CoreLink™ Network Interconnect NIC-301 Cycle Model

Version 9.1.0

User Guide

ARM®

ARM DUI1086A (ID121616)

CoreLink Network Interconnect NIC-301
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 Cycle Model, and how to use it in SoC Designer. It contains the following sections:

1.1 NIC-301 Cycle Model Functionality

The AMBA Network Interconnect is a highly configurable component that enables you to create a complete high performance, optimized AMBA-compliant network infrastructure. The possible configurations for the AMBA Network Interconnect can range from a single bridge component, for example an AHB to AXI protocol bridge, to a complex infrastructure that consists of up to 128 masters and 64 slaves of a combination of different AMBA protocols. For information about these components, see the *AMBA Network Interconnect (NIC-301) Technical Reference Manual*.

Use the AMBA Designer Graphical User Interface configuration tool to design your Network Interconnect. You can then generate, test, and profile complex AMBA bus systems in:

- a transaction-level modeling environment
- Verilog

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.

- Implemented Hardware Features
- Unsupported Hardware Features
- Features Additional to the Hardware

1.1.1 Implemented Hardware Features

The AMBA Network Interconnect is a highly configurable infrastructure component that supports:

- 1-128 AXI or AHB-Lite slave interfaces
- 1-64 master interfaces that can be AXI, AHB-Lite, APB2, or APB3
- Configuration of:
  - an APB port to support 1-16 slaves
  - an AXI port to support four region control bits
- Single-cycle arbitration
- Full pipelining to prevent master stalls
- Programmable control for FIFO transaction release
- Multiple switch networks
- Complex topologies, including Network On Chip (NOC) loop-back connections between switches
- Up to five cascaded switch networks between any master and slave interface pair
- AXI or AHB-Lite masters and slaves with:
  - an address width of 32-64 bits
  - a data width of 32, 64, or 128 bits
- Non-contiguous APB slave address map for a single master interface

- Independent widths of user-defined sideband signals for each channel
- Global Programmers View (GPV) for the entire infrastructure that you can configure so that any master, or a discrete configuration slave interface, can access it
- Highly flexible timing closure options
- AMBA Designer tool-based configuration

1.1.2 Unsupported Hardware Features

The following features of the NIC-301 hardware are not implemented in the Cycle Model:

- The hardware can support a data width choice of 256 bits; the Cycle Model supports only 32, 64, or 128 bits.

1.1.3 Features Additional to the Hardware

The following features that are implemented in the NIC-301 Cycle Model do not exist in the NIC-301 hardware. These features have been added to the Cycle Model for enhanced usability.

- The Address Control registers are read-write in the Cycle Model, whereas they are write-only in the hardware.

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 SoC Designer 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.

This figure shows two different components in order to show a variety of available ports. The available ports depend on how the Cycle Model was configured in AMBA Designer; the names are fully customizable in AMBA Designer.

1.3 Available Component ESL Ports

Table 1-2 describes the ESL ports of the component, created by AMBA Designer, that are exposed in SoC Designer. See the *AMBA Network Interconnect (NIC-301) Supplement to AMBA Designer User Guide*. See also the *AMBA® Network Interconnect (NIC-301) Integration Manual* for more information.

Table 1-2 ESL Component Ports

ESL Port	Description	Direction	Type
<*clock_domain_name*>clk	Clock signal, or signals, for each defined clock domain.	input	Clock slave
<*clock_domain_name*>clk_r	This type of port is available if a transactor port was configured to have a view to the GPV. It must have the same clock frequency as the clock domain clock and must be synchronous to it.	input	Clock slave
<*clock_domain_name*>resetn	Reset signal, or signals, for each defined clock domain.	input	Signal slave
<*clock_domain_name*>resetn_r	Reset signal corresponding to the <*clock_domain_name*>clk_r, if one exists.	input	Signal slave
arqos_m_0	This is a 4-bit port that defines the Read QoS option for the slave port. (Only available if “From Master” was selected as the QoS Type in AMBA Designer.)	input	Signal slave
awqos_m_0	This is a 4-bit port that defines the Write QoS option for the slave port. (Only available if “From Master” was selected as the QoS Type in AMBA Designer.)	input	Signal slave
clk-in	Input clock. When using the <*clock_domain_name*>clk port, the *clk-in* port should not be connected.	input	Clock slave
Protocol Type Master Interface Ports	There is a master interface port created for each port defined in AMBA Designer. For example, *slave_00*.	master	AXI, AHB_LITE, APB
Protocol Type Slave Interface Ports	There is a slave interface port created for each port defined in AMBA Designer. For example, *master_00*.	slave	AXI, AHB_LITE

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 <clock_domain_name>resetn and <clock_domain_name>resetn_r ports. 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.

