

Level 2 Cache Controller (L2CC) Cycle Model

Version 9.1.0

User Guide

ARM®

Level 2 Cache Controller (L2CC) 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**

L220 Cache Controller Cycle Model Functionality	2
Fully Functional and Accurate Features	2
Unsupported Hardware Features	3
Features Additional to the Hardware	3
Adding and Configuring the SoC Designer Component	4
SoC Designer Component Files	4
Adding the Cycle Model to the Component Library	5
Adding the Component to the SoC Designer Canvas	5
Available Component ESL Ports	6
Setting Component Parameters	8
Debug Features	10
Debug Read and Write Operations	10
Register Information	10
Memory View	12
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 ...
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	<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 L220 Cycle Model and how to use it in SoC Designer. It contains the following sections:

- [L220 Cache 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 L220 Cache Controller Cycle Model Functionality

The L220 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 Level 2 Cache Controller (L220) Technical Reference Manual*.

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

1.1.1 Fully Functional and Accurate Features

The following features of the L220 Cache Controller hardware are implemented in the L220 Cycle Model:

- L2 Cache controlling functionality for configuration with one or two master ports and one or two slave ports.
- Line Fill buffers (LFBs), Line Read buffers (LRBs), Eviction Buffer (EB), Write Buffer (WB), and Write-Allocate buffer (WAB).
- Supports all of the AXI cache modes: write-through and write-back, through configuration of the ACACHE lines; Read allocate, write allocate, read and write allocate.
- TrustZone support
- Shared transfers treated as non-cacheable with the option to override this behavior, also known as Shared Override.
- Physically addressed and physically tagged
- Lockdown format C (way locking)
- L2 cache size can be from 16KB to 2MB
- Direct mapped to 8-way associativity
- Pseudo-Random victim selection policy

1.1.2 Unsupported Hardware Features

The following features of the L220 hardware are not implemented in the L220 Cycle Model:

- MBIST controller functionality
- Interface for accessing external cache RAM
- Intelligent Energy Manager (IEM) register slices
- Parity and memory error support
- The following registers are not available to be read / written via debug transactions — for example, in the SoC Designer Registers window:
 - r2 - NTRCLR (Interrupt Clear) register
 - r7 - INVALLINEPA (Invalidate Line by PA), INVALBYWAY (Invalidate by Way), CLEANLINEBYP (Clean Line by PA), CLEANLINEBYINDEXWAY (Clean Line by Index/Way), CLEANBYWAY (Clean by Way), CLEANINVALBYP (Clean and Invalidate Line by PA), CLEANINVALBYINDEXWAY (Clean and Invalidate Line by Index/Way), and CLEANINVALBYWAY (Clean and Invalidate by Way) registers

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 L220 Cycle Model to enhance usability. They do not exist in the L220 hardware:

- The Cycle Model supports enabling of debug and profiling information. See [“Debug Features”](#) on page 1-10 and [“Available Profiling Data”](#) on page 1-12 for more information.

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

Locate the component in the *Component Window* and drag it out to the Canvas. Multiple ports of each type appear only when they have been defined in the Cycle Model RTL configuration file (`L220_defs.v`).

1.3 Available Component ESL Ports

The L220 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
CACHEID ¹	Component identification.	Input	Signal slave
SPNIDEN	Secure privileged non-invasive debug enable.	Input	Signal slave
axi_periph	32-bit AXIv2 port for peripheral read/write accesses. This port enables access to the L220 registers. ²	Slave	AXI Transaction slave
axi_s0	64-bit AXI v2 port for interfacing with <i>instruction</i> interfaces of the connected processor. If the port is not used it must be disabled.	Slave	AXI Transaction slave
axi_s1	64-bit AXI v2 port for interfacing with <i>data</i> interfaces of the connected processor. This port must not be disabled.	Slave	AXI Transaction slave
clk-in	Input clock that can be connected to a clock master. If it is unconnected, it is implicitly connected to the system master clock.	Slave	Clock slave
CO	Eviction (CastOUT) of a line or a half line from the L2 cache.	Output	Signal master
DECERRINT	Decode error received on master port from L3.	Output	Signal master
DRHIT	Data read hit in the L2 cache.	Output	Signal master
DRREQ	Data read lookup to the L2 cache. Results in a hit or miss.	Output	Signal master
DWHIT	Data write hit in the L2 cache.	Output	Signal master
DWREQ	Data write lookup to the L2 cache. Results in a hit or miss.	Output	Signal master
DWTREQ	Data write lookup to the L2 cache with Write-Through attribute. Results in a hit or miss.	Output	Signal master
ECNTRINTR	Event Counter Overflow/Increment.	Output	Signal master
ERRRDINTR	Error on L2 data RAM read.	Output	Signal master
ERRRTINTR	Error on L2 tag RAM read.	Output	Signal master
ERRWDINTR	Error on L2 data RAM write.	Output	Signal master
ERRWTINTR	Error on L2 tag RAM write.	Output	Signal master
IRHIT	Instruction read hit in the L2 cache.	Output	Signal master
IRREQ	Instruction read lookup to the L2 cache. Results in a hit or miss.	Output	Signal master

Table 1-2 ESL Component Ports (continued)

ESL Port	Description	Direction	Type
L2CCINTR	Combined Interrupt Output.	Output	Signal master
PARRDINTR	Parity error on L2 data RAM read.	Output	Signal master
PARRTINTR	Parity error on L2 tag RAM read.	Output	Signal master
SLVERRINTR	Slave error received on master port from L3.	Output	Signal master
WA	Allocation into the L2 cache caused by a write (with Write-Allocate attribute) miss.	Output	Signal master
axi_m0	64-bit AXI v2 master port for interfacing with the L3 memory system. If the port is not used it must be disabled. This port must not be enabled if port <i>axi_s0</i> is disabled.	Master	AXI Transaction master
axi_m1	64-bit AXI v2 master port for interfacing with the L3 memory system. This port must not be disabled.	Master	AXI Transaction master

