Level 2 Cache Controller (PL310) Cycle Model

Version 9.1.0

User Guide

Non-Confidential

ARM®

ARM DUI 1072A (ID120516)

Level 2 Cache Controller (PL310) Cycle Model
User Guide

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

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

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
`courier`	Commands, functions, variables, routines, and code examples that are set apart from ordinary text.	`sparseMem_t SparseMemCreate-New();`
italic	New or unusual words or phrases appearing for the first time.	*Transactors* provide the entry and exit points for data ...
bold	Action that the user performs.	Click **Close** 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	*Advanced Microcontroller Bus Architecture*. 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	*Advanced High-performance Bus*. 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	*Advanced Peripheral Bus*. 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	*Advanced eXtensible Interface*. 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 *Cycle Model Compiler*) 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	*ESL API Simulation Interface*, is based on the SystemC communication library and manages the interconnection of components and communication between components.
CADI	*ESL API Debug Interface*, enables reading and writing memory and register values and also provides the interface to external debuggers.
CAPI	*ESL API Profiling Interface*, 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	*Electronic System Level*. 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	*Hardware Description Language*. A language for formal description of electronic circuits, for example, Verilog.
RTL	*Register Transfer Level*. 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	*Transaction adaptors*. 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 PL310 Cycle Model and how to use it in SoC Designer. It contains the following sections:

1.1 PL310 Cache Controller Cycle Model Functionality

The PL310 L2CC is a Level 2 Cache Controller. The addition of an on-chip secondary cache, also referred to as a Level 2 or L2 cache, is a recognized method of improving the performance of ARM-based systems when significant memory traffic is generated by the processor. By definition, a secondary cache assumes the presence of a Level 1 or primary cache, closely coupled, or internal to the processor.

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 simulates, refer to the *ARM PrimeCell Level 2 Cache Controller (PL310) Technical Reference Manual.*

1.1.1 Implemented Hardware Features

The following features of the PL310 hardware are implemented in the PL310 Cycle Model:

- Slave and master ports
- Line fill, line read, eviction, and store buffers
- Data and tag RAMs
- All types of AXI transactions
- Write back / write through behaviors
- Out-of-order responses
- Address decoding for register accesses and all user-visible registers
- Bank locking, line locking
- Pseudo-Random victim selection policy
- Event pins and registers
- Interrupts
- Maintenance operations
- Force allocate, shared attribute, exclusive configuration
- Locked and exclusive accesses
- External errors
- Hazard detection and correction
- TrustZone support
- Address filtering
- User-defined Data and Tag RAM clock enable signals

1.1.2 Unsupported Hardware Features

The following features of the PL310 hardware are not implemented in the PL310 Cycle Model because they are not relevant from a modeling point of view:

- Parity errors
- Clock management and power modes
- Test mode and scan chains
- Debug mode and debug registers. The debug features are not implemented, but the Cycle Model provides more powerful debug capabilities.
- MBIST support
- The following registers are not available to be read / written via debug transactions — for example, in the SoC Designer Registers window, or by accessing them directly from RealView Debugger:
  - Maintenance registers: Invalidate Line By PA, Invalidate by Way, Clean Line by PA, Clean Line by Index/Way, Clean by Way, Clean and Invalidate Line by PA, Clean and Invalidate Line by Index/Way, and Clean and Invalidate by Way

  The functionality of these registers, however, does exist and can be accessed by software running on the virtual platform.

1.1.3 Features Additional to the Hardware

The following features are implemented in the PL310 Cycle Model to enhance usability. They do not exist in the PL310 hardware:

- Extended profiling features. The Cycle Model provides additional profiling metrics.
- Extended debug features. The Cycle Model provides additional debug capabilities.
- The data and tag RAMs are embedded into the Cycle Model.

