

# PrimeCell® Static Memory Controller (PL350 series) Cycle Model

**Version 9.1.0**

## **User Guide**

**Non-Confidential**



# PrimeCell Static Memory Controller (PL350 series) Cycle Model

## 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

## **Chapter 1.** **Using the Cycle Model in SoC Designer**

PL350 Series Static Memory Controller Cycle Model Functionality .....	2
Fully Functional and Approximate Features .....	2
Unsupported Hardware Features .....	3
Features Additional to the Hardware .....	3
Supported Memory Types .....	4
Adding and Configuring the SoC Designer Component .....	5
SoC Designer Component Files .....	5
Adding the Cycle Model to the Component Library .....	6
Adding the Component to the SoC Designer Canvas .....	6
Available Component ESL Ports .....	7
Setting Component Parameters .....	8
Debug Features .....	10
Register Information .....	10
Memory Information .....	13
Available Profiling Data .....	13



# 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:

- [PL350 Series Static Memory Controller Cycle Model Functionality](#)
- [Supported Memory Types](#)
- [Adding and Configuring the SoC Designer Component](#)
- [Available Component ESL Ports](#)
- [Setting Component Parameters](#)
- [Debug Features](#)
- [Available Profiling Data](#)

## 1.1 PL350 Series Static Memory Controller Cycle Model Functionality

The PL350 Series Cycle Model is fully parameterized so that it supports all configurations in a single model. For a detailed description of the AXI protocol refer to the *AMBA AXI Protocol Specification*.

The SoC Designer Cycle Model of the PL350 SMC provides visibility of the memory mapped registers via the CADI register views. Transaction monitor probes can be hooked to any AXI connection visualizing the transactions. Additionally the SoC Designer profiling interface is used to collect transaction event information.

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.

- [Fully Functional and Approximate Features](#)
- [Unsupported Hardware Features](#)
- [Features Additional to the Hardware](#)

### 1.1.1 Fully Functional and Approximate Features

The following features of the PL350 hardware are implemented in the PL350 Cycle Model, but the exact behavior of the hardware implementation is not accurately reproduced because some approximations and optimizations have been made for simulation performance:

- SRAM memory data widths of 8-bit, 16-bit, or 32-bit
- NAND memory data widths of 8-bit or 16-bit
- AXI data width of 32-bit or 64-bit
- Up to four chip selects per memory interface
- Configurable command, read data, and write data FIFO depths
- An additional pipeline stage within the format logic enables higher AXI clock frequencies at the cost of an additional clock cycle of latency
- Configurable number of outstanding exclusive accesses
- APB interface is modeled as APB transaction interface. For more information on this, refer to the *RVModelLib\_MxAPB.pdf*, which is provided in the AMBA2 package of the SoC Designer Cycle Model Library. In particular, this APB interface does not contain *reset* and *clock* ports. Instead of this, *reset* and *clock* ports are provided separately.

## 1.1.2 Unsupported Hardware Features

The following features of the PL350 hardware are not implemented in the PL350 Cycle Model:

- External Bus Interface (EBI) is not supported
- AXI low-power interface
- Asynchronous clocking mode is not supported
- Asynchronous aclk and mclk and clock ratio between the two clocks
- Multiplexed mode is not supported for SRAM memory
- ECC interfaces and registers are not supported
- ATPG signals
- DFT
- Integration test registers:
  - int\_outputs
  - int\_inputs
  - int\_cfg

## 1.1.3 Features Additional to the Hardware

The following features that are implemented in the PL350 SMC Cycle Model to enhance usability do not exist in the PL350 hardware:

- The PL35x memory controller 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 [“Debug Features”](#) on page 1-10 and [“Available Profiling Data”](#) on page 1-13 respectively.

## 1.1.4 Supported Memory Types

The following SRAM and NAND memories are currently supported.

### 1.1.4.1 SRAM Cycle Model

The SRAM Cycle Model is based on the Micron Cellular RAM (MT45W4MW16BFB), which supports asynchronous, page, and burst (synchronous) operation. For configuring SRAM, refer to the Bus Configuration Register in the datasheet. The Burst Length, Latency Counter, Operating Mode, and Register Select fields of this register should be configured correctly. Other fields should be left at their default values as they are not supported.

It can be configured for 8-bit, 16-bit, or 32-bit.

Memory Size:

8-bit: 0x100000 (1M)

16-bit: 0x200000 (2M)

32-bit: 0x400000 (4M)

Note that the SRAM Cycle Model does not support multiplexed mode.

### 1.1.4.2 NAND Cycle Model

The NAND Cycle Model is based on the Toshiba NAND (TC58DVM82F1FT00), which supports the following modes:

- Serial Data Input
- Read Mode (1)
- Read Mode (2)
- Read Mode (3)
- Auto Program
- Status Read

Note that the following modes are not supported:

- Reset
- Auto block erase
- ID Read

It can be configured for 8-bit or 16-bit.

