CoreLink™ CCN-502 Cycle Model

Version 9.2.0

User Guide

Non-Confidential

arm

101238_0902_00 (ID020918)

CoreLink™ CCN-502 Cycle Model
User Guide

Release Information

The following changes have been made to this document.

Change History

Issue	Date	Confidentiality	Change
0902-00	Feb 2018	Non-Confidential	Doc update to release 9.2

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

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 CCN-502 (PL510) 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();`

Convention	Description	Example
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

The following publications provide information that relate directly to SoC Designer:

- *SoC Designer User Guide* (100996)

The following publications provide reference information about additional Arm products:

- *Arm CoreLink CCN-502 Cache Coherent Network Technical Reference Manual* (100052)
- *AMBA® Specification* (IHI0011)
- *Architecture Reference Manual* (DDI0403, DDI0553)

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 1.4, 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 or VHDL.
RTL	*Register Transfer Level*. A high-level hardware description language (HDL) for defining digital circuits.
SoC Designer	A 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 Component in SoC Designer

This chapter describes the functionality of the CCN-502 Cycle Model and how to use it in SoC Designer. It contains the following sections:

1.1 CCN-502 Cache Coherent Network Functionality

The CCN-502 Cycle Model simulates the Arm CoreLink CCN-502 Cache Coherent Network. The Cycle Model implements the full set of features supported by the RTL; any exceptions are noted in the appropriate section of this guide.

For details about the functionality of the hardware that the Cycle Model simulates, refer to the *CoreLink CCN-502 Cache Coherent Network Technical Reference Manual* (100052).

1.1.1 DSSB support

If DSSB support is selected during Cycle Model configuration time, the Cycle Model:

- Includes the DSSB logic as well as DSSB Register Slices logic.
- Includes the clock signal GCLKCD_NID* (where NID* represents the NodeID of the port that supports DSSB).

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 in Table 1-1:

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. See "Available Component ESL Ports" on page 14 for a list of ports; note that the presence of some ports depends on the configuration chosen when the IP was initially built, as well as the r*x*p*y* version.

1.3 Available Component ESL Ports

Table 1-2 describes the CCN-502 Cycle Model ESL Transactor ports that are exposed in SoC Designer; additional ports (pins) are described in the CCN-502 Technical Reference Manual. Note that port and protocol availability is configuration-specific.

Table 1-2 Transactor Component Ports

ESL Port	Description
ACE_Lite_DVM_S*_NID*	input ACE_Lite/ACE_Lite+DVM RN-I ports
CHI_RNF_NID*	input RNF CHI ports
ACE_Lite_M_NID0	output AXI4/ACE-Lite master port
AXI4_M_NID*	output AXI4 SN-F ports
CHI_SNF_NID*	output CHI SN-F ports

Refer to the section DSSB support for information about ports that support DSSB functionality.

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	Init/ Runtime
acchannelen_rnf	Bitmap for each rnf upstream port to determine if it is enabled for snoop requests.	0 - 0xf	0xf	Init
acchannelen_rni	Bitmap for each rni upstream port to determine if it is enabled or not for DVM requests.	0 - 0x1ff	0	Init
ACLKEN_M_NID*m*	AXI Master bus clock enable	0, 1	1	Runtime
ACLKEN_S*n*_NID*m*	AXI bus clock enable	0, 1	1	Runtime
Align Waveforms	When set to *true*, waveforms dumped by 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 SoC Designer time.	true, false	true	Init
ARM Cycle Models DB Path	Sets the directory path to the database file.	Not used	empty	n/a
CACTIVE_S*n*_NID*m*	Indication that master device is active.	0, 1	0	Runtime
CHI_RNF_NID*m* Enable Debug Messages	Determines whether debug messages are logged for the component.	true, false	false	Runtime
CHI_SNF_NID*m* Enable Debug Messages	Determines whether debug messages are logged for the component.	true, false	false	Runtime
CHI_RNF_NID*m* Protocol Variant	Protocol variant of the corresponding CHI_RNF port. This setting reflects the configuration choice at build time and can not be changed in SoC Designer.	CHI-RNF	CHI-RNF	Runtime
CHI_SNF_NID*m* Protocol Variant	Protocol variant of the corresponding CHI_SNF port. This setting reflects the configuration choice at build time and can not be changed in SoC Designer.	CHI-SNF	CHI-SNF	Runtime

