CoreLink™ GIC-500 Cycle Model

Version 9.0.0

User Guide

Non-Confidential
# CoreLink™ GIC-500 Cycle Model
## User Guide

Copyright © 2016 ARM Limited. All rights reserved.

### Release Information

The following changes have been made to this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 2016</td>
<td>A</td>
<td>Non-Confidential</td>
<td>Initial Release</td>
</tr>
<tr>
<td>November 2016</td>
<td>B</td>
<td>Non-Confidential</td>
<td>Release 9.0.0</td>
</tr>
</tbody>
</table>

### 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

Preface

About This Guide .......................................................... 7
Audience ........................................................................ 7
Conventions ...................................................................... 8
Further reading ............................................................... 8
Glossary ......................................................................... 9

Chapter 1. Using the Cycle Model Component in SoC Designer

GIC-500 Cycle Model Functionality ..................................... 12
  Hardware Features not Implemented ............................... 12
  Features Additional to the Hardware .............................. 13
Adding and Configuring the SoC Designer Component .......... 13
  SoC Designer Component Files ....................................... 13
  Adding the Cycle Model to the Component Library .......... 14
  Adding the Component to the SoC Designer Canvas .......... 14
Available Component ESL Ports ........................................ 15
Setting Component Parameters ......................................... 16
Debug Features ............................................................... 18
Available Profiling Data .................................................. 18
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 GIC-500 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:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>courier</td>
<td>Commands, functions, variables, routines, and code examples that are set apart from ordinary text.</td>
<td><code>sparseMem_t SparseMemCreateNew();</code></td>
</tr>
<tr>
<td>italic</td>
<td>New or unusual words or phrases appearing for the first time.</td>
<td><em>Transactors</em> provide the entry and exit points for data ...</td>
</tr>
<tr>
<td>bold</td>
<td>Action that the user performs.</td>
<td>Click <a href="#"><em>Close</em></a> to close the dialog.</td>
</tr>
<tr>
<td>&lt;text&gt;</td>
<td>Values that you fill in, or that the system automatically supplies.</td>
<td><code>&lt;platform&gt;/</code> represents the name of various platforms.</td>
</tr>
<tr>
<td>[ text ]</td>
<td>Square brackets <code>[ ]</code> indicate optional text.</td>
<td><code>$CARBON_HOME/bin/modelstudio [ &lt;filename&gt; ]</code></td>
</tr>
<tr>
<td>[ text1</td>
<td>text2 ]</td>
<td>The vertical bar `</td>
</tr>
</tbody>
</table>

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 Installation Guide*
- *SoC Designer User Guide*

The following publications provide reference information about other ARM® products:

- *ARM CoreLink™ GIC-500 Generic Interrupt Controller Technical Reference Manual*
- *ARM Generic Interrupt Controller Architectural Specification, Architecture Version 2.0*

See [http://infocenter.arm.com/help/index.jsp](http://infocenter.arm.com/help/index.jsp) for access to ARM documentation.

The following publications provide additional information on simulation:

**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 Component 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:

- GIC-500 Cycle Model Functionality
- Adding and Configuring the SoC Designer Component
- Available Component ESL Ports
- Setting Component Parameters
- Debug Features
- Available Profiling Data
1.1 GIC-500 Cycle Model Functionality

The GIC-500 Cycle Model supports the following features and functionality of the hardware:

- Multiprocessor environments with up to 128 cores.
- Up to 32 affinity-level 1 clusters.
- Up to eight cores for each cluster.
- The following interrupt types:
  - Locality-specific Peripheral Interrupts (LPIs). These interrupts are generated by a peripheral writing to a memory-mapped register in the GIC-500.
  - 32-960 Shared Peripheral Interrupts (SPIs), in increments of 32.
  - 16 Private Peripheral Interrupts (PPIs), that are independent for each core and can be programmed to support either edge-triggered or level-sensitive interrupts.
  - 16 SGIs, that are generated either by using software to write to GICD_SGIR or through the GIC CPU interface of a core.
- Interrupt Translation Service (ITS). This provides device isolation and ID translation for message-based interrupts, which allows virtual machines to program devices directly.
- Memory-mapped access to all registers.
- Interrupt masking and prioritization.
- Programmable interrupt routing that is based on affinity.
- Three different interrupt groups, which allow interrupts to target different Exception levels:
  - Group 0.
  - Non-secure Group 1.
  - Secure Group 1.
- A global Disable Security (DS) bit. This allows support for systems with and without security.
- 32 priority values (five bits for each interrupt).