1.1.4 SoC Designer Interfaces

The PL310 Cycle Model interfaces with the SoC Designer tools using:

- CASI for simulation control, component connections, parameters, save and restore
- CAPI for profiling
- CADI for register view, memory view, and debug operations

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<*component_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<*component_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 Cycle Model is located and select the component configuration file:
   - `maxlib.lib<component_name>.conf` (for Linux)
   - `maxlib.lib<component_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

Locate the component in the *Component Window* and drag it out to the Canvas. Note that some ports are not modeled because the associated function is not implemented: power management, test and debug modes, MBIST and synchronous interfacing modes.

1.3 Available Component ESL Ports

The PL310 Cycle Model has AXI slave and master ports, and pins. They all are implemented using standard transaction classes. Table 1-2 describes the ESL ports that are exposed in SoC Designer.

Table 1-2 ESL Component Ports

ESL Port	Description	Direction	Type
clk-in	Input clock	Slave	clock
axi_s0	AXI slave port 0	Slave	AXI
axi_s1	AXI slave port 1	Slave	AXI
ASSOCIATIVITY [1.]	Cache associativity	Input	int
CACHEID [1]	Component identification	Input	int
DATACLKEN[2]	Clock enable for data RAM interface	Input	bool
DATARAMCLK[2.]	Clock for data RAM, see also DATACLKEN	Input	clock
INCLKENM0/ OUTCLKENM0	Clock enable for input/output clocks for port axi_m0	Input	int

Table 1-2 ESL Component Ports (continued)

ESL Port	Description	Direction	Type
INCLKENM1/ OUTCLKENM1	Clock enable for input/output clocks for port axi_m1	Input	int
INCLKENS0/ OUTCLKENS0	Clock enable for input/output clocks for port axi_s0	Input	int
INCLKENS1/ OUTCLKENS1	Clock enable for input/output clocks for port axi_s1	Input	int
REGFILEBASE	Register file base address	Input	int
SPNIDEN	Bypass security for output events	Input	bool
TAGCLKEN[2.]	Clock enable for tag RAM interface	Input	bool
TAGRAMCLK[2.]	Clock for tag RAM, see also TAGCLKEN	Input	clock
WAYSIZE [1.]	Cache way size	Input	int
axi_m0	AXI master port 0	Master	AXI
axi_m1	AXI master port 1	Master	AXI
IDLE	Notify PL310 is idle	Output	bool
DATAWRITELAT	Write data RAM latency	Output	int
DATAREADLAT	Read data RAM latency	Output	int
CO	Cast out (line eviction) event	Output	bool
DRHIT	Data read hit event	Output	bool
DRREQ	Data read miss event	Output	bool
DWHIT	Data write hit event	Output	bool
DWREQ	Data write-back miss event	Output	bool
DWTREQ	Data write-through miss event	Output	bool
IRHIT	Instruction read hit event	Output	bool
IRREQ	Instruction read miss event	Output	bool
WA	Write allocation	Output	bool
DECERRINTR [3]	Imprecise DEC error interrupt	Output	bool
SLVERRINTR [3.]	Imprecise SLV error interrupt	Output	bool
ECNTRINTR [3.]	Configurable event interrupt	Output	bool
L2CCINTR [3.]	Global interrupt	Output	bool

1. These input values are only taken into account after reset.
2. Available only if user defined RAM clock enable option was selected.See the section on “RAM clocking” in the *ARM PrimeCell Level 2 Cache Controller (PL310 Technical Reference Manual* for more details on how to drive these inputs.
3. For these interrupt ports, the active high/low setting is controlled by the `negLogic` component parameter. The default is active high.

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 REGFILEBASE, CACHEID, SPNIDEN, CFGBIGEND, ASSOCIATIVITY, and WAYSIZE ports. In those cases, the parameter value will be used whenever the ESL port is not connected. If the port is connected, the connection value takes precedence over the parameter value.