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Init/ Runtime
CLREXMONACK_NID*m*	Indicates that an exclusive monitor in the CCN-502 has been cleared.	0, 1	0	Runtime
DBGWATCHTRIGACK	External acknowledgment of receipt of DBGWATCHTRIGREQ.	0, 1	0	Runtime
DCLKEN	Debug clock enable, used to control the clock for the STMHWEVENT interface.	0, 1	0	Runtime
Dump Waveforms	Whether SoC Designer dumps waveforms for this component	true, false	false	Init
Fast Application Load Support	Identifies that the component supports fast debug access for application load.	Not configurable	Yes	N/A
HNF3_SAM_CONTROL HNF5_SAM_CONTROL HNF9_SAM_CONTROL HNF11_SAM_CONTROL HNF13_SAM_CONTROL	The parameter value overwrites the reset value of the corresponding register. This facilitates Swap & Play support in the component as well as manual loading of the AXF image into the memory at system start up. The parameter default value matches the reset value of the corresponding register in the particular configuration. The parameter should be set to the value with which the application initializes the corresponding register.	Value	Configuration-dependent	Init
PERIPHBASE	Base address for CCN-502 programmable registers. Bits 23:0 are ignored.	integer	0	Init
PMUSNAPSHOTREQ	External request that the live PMU counters are snapshotted to the shadow registers.	0, 1	0	Runtime
PREQ_L3RAM*n*	Indicates a request for a power state transition.	0, 1	0	Runtime
PSTATE_L3RAM*n*	The power state to which a transition is requested.	0-3	0	Runtime
PSTATE_LOGIC	The power state to which a transition is requested.	0, 1	1	Runtime
PSTATE_SF	The power state to which a transition is requested.	0-3	3	Runtime
QREQ*n*_CLKCTL	Request from the ExtCC for the CCN-502 to prepare to stop the clocks	0, 1	1	Runtime
SAMHNF*n*NODEID	HN-F *n* Node ID. CCN-502 has a fixed-node topology; do not change the default value.	—	differs per node	Init
SAMHNFMODE	Number of HN-Fs. Depends on the selected CCN partition. Do not change the default value.	1, 2	1, 2	Init
SAMHNI*n*NODEID	HN-I *n* Node ID. CCN-502 has a fixed-node topology; do not change the default value.	—	differs per node	Init

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Init/ Runtime
SAMMNNODEID	MN Node ID. Configuration-dependent. CCN-502 has a fixed-node topology; do not change the default value.	0	0	Init
SBSX_128_n256	Data width on SBSX AMBA interface. Configuration-dependent.	0, 1	0	Init
systemaddrmap	Bitmap for 20 regions in CCN interconnect. Every two bits describes the region type for the corresponding region. Values are grouped in the format {MAP19}…{MAP0}, a total of 40-bit wide hex value. For example: — MAP0=1, MAP1=1: in binary format: 0101, in hex format: 0x5 — MAP1=1, MAP4=1: in binary format: 01_0000_0100, in hex format: 0x104 — MAP19=1: in binary format: 0100_0000_0000_0000_0000_0000_0000_0000_0000_0000, in hex format: 0x40_0000_0000	*integer*	0	Init
Waveform file	Name of the waveform file.	*string*	arm_cm_ PL502.vcd	Init
Waveform format	The format of the waveform dump file.	VCD, FSDB	VCD	Init
Waveform timescale	Sets the timescale to be used in the waveform.	Set of values in pulldown menu.	1 ns	Init

1.5 Debug Features

The CCN-502 Cycle Model has a debug interface that allows the user to view, manipulate, and control the registers and memory. To access the memory and register views in SoC Designer, right click on the Cycle Model and choose the appropriate menu entry.

1.5.1 Debug Read and Write Operations

The debug read and write operations are initiated by a master connected to the AXI slave ports of the CCN-502. These operations are used when the memory content is examined or modified from a device model (typically a CPU core) connected to the CCN-502, or to a component upstream on the slave side. The CCN-502 directs debug read/ write requests to the appropriate ports.

This component supports full system coherent memory views. It requests read/write of the memory location across all of its slaves to the connected components, and then if required (based on the memory map defined by the current settings of the ADDRMAPx parameters), it sends the request through the appropriate master port. Refer to the *SoC Designer User Guide* for details about full system coherent memory views.

