timing	Read-Write
t_wr_reg	Write recovery timing	Read-Write
t_wtr_reg	Write to read timing	Read-Write
t_wtw_reg	Write to write timing	Read-Write
t_eckd_reg	Enter DRAM clock disable timing	Read-Write
t_xckd_reg	Exit DRAM clock disable timing	Read-Write
t_ep_reg	Enter power down timing	Read-Write
t_xp_reg	Exit power down timing	Read-Write
t_esr_reg	Enter self-refresh timing	Read-Write
t_xsr_reg	Exit self-refresh timing	Read-Write
t_srckd_reg	Self-refresh to DRAM clock disable timing	Read-Write

**Table 1-4 DMC Configuration Registers (continued)**

Name	Description	Type
t_cksrd_reg	DRAM clock enable to exit self-refresh timing	Read-Write
t_rddata_en_reg	DFI read timing	Read-Write
t_phywrlat_reg	DFI write timing	Read-Write
rdlvl_control	Read training control	Read-Write

### 1.5.1.2 Memory Registers

The registers in Table 1-5 are available for each memory interface. In this table:

- $a = 0$  or  $1$
- $b = 0, 1, 2,$  or  $3$
- $c = 0$  to  $\langle \text{number of banks} \rangle$

**Table 1-5 Memory Registers**

Name	Description	Type
CS< $a$ > MR< $b$ >	Mode register $b$ for Chip Select $a$ .	Read-Write
CS< $a$ > ACTIVATE	Activate command for Chip Select $a$ .	Read-Write
CS< $a$ > READ	Read command for Chip Select $a$ .	Read-Write
CS< $a$ > WRITE	Write command for Chip Select $a$ .	Read-Write
CS< $a$ > PRECHARGE	Precharge command for Chip Select $a$ .	Read-Write
CS< $a$ > REFRESH	Refresh command for Chip Select $a$ .	Read-Write
CS< $a$ > READ DATA	Read Data command for Chip Select $a$ .	Read-Write
CS< $a$ > WRITE DATA	Write Data command for Chip Select $a$ .	Read-Write
CS< $a$ > Current RD Utilization	Shows current Read utilization for Chip Select $a$ .	Read-Only
CS< $a$ > Current WR Utilization	Shows current Write utilization for Chip Select $a$ .	Read-Only
CS< $a$ > Current Bus Utilization	Shows current Bus utilization for Chip Select $a$ .	Read-Only
CS< $a$ > ACTIVATE Bank < $c$ >	Activate command for Chip Select $a$ , Bank $c$ .	Read-Write
CS< $a$ > READ Bank < $c$ >	Read command for Chip Select $a$ , Bank $c$ .	Read-Write
CS< $a$ > WRITE Bank < $c$ >	Write command for Chip Select $a$ , Bank $c$ .	Read-Write
CS< $a$ > ACTIVATE Event	Number of ACTIVATE events for Chip Select $a$ .	Read-Only
CS< $a$ > READ Event	Number of READ events for Chip Select $a$ .	Read-Only
CS< $a$ > WRITE Event	Number of WRITE events for Chip Select $a$ .	Read-Only
CS< $a$ > BST Event	Number of BURST events for Chip Select $a$ .	Read-Only

**Table 1-5 Memory Registers (continued)**

Name	Description	Type
CS< <i>a</i> > PRECHARGE Event	Number of PRECHARGE events for Chip Select <i>a</i> .	Read-Only
CS< <i>a</i> > REFRESH Event	Number of REFRESH events for Chip Select <i>a</i> .	Read-Only
CS< <i>a</i> > READ DATA Event	Number of READ DATA events for Chip Select <i>a</i> .	Read-Only
CS< <i>a</i> > WRITE DATA Event	Number of WRITE DATA events for Chip Select <i>a</i> .	Read-Only

In addition to these registers, the Mode registers for different types of memories are also visible.

## 1.5.2 Memory Information

The SoC Designer Memory view does not display any values when the DMC-400 Cycle Model is in the CONFIG state.

## 1.6 Available Profiling Data

Profiling data is enabled, and can be viewed using the Profiling Manager, which is accessible via the Debug menu in the SoC Designer Simulator. The profiling events are the ones that can be monitored in the hardware using counters. The DMC-400 Cycle Model profiles the events shown in Table 1-6.

**Table 1-6 DMC-400 Profiling Streams and Events**

Stream	Events	X axis	Y axis
Memory Command	Read	Cycle	Memory Command
	Write		
	Precharge		
	Precharge All		
	Self Recharge		
	Self Recharge Exit		
	Auto Refresh		
	Mode Reg		
Write Latency	Green	Cycle	Write Latency
	Amber		
	Red		
Read Latency	Green	Cycle	Read Latency
	Amber		
	Red		

The Write Latency is measured from the time of the write request on the AW channel to the time of the response on the B channel. The Read Latency is measured in the same way, using the AR and R channels. The latency values are assigned to buckets based on thresholds you can set using component parameters.