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 component's 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

Name	Description	Allowed Values	Default Value	Runtime[1]
Align Waveforms	When set to *true*, 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 *false*, 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
ASSOCIATIVITY	Defines the value used for the ASSOCIATIVITY input if it is left unconnected.	0,1 [2]	0	Reset
axi_m0 Enable Debug Messages	Whether debug messages are logged for master port 0.	true, false	false	Yes
axi_m1 Enable Debug Messages	Whether debug messages are logged for master port 1.	true, false	false	Yes
axi_s[0-1] axi_size[0-5] [3]	These parameters are obsolete and should be left at their default values.[4]	0x0 – 0x100000000	size0 default is 0x100000000, size1-5 default is 0	No
axi_s[0-1] axi_start[0-5]		0x00000000 – 0xffffffff	0x00000000	No

Table 1-3 Component Parameters (continued)

axi_s0 Enable Debug Messages	Whether debug messages are logged for slave port 0.	true, false	false	Yes
axi_s1 Enable Debug Messages	Whether debug messages are logged for slave port 1.	true, false	false	Yes
CACHEID	Sets initial value of CACHEID register, the cache controller cache ID.	any	1234	Reset
Carbon DB Path	Sets the directory path to the database file.	Not Used	empty	No
CFGBIGEND	Sets initial value of CFGBIGEND big endian mode register.	true, false	false	Yes
CFGADDRFILTEN	Enables address filtering if it is available in the component configuration.	0,1	0 (disabled)	Yes
CFGADDRFILTEND_31_20	Sets the initial value for the address filtering end register.	0x0 -- 0xFFF[5]	0x0	Yes
CFGADDRFILTSTART_31_20	Sets the initial value for the address filtering start register.	0x0 -- 0xFFF[5].	0x0	Yes
Dump Waveforms	Whether SoC Designer dumps waveforms for this component.	true, false	false	Yes
Enable Debug Messages	Whether debug messages are logged for the component.	true, false	false	Yes
INCLKENM0 INCLKENM1 INCLKENS0 INCLKENS1 OUTCLKENM0 ...	Defines the value used for the like-named port if that port is left unconnected. For each port there are two clock enable signals, e.g. INCLKENS0 and OUTCLKENS0	0,1	1 (enabled)	Yes
negLogic	Sets interrupt assertion to use negative logic. Default of *false* means 0=off and 1=on. *True* means 0=on and 1=off.	true, false	false	No
REGFILEBASE [6]	Defines the value used for the REGFILEBASE input if it is left unconnected. The base address for accessing the configuration registers.	any	0	Yes
SPNIDEN	Defines the value used for the SPNIDEN input if it is left unconnected. This signal enables you to bypass security for output events. This must be set to *true* to enable profiling (see “Available Profiling Data” on page 1-14).	true, false	false	Yes

Table 1-3 Component Parameters (continued)

WAYSIZE	Defines the value used for the WAYSIZE input if it is left unconnected.	1,2,3,4,5,6 [7]	1	Reset
Waveform File [8]	Name of the waveform file.	*string*	arm_cm_pl310.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. Effective associativity is respectively 8 (0) and 16 (1) ways.
3. The square brackets used in parameter names specify a range of numbers that are available for the Cycle Model. The parameter name for the start addresses “axi_s[0-1] axi_start[0-5]” for example will be expanded to 12 possible parameter name combinations that range from “axi_s0 axi_start0” to “axi_s1 axi_start5”. The size of a memory region depends on the “axi_s[N] axi_start[M]” and “axi_s[N] axi_size[M]” parameters. The end address is calculated as StartAddr +Size -1. The size of the memory region must not exceed the value of 0x100000000. If the sum of StartAddr+Size is greater than 0x100000000, the size of the memory region is reduced to the difference: 0x100000000-StartAddr.
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. The CFGADDRFILTSTART_31_20 and CFGADDRFILTEND_31_20 parameters, as well as the associated internal registers, hold the upper 12 bits of the address filtering values, not the full 32 bits.
6. The REGFILEBASE parameter, as well as the REGFILEBASE input pin, represents the 20 most significant bits of the register file base address, not the full 32-bit address.
7. Effective way size in KB is equal to $16*2^{WAYSIZE-1}$: 16, 32, 64,128, 256 or 512 KB.
8. 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.

