CoreLink™ DMA Controller DMA-330 Cycle Model

Version 9.1.0

User Guide

Non-Confidential

ARM DUI 1073A (ID120616)

CoreLink DMA Controller DMA-330 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

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 component, and how to use it in SoC Designer. It contains the following sections:

- DMA-330 Cycle Model Functionality
- Adding and Configuring the SoC Designer Component
- Available Component ESL Ports
- Setting Component Parameters
- Debug Features
- Available Profiling Data

1.1 DMA-330 Cycle Model Functionality

The DMA-330 Cycle Model is a high-performance, area-optimized SDRAM or Mobile SDR memory controller that provides an AXI interface for DMA transfers. It is programmed and controlled by two APB interfaces, one operating in the TrustZone secure mode, and the other operating in the non-secure mode. For a detailed description of the AXI protocol refer to the *AMBA AXI Protocol Specification*. For a detailed description of the APB protocol refer to the *AMBA APB Protocol Specification*.

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 represents, refer to the *ARM CoreLink DMA Controller DMA-330 Technical Reference Manual*.

Use the AMBA Designer Graphical User Interface configuration tool to design your DMAC. 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.

1.1.1 Implemented Hardware Features

The DMAC provides the following features:

- an instruction set that provides flexibility for programming DMA transfers
- single AXI master interface that performs the DMA transfers
- dual APB slave interfaces, designated as secure and non-secure, for accessing registers in the DMAC
- supports TrustZone technology
- supports multiple transfer types:
  - memory-to-memory
  - memory-to-peripheral
  - peripheral-to-memory
  - scatter-gather
- configurable RTL that enables the DMAC to be optimized for the application
- programmable security state for each DMA channel
- signals the occurrence of various DMA events using the interrupt output signals
- AMBA Designer tool-based configuration

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.

For the examples in this guide, two peripheral interfaces have been defined, indicated by the interface names, which have *_0* and *_1* suffixes, for example *daready_0* and *daready_1*. Your component may appear with fewer or more interfaces, depending on how it was configured using AMBA Designer. See the *CoreLink DMA Controller DMA-330 Supplement to AMBA Designer User Guide* for information about the features you can configure.

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 *CoreLink DMA Controller DMA-330 Integration Manual* for more information.

Table 1-2 ESL Component Ports

Name	Description	Direction	Type
apbns	Non-secure APB interface.	slave	APB transaction slave
apbs	Secure APB interface.	slave	APB transaction slave
boot_addr	The address location that contains the first instruction that the DMAC execute.	input	Signal slave
boot_from_pc	A 1-bit signal to control whether or not the DMAC will try to execute instructions located at *boot_addr* when it comes out of reset.	input	Signal slave
boot_irq_ns	The security state of the interrupt outputs.	input	Signal slave
boot_manager_ns	The security state of the DMA manager thread.	input	Signal slave
boot_periph_ns	The security state of the peripheral request interface.	input	Signal slave
daready_*x* [1]	Peripheral ready.	input	Signal slave
drlast_*x* [1]	Last data transfer.	input	Signal slave
drtype_*x* [1]	Peripheral request/acknowledgement type.	input	Signal slave
drvalid_*x* [1]	Peripheral control valid.	input	Signal slave
pclken	Clock enable for APB interfaces.	slave	Signal slave
resetn	Reset source.	input	Signal slave
clk-in	AXI input clock.	slave	Clock slave
axi	AXI master interface used to transfer data from a source AXI slave to a destination AXI slave.	master	AXI transaction FlowThru master
datype_*x* [1]	Acknowledgement type.	output	Signal master
davalid_*x* [1]	Control valid.	output	Signal master
drready_*x* [1]	DMAC ready.	output	Signal master
irq	Interrupt request.	output	Signal master
irq_abort	Interrupt abort.	output	Signal master

1. Where *x* is a number 0 through n, depending on the number of peripheral interfaces defined in AMBA Designer.

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 boot_addr, boot_from_pc, boot_irq_ns, boot_manager_ns, boot_periph_ns, and pclken 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
apbns Base Address	Base address of the non-secure APB interface.	0x0 – 0xffffffff	0x0	No
apbns Enable Debug Messages	Whether debug messages are logged for the non-secure APB port.	true, false	false	Yes
apbns Size	Region size of the non-secure APB interface.	0x0 – 0x100000000	0x100000000	No
apbs Base Address	Base address of the secure APB interfaces.	0x0 – 0xffffffff	0x0	No
apbs Enable Debug Messages	Whether debug messages are logged for the secure APB port.	true, false	false	Yes
apbs Size	Region size of the secure APB interface.	0x0 – 0x100000000	0x100000000	No
axi Enable Debug Messages	Whether debug messages are logged for the AXI port.	true, false	false	Yes