1. These input values are only taken into account after reset.
2. The data portion of the port is 32 bits, but the address is only 12 bits wide. You may connect it to master that uses a 32-bit address. However, only the low 12 bits of the address are used to access the L220 registers. While the AXIv2 protocol allows this configuration, SoC Designer Simulator emits a warning about the mismatch in the address widths, and warnings about non-zero values in the ignored bits: `WARNING :: <CPU> cpu[0]::axi_m is driving non-zero bits in bit positions beyond the address width of the connected slave (32)`

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

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 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 <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 wave-form data, however, the component time is not aligned with the SoC Designer time.	true, false	true	No
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_periph axi_size[0-5]	These parameters are obsolete and should be left at the default values. ²	n/a	size0 default is 0x100000000, size1-5 default is 0	No
axi_periph axi_start[0-5]			0x00000000	No
axi_periph Enable Debug Messages	Whether debug messages are logged for slave port 0.	true, false	false	Yes
axi_s0 axi_size[0-5]	These parameters are obsolete and should be left at the default values. ²	n/a	size0 default is 0x100000000, size1-5 default is 0	No
axi_s0 axi_start[0-5]			0x00000000	No
axi_s0 Enable Debug Messages	Whether debug messages are logged for slave port 0.	true, false	false	Yes
axi_s1 axi_size[0-5]	These parameters are obsolete and should be left at the default values. ²	n/a	size0 default is 0x100000000, size1-5 default is 0	No
axi_s1 axi_start[0-5]			0x00000000	No

Table 1-3 Component Parameters (continued)

axi_s1 Enable Debug Messages	Whether debug messages are logged for slave port 1.	true, false	false	Yes
Carbon DB Path	Sets the directory path to the data-base file.	Not Used	empty	No
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
Waveform File ³	Name of the waveform file.	<i>string</i>	arm_cm_L220.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. 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*.
3. 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 L220 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 L220. This is used when one wants to look at the memory content from a CPU Core connected to the L220. The L220 tries to locate the target address in its internal buffers, its data RAM and propagates the debug operation to its master ports.

1.5.2 Register Information

This section lists the Registers available for the L220 Cycle Model. Registers are grouped according to functional area into different sets. The available groups are:

- [r0 Registers](#)
- [r1 Registers](#)
- [r2 Registers](#)
- [r7 Registers](#)
- [r9 Registers](#)

See the *ARM Level 2 Cache Controller (L220) Technical Reference Manual* for details about these registers.

1.5.2.1 r0 Registers

Table 1-4 shows the Cache registers.

Table 1-4 Cache ID and Cache Type Registers

Name	Base Offset	Type	Reset Value	Description
CACHEID	0x000	RO	0x41000086 ¹	Cache ID Register
CACHETYPE	0x004	RO	0x1c100100 ²	Cache Type Register

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

1.5.2.2 r1 Registers

Table 1-5 shows the Control registers.

Table 1-5 Control Registers

Name	Base Offset	Type	Reset Value	Description
CTRL	0x100	RW	0x00000000	Control Register
AUXCTRL	0x104	RW	0x02020fff ¹	Auxiliary Control Register

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

1.5.2.3 r2 Registers

Table 1-6 shows the Event Counter Control and Interrupt Control registers.

Table 1-6 Event Counter Control and Interrupt Control Registers

Name	Base Offset	Type	Reset Value	Description
EVTCTRCTRL	0x200	RW	0x00000000	Event Counter Control Register
EVTCTR1CNFG	0x204	RW	0x00000000	Event Counter 1 Configuration Register
EVTCTR0CNFG	0x208	RW	0x00000000	Event Counter 0 Configuration Register
EVTCTR1VAL	0x20C	RW	0x00000000	Event Counter 1 Value Register
EVTCTR0VAL	0x210	RW	0x00000000	Event Counter 0 Value Register
INTRMASK	0x214	RW	0x00000000	Interrupt Mask Register
MASKINTRSTATUS	0x218	RO	0x00000000	Masked Interrupt Status Register
RAWINTRSTATUS	0x21C	RO	0x00000000	Raw Interrupt Status Register

1.5.2.4 r7 Registers

Table 1-7 shows the Cache Maintenance Operations registers.

Table 1-7 Maintenance Registers

Name	Base Offset	Type	Reset Value	Description
CACHESYNC	0x730	RO	0x00000000	Cache Sync Operations Register

1.5.2.5 r9 Registers

Table 1-8 shows the Cache Lockdown registers.

Table 1-8 Lockdown Registers

Name	Base Offset	Type	Reset Value	Description
DATALOCK	0x900	RW	0x00000000	Data Lockdown Register
INSTRLOCK	0x904	RW	0x00000000	Instruction Lockdown Register

1.5.3 Memory View

The memory view that combines the data RAM in the L220 Cycle Model and the external Level-3 memory seen from the AXI master ports on the L220. If a certain address is cached (either in the cache RAM or in one of the internal buffers), the value is read from the cache. Otherwise, the value is obtained from Level-3 memory. To jump to an address, enter the desired value into the *Address* field. The *Block* combo box entry has no function in this view and can be ignored.

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 L220 Cycle Model profiles the following events:

- CO (Eviction)
- DRHIT (Data read hit)
- DRREQ (Data read request)
- DWHIT (Data write hit)
- DWREQ (Data write request)
- DWTREQ (Data write-through request)
- IRHIT (Instruction read hit)
- IRREQ (Instruction read request)
- WA (Write allocate)