1.5 Debug Features

The PL310 Cycle Model has a debug interface (CADI) that allows the user to view, manipulate, and control the registers and memory. A view can be accessed in SoC Designer by right clicking on the Cycle Model and choosing the appropriate menu entry.

The debug features described here include:

- Debug Read and Write Operations
- Register Information
- Memory View

1.5.1 Debug Read and Write Operations

The debug read and write operations are initiated by a master controlling the AXI slave ports of the PL310. This is used when one wants to look at the memory content from a CPU Core connected to the PL310. The PL310 tries to locate the target address in its internal buffers, its data RAM and propagates the debug operation to its master ports.

When a debug read or write address falls into the PL310 register page (set by REGFILEBASE), it is directly forwarded to the master ports. To access registers, refer to the Register Information section.

1.5.2 Register Information

This section lists the Registers available for the PL310 Cycle Model. Registers are grouped according to functional area into different sets. The available groups are:

- Control Registers
- Debug_Prefetch_Power Registers
- Events Registers
- Filtering Registers
- ID Registers
- Interrupt Registers
- Lockdown Registers
- Maintenance Registers

Note that writing in some registers, such as the Maintenance related ones, starts a background process that may take a number of cycles to complete. When these registers are modified through the SoC Designer interface you must still wait a number of cycles for the process to complete. You can check the status of the background process by viewing the register using the SoC Designer interface.

See the *ARM PrimeCell Level 2 Cache Controller (PL310) Technical Reference Manual* for details about these registers.

1.5.2.1 Control Registers

Table 1-4 shows the Control registers.

Table 1-4 Control Registers

Name	Base Offset	Type	Reset Value	Description
Control	0x100	RW	0x00000000	Register 1, Control Register
AuxControl	0x104	RW	0x02000000 [1]	Register 1, Auxiliary Control Register
TagLatencies	0x108	RW	0x00000777	Register 1, Tag and Data RAM Latency Control Registers
DataLatencies	0x10C	RW	0x00000777	

1. This value is pin dependent, depending on how external WAYSIZE and ASSOCIATIVITY pins are tied.

1.5.2.2 Debug_Prefetch_Power Registers

Table 1-5 shows the Debug, Prefetch, Power registers.

Table 1-5 Debug Prefetch Power Registers

Name	Base Offset	Type	Reset Value	Description
DebugControl	0xF40	RW	0x00000000	Register 15, Debug Register
Prefetch_Control	0xF60	RW	0x00000000	Register 15, Prefetch Control Register
Power_Control	0xF80	RW	0x00000000	Register 15, Power Control Register

1.5.2.3 Events Registers

Table 1-6 shows the Event Counter Control registers.

Table 1-6 Events Registers

Name	Base Offset	Type	Reset Value	Description
EvCounterCtrl	0x200	RW	0x00000000	Register 2, Event Counter Control Register
EvCounter1Conf	0x204	RW	0x00000000	Register 2, Event Counter Configuration Registers
EvCounter0Conf	0x208	RW	0x00000000	
EvCounter1Value	0x20C	RW	0x00000000	Register 2, Event Counter Value Registers
EvCounter0Value	0x210	RW	0x00000000	

1.5.2.4 Filtering Registers

Table 1-7 shows the Address Filtering registers.

Table 1-7 Filtering Registers

Name	Base Offset	Type	Reset Value	Description
AddrFilteringStart	0xC00	RW	0x00000000	Register 12, Address Filtering
AddrFilteringEnd	0xC04	RW	0x00000000	

These registers are only available if the L2C-310 was configured with address filtering. These registers are loaded at reset from the parameters with the associated names. You may change the values of these registers at runtime but care must be taken to avoid unpredictable results. For more details see the *ARM PrimeCell Level 2 Cache Controller (PL310) Technical Reference Manual*

1.5.2.5 ID Registers

Table 1-8 shows the Cache ID and Cache Type registers.