For details about the functionality of the hardware that the Cycle Model represents, refer to the ARM CoreLink™ GIC-500 Generic Interrupt Controller Technical Reference Manual.

1.1.1 Hardware Features not Implemented

The following features of the GIC-500 hardware are not implemented in the Cycle Model:

- Register views are not supported with this release.
- Memory views are not supported with this release.
1.1.2 Features Additional to the Hardware

The following features that are implemented in the GIC-500 Cycle Model do not exist in the GIC-500 hardware. These features have been added to the Cycle Model for enhanced usability.

- The component supports negative-level interrupt signals (both input and output). This is configurable using the negLogic parameter (see Table 1-3 on page 16).
- Waveform dumping using the waveform-related parameters described in Table 1-3 on page 16.

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

<table>
<thead>
<tr>
<th>Platform</th>
<th>File Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>maxlib.lib&lt;model_name&gt;.conf</td>
<td>SoC Designer configuration file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx.so</td>
<td>SoC Designer component runtime file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx_DBG.so</td>
<td>SoC Designer component debug file</td>
</tr>
<tr>
<td>Windows</td>
<td>maxlib.lib&lt;model_name&gt;.windows.conf</td>
<td>SoC Designer configuration file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx.dll</td>
<td>SoC Designer component runtime file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx_DBG.dll</td>
<td>SoC Designer component debug file</td>
</tr>
</tbody>
</table>
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 the 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<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. Depending on your configuration, ports may differ slightly from those listed in Table 1-2 on page 15.
1.3 Available Component ESL Ports

Table 1-2 describes the GIC-500 ESL ports that are exposed in SoC Designer. See the *ARM CoreLink™ GIC-500 Generic Interrupt Controller Technical Reference Manual* for more information.

**Table 1-2  GIC-500 ESL Component Ports**

<table>
<thead>
<tr>
<th>ESL Port</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI4Stream_AXI4_Stream0n_M</td>
<td>GIC stream master interface.</td>
<td>Transaction Master</td>
</tr>
<tr>
<td>AXI4Stream_AXI4_Stream0n_S</td>
<td>GIC stream slave interface.</td>
<td>Transaction Slave</td>
</tr>
<tr>
<td>AXI4_AXI_M</td>
<td>AXI4 master interface.</td>
<td>Transaction Master</td>
</tr>
<tr>
<td>AXI4_AXI_S</td>
<td>AXI4 slave interface.</td>
<td>Transaction Slave</td>
</tr>
<tr>
<td>axim_err</td>
<td>Indicates a bus error received by the AXI4 master port.</td>
<td>Signal Master</td>
</tr>
<tr>
<td>clk</td>
<td>Global clock signal for AXI and other interfaces.</td>
<td>Clock Slave</td>
</tr>
<tr>
<td>clk-in</td>
<td>Clock slave port.</td>
<td>Clock Slave</td>
</tr>
<tr>
<td>cpu_active_n</td>
<td>Controls whether GIC-500 considers a core as its first choice for an SPI that targets multiple cores.</td>
<td>Signal Slave</td>
</tr>
<tr>
<td>ecc_fatal</td>
<td>Indicates an uncorrectable ECC error.</td>
<td>Signal Master</td>
</tr>
<tr>
<td>mbistack</td>
<td>MBIST Mode Ready.</td>
<td>Signal Master</td>
</tr>
<tr>
<td>mbistoutdata</td>
<td>MBIST data out.</td>
<td>Signal Master</td>
</tr>
<tr>
<td>ppi[16-31]_n</td>
<td>PPI input.</td>
<td>Signal Slave</td>
</tr>
<tr>
<td>resetn</td>
<td>Common reset.</td>
<td>Signal Slave</td>
</tr>
<tr>
<td>spi</td>
<td>SPI inputs.</td>
<td>Signal Slave</td>
</tr>
<tr>
<td>wake_request_n</td>
<td>Indicates that a directed interrupt is pending for a core that has set GICR_WAKER.ProcessorSleep.</td>
<td>Signal Master</td>
</tr>
</tbody>
</table>
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.