1.4 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 *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
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	Whether debug messages are logged for the component.	true, false	false	Yes
<*clock_domain_name*> resetn	Sets the value for the specified Reset signal, or signals, corresponding to the specified clock domain.	0x0, 0x1	0x1	Yes
<*clock_domain_name*> resetn_r	If a <*clock_domain_name*>resetn_r signal exists, this parameter sets the value for the specified Reset signal, or signals, corresponding to the specified clock domain.	0x0, 0x1	0x1	Yes
<*slave port name*>_ size[0-5] [2]	Sizes of memory regions.	0x0 - 0x100000000	size0 default is 0x100000000, size1-5 default is 0	No
<*slave port name*>_ start[0-5] [2]	Start addresses of memory regions.	0x0 - 0xffffffff	0x00000000	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
<*master port name*> Enable Debug Messages	Whether debug messages are logged for the master ports. There is one parameter for each master port.	true, false	false	Yes
<*slave port name*> Enable Debug Messages	Whether debug messages are logged for the slave ports. There is one parameter for each slave port.	true, false	false	Yes
Track In-Flight Data[3]	Enables or disables debug access for in-flight transactions.	true, false	true	No
Waveform File [4]	Name of the waveform file.	*string*	arm_cm_pl301_ NIC301<*component_name*>.vcd	No
Waveform Format	The format of the waveform dump file.	VCD	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. The square brackets indicate the memory regions available for the Cycle Model. For example, the parameter name for the start addresses “s_[0-1]_start[0-5]” is expanded to 12 possible parameter name combinations that range from “s_0_start0” to “s_1_start5”. The size of a memory region depends on the “s[N]_start[M]” and “s[N]_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.
3. See Using the Track In-Flight Data parameter below for more about this parameter.
4. 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.

1.4.1 Using the Track In-Flight Data parameter

The Track In-Flight Data parameter should be set to True (this is the default) if you are using any `debugAccess` functions in the NIC-301. `debugAccess` functions are used to implement the Memory view of a processor and any Disassembly view of memory. If in-flight tracking is disabled, then the information in one of these views may be incorrect while there are incomplete write transactions within the NIC301.

Disabling Track In-Flight Data may be appropriate if the accuracy of a processor’s Disassembly view or Memory view is not critical. Whether or not Track In-Flight Data is enabled, opening a Memory view from the desired memory always provides an accurate view of its contents.

1.5 Debug Features

The NIC-301 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.

1.5.1 Register Information

The NIC-301 Cycle Model has sets of registers that are accessible via the debug interface. The slave interface, master interface, and internal interface port registers are described in the Interface Block registers section.

The registers are listed below:

- Address Control Registers
- Peripheral ID Registers
- Interface Block Registers

1.5.1.1 Address Control Registers

Table 1-4 shows the Address Control registers.

Table 1-4 Address Control Registers

Name	Description	Type
remap	Remap register, up to eight global remap states are available.	read-write
security0	Slave 0 security setting. 1 bit for non-virtual slaves, up to 16 for virtual or APB master interfaces, and you can configure it as follows: 0 Secure 1 Non-secure	read-write
security1	Slave 1 security setting. 1 bit for non-virtual slaves, up to 16 for virtual or APB master interfaces, and you can configure it as follows: 0 Secure 1 Non-secure	read-write
security<*n*>	Slave *n* security setting. 1 bit for non-virtual slaves, up to 16 for APB master interfaces.	read-write

1.5.1.2 Peripheral ID Registers

If you configure any registers in the programmers view, peripheral ID registers are always visible. This provides a low gate count option for identification. Table 1-5 shows the peripheral ID registers.