Table 1-8 Cache ID and Cache Type Registers

Name	Base Offset	Type	Reset Value	Description
CacheID	0x000	RO	0x410000fd [1]	Register 0, Cache ID Register
CacheType	0x004	RO	0x1e000000 [2]	Register 0, Cache Type Register

1. This value is pin and IP version dependent, and can change if the CACHEID pins are tied.
2. This value is pin dependent, depending on how external WAYSIZE and ASSOCIATIVITY pins are tied.

1.5.2.6 Interrupt Registers

Table 1-9 shows the Interrupt Control registers. These registers can be accessed by secure and non-secure operations.

Table 1-9 Interrupt Registers

Name	Base Offset	Type	Reset Value	Description
InterruptMask	0x214	RW	0x00000000	Register 2, Interrupt Registers
MaskedInterrupt (Status)	0x218	RO	0x00000000	
RawInterrupt (Status)	0x21C	RO	0x00000000	

1.5.2.7 Lockdown Registers

Table 1-10 shows the Cache Lockdown registers.

Table 1-10 Lockdown Registers

Name	Base Offset	Type	Reset Value	Description
DataLock0	0x900	RW	0x00000000	Register 9, Cache Lockdown
InstLock0	0x904	RW	0x00000000	
DataLock1 [1]	0x908	RW	0x00000000	
InstLock1 [1]	0x90C	RW	0x00000000	
DataLock2 [1]	0x910	RW	0x00000000	
InstLock2 [1]	0x914	RW	0x00000000	
DataLock3 [1]	0x918	RW	0x00000000	
InstLock3 [1]	0x91C	RW	0x00000000	
DataLock4 [1]	0x920	RW	0x00000000	
InstLock4 [1]	0x924	RW	0x00000000	
DataLock5 [1]	0x928	RW	0x00000000	
InstLock5 [1]	0x92C	RW	0x00000000	
DataLock6 [1]	0x930	RW	0x00000000	
InstLock6 [1]	0x934	RW	0x00000000	
DataLock7 [1]	0x938	RW	0x00000000	
InstLock7 [1]	0x93C	RW	0x00000000	
LockEnaByLine [2]	0x950	RW	0x00000000	
UnlockAllLines [2]	0x954	RW	0x00000000	

1. These registers are implemented if the option pl310_LOCKDOWN_BY_MASTER is enabled in the Cycle Model configuration file, *pl310_defs.v*. Otherwise, they are unused.
2. These registers are implemented if the option pl310_LOCKDOWN_BY_LINE is enabled. Otherwise, they are unused.

1.5.2.8 Maintenance Registers

Table 1-11 shows the Cache Maintenance Operations registers.

Table 1-11 Maintenance Registers

Name	Base Offset	Type	Reset Value	Description
Sync	0x730	RW	0x00000000	Register 7, Cache Maintenance Operations

1.5.3 Memory View

Both the Data and Tag RAMs can be observed through the Memory view of the PL310 Cycle Model cache memory. Use the “Space” pull-down to select the memory to view. The Data RAM(s) are displayed as lines of 256 bits which is the data RAM organization. Note that these 256 bit “words” are still addressed as bytes.

Refer to the *SoC Designer User Guide* for details about viewing memory.

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.

Note: The PL310 pin SPNIDEN must be set to one (1) to enable profiling information. This is set using the component parameter SPNIDEN (see Table 1-3 *on page 1-7). When set to the default value of 0, the event counters only increment on non-secure events.*

The PL310 Cycle Model profiles the following events:

- Eviction (CastOut) of a line from the L2 cache
- Data read hit
- Data read request
- Data write hit
- Data write request
- Data write-through request
- Instruction read hit
- Instruction read request
- Prefetch linefill sent to L3
- Write allocate

Note that the events are linked to the cycle count and also to addresses. Therefore, the profiling results can be observed as accesses per cycle, or accesses per address, or addresses per cycle. In all cases, the color scheme identifies the types of events.