Memory Size (8-bit or 16-bit):

Main Memory: 0x2000000 (32M)

Spare Memory: 0x100000 (1M)

## 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.

## 1.3 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*.

### 1.3.1 Adding the Component to the SoC Designer Canvas

Locate the component in the *Component Window* and drag it out to the Canvas.



## 1.4 Available Component ESL Ports

The PL350 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. See the *ARM PrimeCell® Static Memory Controller (PL350) Technical Reference Manual* for more information.

**Table 1-2 ESL Component Ports**

ESL Port	Description	Direction	Type
address_mask<x> <sup>1</sup> _ <sub>&lt;n&gt;<sup>2</sup></sub>	Address mask tie-off for chip <n> for interface <x>. This mask is applied to the AXI address bits before the comparison with the <i>address_matchx_n</i> value.	Input	Signal slave
address_match<x> <sup>1</sup> _ <sub>&lt;n&gt;<sup>2</sup></sub>	Address match tie-off for chip <n> for interface <x>. This is the comparison value that determines the chip select base address. For example, <i>address_match0</i> and <i>address_mask0</i> set up the address mapping for memory interface 0.	Input	Signal slave
apb	APB port for memory mapped register accesses. Refer to section A.4 of the <i>PL350 Technical Reference Manual</i> for the APB signal list.	Input	APB transaction slave
axi	AXI port for memory accesses. The port implementation is compliant with the AXI v2 protocol. Only one AXI master port is allowed to connect to this port. Refer to section A.3 of the <i>PL350 Technical Reference Manual</i> for the AXI signal list.	Input	AXI transaction slave
nand_booten_<x> <sup>1</sup>	This signal enables nand booting functionality. Available <i>only</i> in configurations with NAND interfaces (PL351/PL353).	Input	Signal slave
pclken	Clock enable for APB domain.	Input	Signal slave
remap_<x> <sup>1</sup>	This signal is used to remap the specified chip to address 0x0.	Input	Signal slave
user_status	General purpose APB-accessible input port.	Input	Signal slave
clk-in	Clock slave port.	Input	Clock slave
smc_int	Combined interrupt output.	Output	Signal master
smc_int<x> <sup>1</sup>	Individual memory interrupt output. Available <i>only</i> in configurations with multiple memory interfaces (PL353/PL354).	Output	Signal master
user_config	General purpose APB-accessible output port.	Output	Signal master

1. Where  $\langle x \rangle$  represents memory interface 0 or 1.
2. Where  $\langle n \rangle$  indicates chip select 0 to 3.

All pins that are not listed in this table have been either tied or disconnected for performance reasons.

*Note: Some ESL component port values can be set using a component parameter. This includes the ports `nand_booten_<x>`, `address_mask<x>_<n>`, `address_match<x>_<n>`, and `user_status`. In those cases, the parameter value is used whenever the ESL port is not connected. If the port is connected, the connection value takes precedence over the parameter value.*

## 1.5 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 component and select **Edit Parameters...**. 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>
<code>address_mask&lt;x&gt;<sup>2</sup>_&lt;n&gt;<sup>3</sup></code>	Address mask for chip $\langle n \rangle$ for interface $\langle x \rangle$ .	0x0 - 0xFF	0x00	No
<code>address_match&lt;x&gt;<sup>2</sup>_&lt;n&gt;<sup>3</sup></code>	Address match for chip $\langle n \rangle$ for interface $\langle x \rangle$ .	0x0 - 0xFF	0x00	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
axi axi_size[0-5]	These parameters are obsolete and should be left at the default values. <sup>4</sup>	n/a	size0 default is 0x100000000,	No
axi axi_start[0-5]			size1-5 default is 0x0 0x00000000	No
axi Enable Debug Messages	When set to <i>true</i> writes AXI debug messages onto the SoC Designer output window.	true, false	false	Yes
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
Maximum amber read latency	Max value displayed as amber for read latency profiling.	>0	30	Yes
Maximum amber write latency	Max value displayed as amber for write latency profiling.	>0	30	Yes
Maximum green read latency	Max value displayed as green for read latency profiling.	>0	20	Yes
Maximum green write latency	Max value displayed as green for write latency profiling.	>0	20	Yes
nand_booten_<x> <sup>2</sup>	Enable NAND booting functionality for the memory interface.	0, 1	0x0	No
pclken	clock enable for APB domain.	0, 1	0x1	Yes
user_status	General purpose APB-accessible pin.	0x0 - 0xFF	0x0	Yes
Waveform File <sup>5</sup>	Name of the waveform file.	<i>string</i>	arm_cm_pl35x_<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 is taken into account *only* at the next reset.
2. Where <x> represents memory interface 0 or 1.
3. Where <n> indicates chip select 0 to 3.

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. 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.

## 1.6 Debug Features

The PL350 AXI Memory Controller 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. A view can be accessed in the 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. The views shown in this section are for the SoC Designer Simulator.

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

### 1.6.1 Register Information

Registers are grouped into different sets according to functional area. For each enabled port of the PL350 AXI Memory Controller Cycle Model, there is tab in the SoC Designer Simulator.