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
boot_addr	Tie-off value for boot_addr[31:0].	0x0 – 0x100000000	0x0	Yes
boot_from_pc	Tie-off value for boot_from_pc.	true, false	false	Yes
boot_irq_ns	Tie-off value for boot_irq_ns.	0x0 – 0x100000000	0x0	Yes
boot_manager_ns	Tie-off value for boot_manager_ns.	0x0 – 0x100000000	0x1	Yes
boot_periph_ns	Tie-off value for boot_periph_ns.	0x0 – 0x100000000	0x0	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
pclken	Tie-off value for pclken.	0x0 – 0x100000000	0x0	Yes
Waveform File [2]	Name of the waveform file.	*string*	arm_cm_pl330_<*component_name*>.fsdb	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. 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 DMA-330 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 DMA-330 Cycle Model has five sets of registers that are accessible via the debug interface. Registers are grouped into sets according to functional area. The registers are listed below:

- Control Registers
- Channel Registers
- Debug Registers
- Configuration Registers
- Peripheral and PrimeCell Identification Registers

See the *ARM CoreLink DMA Controller DMA-330 Technical Reference Manual* for detailed descriptions of these registers.

1.5.1.1 Control Registers

The Control group contains registers that control the DMAC.

Table 1-4 Control Registers Summary

Name	Description	Type
DSR	DMA Manager Status register	read-only
DPC	DMA program counter register	read-only
INTEN	Interrupt enable register	read-write
INT_EVENT_RIS	Event-Interrupt Raw Status register	read-only
INTMIS	Interrupt status register	read-only
INTCLR	Interrupt clear register	write-only
FSRD	Fault status DMA manager register	read-only
FSRC	Fault status DMA channel register	read-only
FTRD	Fault type DMA manager register	read-only
FTR[0-7] [1]	Fault type for DMA channels 0 through 7	read-only

1. FTR0 - FTC7 registers appear only when the DMA-330 is configured to support more than one DMA channel. For example, if the DMA-330 is configured to support three channels, FTR0, FTR1, and FTR2 registers will exist.

1.5.1.2 Channel Registers

The Channel group contains registers for each DMA channel. They provide the status of the DMA channel threads. The contents of Table 1-5 are replicated for a maximum of eight channels.

Table 1-5 Channel Registers Summary

Name	Description	Type
CSR[0-7] [1]	Channel status	read-only
CPC[0-7] [1]	Channel Program Counter (PC)	read-only
SAR[0-7] [1]	Source address	read-only
DAR[0-7] [1]	Destination address	read-only
CCR[0-7] [1]	Channel control	read-only
LC0_[0-7] [1]	Loop counter 0	read-only
LC1_[0-7] [1]	Loop counter 1	read-only

1. Registers above zero appear only when the DMA-330 is configured to support more than one DMA channel.

1.5.1.3 Debug Registers

The DMAC Debug group contains registers that enable you to send instructions to a thread when debugging the program code, or enable system firmware to send instructions to the DMA manager thread.

Table 1-6 Debug Registers Summary

Name	Description	Type
DBGSTATUS	Debug status	read-only
DBGCMD	Debug command	write-only
DBGINST0	Debug instruction 0	write-only
DBGINST1	Debug instruction 1	write-only

1.5.1.4 Configuration Registers

The Configuration group contains registers that enable system firmware to discover the configuration of the DMAC.

Table 1-7 Configuration Registers Summary

Name	Description	Type
CR0	Configuration register 0	read-only
CR1	Configuration register 1	read-only
CR2	Configuration register 2	read-only
CR3	Configuration register 3	read-only

Table 1-7 Configuration Registers Summary

Name	Description	Type
CR4	Configuration register 4	read-only
CRD	DMA Configuration register	read-only
WD	Watchdog register	read-write

1.5.1.5 Peripheral and PrimeCell Identification Registers

The Peripheral and PrimeCell group contains registers that enable system firmware to identify a PrimeCell component.

Table 1-8 Peripheral and PrimeCell Registers Summary

Name	Description	Type
periph_id_0	Peripheral ID 0	read-only
periph_id_1	Peripheral ID 1	read-only
periph_id_2	Peripheral ID 2	read-only
periph_id_3	Peripheral ID 3	read-only
pcell_id_0	PrimeCell ID 0	read-only
pcell_id_1	PrimeCell ID 1	read-only
pcell_id_2	PrimeCell ID 2	read-only
pcell_id_3	PrimeCell ID 3	read-only

1.6 Available Profiling Data

The DMA-330 Cycle Model component has no profiling capabilities.