Table 1-5 Peripheral ID Registers

Name	Description	Type
Peripheral_ID0	Peripheral Identification Register 0 - Part Number	read-only
Peripheral_ID1	Peripheral Identification Register 1 - JEP106 and Part Number	read-only
Peripheral_ID2	Peripheral Identification Register 2 - Revision, JEP106 code flag, JEP106	read-only
Peripheral_ID3	Peripheral Identification Register 3 - Can be set using the AMBA Designer Graphical User Interface (GUI)	read-only
Peripheral_ID4	Peripheral Identification Register 4 - 4KB count, JEP106 continuation code	read-only
Peripheral_ID5	Peripheral Identification Register 5 - Reserved	read-only
Peripheral_ID6	Peripheral Identification Register 6 - Reserved	read-only
Peripheral_ID7	Peripheral Identification Register 7 - Reserved	read-only
Component_ID0	Component Identification Register 0 - Preamble	read-only
Component_ID1	Component Identification Register 1 - Generic IP component class, preamble	read-only
Component_ID2	Component Identification Register 2 - Preamble	read-only
Component_ID3	Component Identification Register 3 - Preamble	read-only

1.5.1.3 Interface Block Registers

One register tab exists for each Interface Block (IB), where the IB can be:

- Slave Interface Block (ASIB)
- Master Interface Block (AMIB)
- Internal network Interface Block (IB)

Table 1-6 shows the Interface Block registers for the ASIB, AMID, and IB. Note that a tab may be provided for each defined port (only for those ports that exist in the HDL). The tab shows the type of port (SI, MI, or II) and the name of the port. Note that not all the registers are available for each IB type, or for each protocol type (AHB, AXI, APB).

Table 1-6 Interface Block Registers

Name	Description	Type
sync_mode	This 3-bit register is valid only with a FIFO for all channels. You can configure the bits to create different clock domain boundaries as follows: 0 sync 1:1 1 sync *n*:1 2 sync 1:*n* 3 sync *m*:*n* 4 async 5 reserved 6 reserved 7 reserved	read-write
fn_mod	This 1-bit register is the issuing functionality modification register. Issuing override sets block issuing capability to be forced to one transaction, and you can configure the bit as follows: 0 Read issuing, *read_iss_override* 1 Write issuing, *write_iss_override*	read-write
fn_mod2	Bypass merge, only if upsizing or downsizing. See the Upsizing data width function and Downsizing data width function sections in the TRM for more information.	read-write
fn_mod_ahb	This 3-bit register is valid for AHB interfaces only. You can configure the bits of this register as follows: bit 0 rd_incr_override bit 1 wr_incr_override bit 2 lock_override In the TRM, see Lock transactions for information on overriding locks, and see Combination 4 for information on *wr_incr_override* and *rd_incr_override*.	read-write
fn_mod_iss_bm	See the TRM for more information.	read-write
ahb_cntl	This 2-bit register is valid for AHB interfaces only. You can configure the bits as follows: bit 0 decerr_en bit 1 force_incr See AHB master interfaces in the TRM for more information.	read-write

Table 1-6 Interface Block Registers (continued)

Name	Description	Type
wr_tidemark	This 4-bit register is valid only with a FIFO for the WFIFO channel, and if not an AHB slave interface. See the FIFO and clocking function section in the TRM for more information.	read-write
read_qos	This 4-bit register is available in the ASIB only. This is the Read channel QoS value.	read-write
write_qos	This 4-bit register is available in the ASIB only. This is the Write channel quality value.	read-write
The following registers are only available for interface blocks that were configured with the QoS extensions that are provided by the QoS Extension package for the NIC301. This configuration must be done while defining the NIC301.		
qos_cntl	QoS Control Register	read-write
max_ot	Maximum number of outstanding transactions Register	read-write
max_comb_ot	Maximum number of combined transactions Register	read-write
aw_p	AW channel peak rate Register	read-write
aw_b	AW channel burstiness allowance Register	read-write
aw_r	AW channel average rate Register	read-write
ar_p	AR channel peak rate Register	read-write
ar_b	AR channel burstiness allowance Register	read-write
ar_r	AR channel average rate Register	read-write
tgt_latency	Target latency Register	read-write
ki	Latency regulation Register	read-write
qos_range	QoS range Register	read-write

1.6 Available Profiling Data

The NIC-301 Cycle Model component has no profiling capabilities.