The set of available registers is similar among the different controller configuration (PL351, PL352, etc.). The registers are described briefly in this section. See the *ARM PrimeCell Static Memory Controller (PL350) Technical Reference Manual* for complete information.

The following Register tabs are supported:

- [Memory Configuration Registers](#)
- [Chip Configuration Registers](#)
- [User Configuration Registers](#)
- [PrimeCell Configuration Registers](#)

Each register has a pop-up tool tip attached to it, providing further details on the purpose and behavior of the register. Some register values are displayed in a symbolic format by default. The display format can be changed to hexadecimal or decimal from the Register view's context menu.

The AXI pin interface is displayed as a set of pseudo registers that are traceable. The integration test registers are not modeled. Because more than one register view can be open simultaneously it is possible to view several ports at the same time.

Breakpoints can be set on all registers. A breakpoint only triggers for value changes. Register values can also be traced to be displayed in the waveform viewer.

### 1.6.1.1 Memory Configuration Registers

Table 1-4 shows the Memory Configuration registers. Use these registers for the global configuration, and control of the operating state, of the SMC.

**Table 1-4 Memory Configuration Registers**

Name	Description	Type
memc_status	The Memory Controller Status register provides information on the configuration of the memory controller, and also on the current state of the memory controller.	read-only
memif_cfg	The Memory Interface Configuration register provides information on the configuration of the memory interface.	read-only
memc_cfg_set	The Set Configuration register enables the SMC to be changed to low-power state, and interrupts enabled.	write-only
direct_cmd	The Direct Command register passes commands to the external memory, and controls the updating of the chip configuration registers.	write-only
set_cycles	The Set Cycles register contains values that are written to the SRAM or NAND registers when the SMC receives a write request.	write-only
set_opmode	The Set Opmode register is the holding register for the <i>opmode</i> registers (see “ <a href="#">Chip Configuration Registers</a> ” on page 1-11).	write-only
refresh_period_0	The Refresh Period 0 register enables the SMC to insert idle cycles during consecutive bursts, which enables the PSRAM devices on memory interface 0 to initiate a refresh cycle.	read-write
refresh_period_1	The Refresh Period 1 register enables the SMC to insert idle cycles during consecutive bursts, which enables the PSRAM devices on memory interface 1 to initiate a refresh cycle.	read-write

### 1.6.1.2 Chip Configuration Registers

Table 1-5 shows the CHIP\_CFG Chip Configuration registers. These registers hold the operating parameters of each chip select. If the SMC is not configured to support all chip selects, the corresponding registers are not implemented.

**Table 1-5 Chip Configuration Registers**

Name	Description	Type
sram_cycles<x>_<n>	SRAM Cycles register. There is an instance of this register for each SRAM chip supported.	read-only
nand_cycles<x>_<n>	NAND Cycles register. There is an instance of this register for each NAND chip supported.	read-only
opmode<x>_<n>	opmode register. There is an instance of the opmode register for each chip supported.	read-only

Where <x> represents memory interface 0 or 1, and <n> indicates chip select 0 to 3.

### 1.6.1.3 User Configuration Registers

Table 1-6 shows the USER\_CFG User Configuration registers. These registers provide general purpose I/O for user-specific applications.

**Table 1-6 User Configuration Registers**

Name	Description	Type
user_status	This register returns the state of the user_status[7:0] primary inputs.	read-only
user_config	This register sets the value of the user_config[7:0] primary outputs.	read-write

### 1.6.1.4 PrimeCell Configuration Registers

Table 1-7 shows the PrimeCell Identification registers. These registers enable the identification of system components by software.

**Table 1-7 PrimeCell ID Registers Summary**

Register	Description	Type
periph_id_0	Peripheral Identification 0 register. Identifies the part number of the peripheral.	read-only
periph_id_1	Peripheral Identification 1 register. Identifies the part number and designer of the peripheral.	read-only
periph_id_2	Peripheral Identification 2 register. Identifies the revision and designer of the peripheral.	read-only
periph_id_3	Peripheral Identification 3 register. Identifies the configuration of the peripheral.	read-only
pcell_id_0	PrimeCell Identification 0 register. Determines the reset value.	read-only
pcell_id_1	PrimeCell Identification 1 register. Determines the reset value.	read-only
pcell_id_2	PrimeCell Identification 2 register. Determines the reset value.	read-only
pcell_id_3	PrimeCell Identification 3 register. Determines the reset value.	read-only

## 1.6.2 Memory Information

Component parameters are used for debug access before the start of simulation, so program loading uses the **address\_mask**<x>\_<n> and **address\_match**<x>\_<n> parameters values. Once the simulation starts (cycle count > 0), debug access uses "opmode" register values.

## 1.7 Available Profiling Data

The PL35x component supports CAPI profiling. 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 PL35x Cycle Model profiles the events shown in Table 1-8.

**Table 1-8 PL350 Profiling Streams and Events**

Stream	Events	X axis	Y axis
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.