1.5.2 Register Information

The Cycle Model supports the full set of Arm registers, with the limitations noted below:

- Debug Write access to Write-Only registers is not supported in the Register View and via the CADI interface. Write transactions to these registers that are initiated by the AXI/CHI bus masters connected to CCN502 are supported.

Refer to the *Arm CoreLink CCN-502 Cache Coherent Network Technical Reference Manual* (100052) for details about supported registers.

1.6 Hardware Profiling

Hardware events are uniquely identified by their Event Number as defined in the *Arm CoreLink CCN-502 Cache Coherent Network Technical Reference Manual* (100052).

Table 1-4 Profiling Events

Stream	Event Name	Comments
HN-F	HNF_0x1_CACHE_MISS	Total cache misses.
	HNF_0x2_L3_SF_CACHE_ACCESS	Total number of cache accesses.
	HNF_0x3_CACHE_FILL	Total allocations in HN L3 cache.
	HNF_0x4_POCQ_RETRY	Total number of requests that have been retried.
	HNF_0x5_POCQ_REQS_RECVD	Total number of requests received by the HN.
	HNF_0x6_SF_HIT	Total number of snoop filter hits.
	HNF_0x7_SF_EVICTIONS	Total number of snoop filter evictions.
	HNF_0x8_SNOOPS_SENT	Number of snoops sent. Does not differentiate between broadcast or directed snoops.
	HNF_0x9_SNOOPS_BROADCAST	Number of snoop broadcasts sent.
	HNF_0xA_L3_EVICTION	Number of L3 evictions.
	HNF_0xB_L3_FILL_INVALID_WAY	Number of L3 fills to an invalid way.
	HNF_0xC_MC_RETRIES	Number of requests receiving retry response from the memory controller.
	HNF_0xD_MC_REQS	Total number of requests that are sent to the memory controller.
	HNF_0xE_QOS_HH_RETRY	Number of times HN-F protocol retried a QoS 15 (highest) class request.
XP	XP_<*signal*>_Bus<*n*>_<*addr*>_UPLOAD_STARVATION	Upload starvation. Signaled when this XP sets the Hbit, per-VC, per-direction.
	XP_<*signal*>_Bus<*n*>_<*addr*>_DOWNLOAD_STARVATION	Download starvation. Signaled when this XP sets the S-bit, per-VC, per-direction.
	XP_<*signal*>_Bus<*n*>_<*addr*>__RESPIN	Respin. Signaled when this XP sets the P-Cnt, per-VC, per-direction.
	XP_<*signal*>_Bus<*n*>_<*addr*>_VALID_FLIT	A valid flit is passing through the XP, per-VC, perdirection

Table 1-4 Profiling Events (continued)

Stream	Event Name	Comments
RN-I	RNI_0x01_RDATABEATS_P0	S0 RDataBeats
	RNI_0x02_RDATABEATS_P1	S1 RDataBeats.
	RNI_0x03_RDATABEATS_P2	S2 RDataBeats.
	RNI_0x04_RXDATFLITV	RXDAT flits received.
	RNI_0x05_TXDATFLITV	TXDAT flits sent.
	RNI_0x06_TXREQFLITV	Total TXREQ flits sent.
	RNI_0x07_TXREQFLITV_RETRIED	Retried TXREQ flits sent.
	RNI_0x08_RRTFULL	Read request tracker full.
	RNI_0x09_WRTFULL	Write request tracker.
	RNI_0x0A_TXREQFLITV_REPLAYED	Replayed TXREQ flits.
MNHN-I	MN_0x00_EOBARRIER	EOBarrier count. Available through the DWM.
	MN_0x01_ECBARRIER	ECBarrier count. Available through the DWM.
	MN_0x02_DVMOP	DVMOp count. Available through the DWM.
	HNI_0x01_TXDATFLITV	Transmitted data flits. Available through the DWM.
	HNI_0x02_RXDATFLITV	Received data flits. Available through the DWM.
	HNI_0x04_RXREQFLITV	Received requests. Available through the DWM.
	HNI_0x08_RXREQ_REQORDER	Received ReqOrder requests. Available through the DWM.
SBSX	0x01_TXDATFLITV	Transmitted data flits.
	0x02_RXDATFLITV	Received data flits.
	0x04_RXREQFLITV	Received requests.