   The list of available parameters will be slightly different depending on the settings that you enabled in the configuration.

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 that are available for the GIC-500 are described in Table 1-3. All pins that are not listed in this table have been either tied or disconnected for performance reasons.

**Table 1-3 GIC-500 Component Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default Value</th>
<th>Runtime¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Align Waveforms</td>
<td>When set to <strong>true</strong>, 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 <strong>false</strong>, the reset sequence is dumped to the waveform data, however, the component time is not aligned with the SoC Designer time.</td>
<td>true, false</td>
<td>true</td>
<td>No</td>
</tr>
<tr>
<td>AXI4_AXI_M Enable Debug Messages</td>
<td>Enable or disable AXI4 port debug.</td>
<td>true, false</td>
<td>false</td>
<td>No</td>
</tr>
<tr>
<td>AXI4_AXI_M Protocol Variant</td>
<td>Variant of the AXI4 protocol to use.</td>
<td>AXI4</td>
<td>AXI4</td>
<td>No</td>
</tr>
<tr>
<td>AXI4_AXI_S axi_size[0-5]</td>
<td>These parameters are obsolete and should be left at their default values.²</td>
<td>See default values</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>AXI4_AXI_S axi_start[0-5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AXI4_AXI_S Enable Debug Messages</td>
<td>Enable AXI4 Slave port debug.</td>
<td>true, false</td>
<td>false</td>
<td>No</td>
</tr>
<tr>
<td>AXI4_AXI_S Protocol Variant</td>
<td>Variant of the AXI4 protocol to use on the AXI4 slave port.</td>
<td>AXI4</td>
<td>AXI4</td>
<td>No</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
<td>Allowed Values</td>
<td>Default Value</td>
<td>Runtime</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>----------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>AXI4Stream_AXI4_Stream0n_M</td>
<td>Enable AXI4Stream master port debug message.</td>
<td>true, false</td>
<td>false</td>
<td>Yes</td>
</tr>
<tr>
<td>AXI4Stream_AXI4_Stream0n_S</td>
<td>Enable AXI4Stream slave port debug message.</td>
<td>true, false</td>
<td>false</td>
<td>Yes</td>
</tr>
<tr>
<td>Carbon DB Path</td>
<td>Sets the directory path to the database file.</td>
<td>not used</td>
<td>empty</td>
<td>No</td>
</tr>
<tr>
<td>cpu_active_n</td>
<td>Controls whether the GIC-500 considers a core as its first choice for an SPI that targets multiple cores.</td>
<td>0 to ((2^{(x+1)} – 1)^4)</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>Dump Waveforms</td>
<td>Whether SoC Designer dumps waveforms for this component.</td>
<td>true, false</td>
<td>false</td>
<td>Yes</td>
</tr>
<tr>
<td>Enable Debug Messages</td>
<td>When set to true writes the debug messages to the SoC Designer output window.</td>
<td>true, false</td>
<td>false</td>
<td>Yes</td>
</tr>
<tr>
<td>ppi[16-31]_n</td>
<td>ppi input.</td>
<td>0 to ((2^{(x+1)} – 1))</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>spi</td>
<td>spi input.</td>
<td>configuration-dependent</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>Waveform File</td>
<td>Name of the waveform file.</td>
<td>string</td>
<td>arm_cm_GIC-500.vcd</td>
<td>No</td>
</tr>
<tr>
<td>Waveform Format</td>
<td>The format of the waveform dump file.</td>
<td>VCD, FSDB</td>
<td>VCD</td>
<td>No</td>
</tr>
<tr>
<td>Waveform Timescale</td>
<td>Sets the timescale to be used in the waveform.</td>
<td>Set of values in pulldown menu.</td>
<td>1 ns</td>
<td>No</td>
</tr>
</tbody>
</table>

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. \(n\) denotes the cluster number.
4. \(x\) denotes the number of the last core in the cluster.
5. 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

This release of the GIC-500 Cycle Model does not support register views.

1.6 Available Profiling Data

The GIC-500 model component has no profiling capabilities.