CoreLink GIC-400 Cycle Model

Version 9.0.0

User Guide

Non-Confidential

ARM®

ARM DUI 1038A (ID102416)

CoreLink GIC-400 Cycle Model
User Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
November 2016	D	Non-Confidential	Restamp Release 9.0.0

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 Plus.

About This Guide

This guide provides all the information needed to configure and use the GIC-400 Cycle Model in SoC Designer Plus.

Audience

This guide is intended for experienced hardware and software developers who create components for use with SoC Designer Plus. You should be familiar with the following products and technology:

- SoC Designer Plus
- 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

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

- *SoC Designer Plus Installation Guide*
- *SoC Designer Plus User Guide*

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

- *ARM CoreLink™ GIC-400 Generic Interrupt Controller Technical Reference Manual*
- *ARM Generic Interrupt Controller Architectural Specification, Architecture Version 2.0*

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 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 Plus, 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 Plus

This chapter describes the functionality of the Cycle Model component, and how to use it in SoC Designer Plus. It contains the following sections:

1.1 GIC-400 Model Functionality

For details about the functionality of the GIC-400 hardware that the Cycle Model represents, refer to the *ARM CoreLink™ GIC-400 Generic Interrupt Controller Technical Reference Manual.*

You can configure the GIC to provide the optimum features, performance, and gate count required for your intended application. The main product configurations are:

- 1 to 8 CPU interfaces
- 0 to 480 SPIs (in multiples of 32)
- Width of the AXI ID signals for reads (RID)
- Width of the AXI ID signals for writes (WID)

1.1.1 Implemented Hardware Features

The GIC-400 Cycle Model supports the following features of the GIC-400 hardware:

- GIC Security Extensions.
- GIC Virtualization Extensions.
- The following interrupt types:
  - 16 Software Generated Interrupts (SGIs)
  - 6 external Private Peripheral Interrupts (PPIs) for each processor
  - 1 internal PPI for each processor
  - A configurable number of Shared Peripheral Interrupt (SPIs).
- Assertion of signals to indicate pending physical and virtual interrupts (refer to Table 1-2 on page 16 for details).

For more information about the GIC Security Extensions and GIC Virtualization Extensions, refer to the *ARM Generic Interrupt Controller Architecture Specification.*

1.1.2 Unsupported Hardware Features

The following features of the GIC-400 hardware are not implemented in the Cycle Model:

- Register write accesses (CadiRegWrite accesses) using the SoC Designer Plus Register View of the GIC-400 component.
- Memory view of GIC-400 registers via connected processors is not supported. If you open the memory view of the processor component and select an address that is served by the GIC-400 in that system, the data is shown as zeros.

1.1.3 Features Additional to the Hardware

The following features that are implemented in the GIC-400 Cycle Model do not exist in the GIC-400 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 17).
- Waveform dumping using the waveform-related parameters described in Table 1-3 on page 17.

1.2 Adding and Configuring the Component

The following topics briefly describe how to use the component. See the *SoC Designer Plus User Guide* for more information.

- SoC Designer Plus Component Files
- Adding the Cycle Model to the Component Library
- Adding the Component to the SoC Designer Canvas

1.2.1 SoC Designer Plus Component Files

The component files are the final output from the Model Studio compile and are the input to SoC Designer Plus. 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 Plus. 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 Plus Component Files

Platform	File	Description
Linux	maxlib.lib<*model_name*>.conf	SoC Designer Plus configuration file
	lib<*component_name*>.mx.so	SoC Designer Plus component runtime file
	lib<*component_name*>.mx_DBG.so	SoC Designer Plus component debug file
Windows	maxlib.lib<*model_name*>.windows.conf	SoC Designer Plus configuration file
	lib<*component_name*>.mx.dll	SoC Designer Plus component runtime file
	lib<*component_name*>.mx_DBG.dll	SoC Designer Plus component debug file

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 Plus *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 16.

1.3 Available Component ESL Ports

Table 1-2 describes the GIC-400 ESL ports that are exposed in SoC Designer Plus. See the *ARM CoreLink™ GIC-400 Generic Interrupt Controller Technical Reference Manual* for more information.

Table 1-2 GIC-400 ESL Component Ports

ESL Port	Description	Direction	Type
AXI4	32-bit AXI4 slave for configuring GIC-400.	Input	Transactor slave
IRQS	Interrupt source inputs (SPIs)	Input	Interrupt slave
CNTHPIRQ	PPI with interrupt ID 26 Hypervisor Timer Event	Input	Signal slave
CNTPNSIRQ	PPI with interrupt ID 30 Non-secure Physical Timer Event	Input	Signal slave
CNTPSIRQ	PPI with interrupt ID 29 Secure Physical Timer Event	Input	Signal slave
CNTVIRQ	PPI with interrupt ID 27 Virtual Timer Event	Input	Signal slave
LEGACYFIQ	Legacy FIQ interrupt for CPU interface <*n*>.	Input	Interrupt slave
LEGACYIRQ	Legacy IRQ interrupt for CPU interface <*n*>.	Input	Interrupt slave
FIQCPU_<*n*>[a]	FIQ interrupt for CPU interface <*n*>.	Output	Signal master
FIQOUT_<*n*>[1]	FIQ wakeup output for CPU interface <*n*>.	Output	Signal master
IRQCPU_<*n*>[1]	IRQ interrupt for CPU interface <*n*>.	Output	Signal master
IRQOUT_<*n*>[1]	IRQ wakeup output for CPU interface <*n*>.	Output	Signal master
VFIQCPU_<*n*>[1]	Virtual FIQ to processors.	Output	Signal master
VIRQCPU_<*n*>[1]	Virtual IRQ to processors.	Output	Signal master
clk-in	Clock slave port.	Input	Clock slave
CFGDISABLE	Prevents modification of certain secure registers, including bits that correspond to the Lockable SPIs. Typically deasserted from reset until Secure software has configured the GIC-400 and subsequently asserted permanently to provide extra security.	Input	Signal slave

a. <*n*> represents the number of CPU interfaces, from 0 to7.

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-400 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-400 Component Parameters

Parameter Name	Description	Allowed Values	Default Value	Runtime[a]
Align Waveforms	When set to *true*, waveforms dumped from the component are aligned with the SoC Designer Plus 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 Plus time.	true, false	true	No
AXI4 axi_size[0-5] AXI4 axi_start[0-5]	These parameters are obsolete and should be left at their default values.[b]	See default values	**size0** default is 0x100000000, **size 1-5** default is 0 **start**: 0x00000000	No
AXI Enable Change USER[c]	On/Off switch for the AXI transaction USER field setting function.	true, false	true	Yes
AXI4 Enable Debug Messages	Enable AXI4 port debug.	true, false	false	No
AXI4 Protocol Variant	Variant of the AXI4 protocol to use.	AXI4	AXI4	No

Table 1-3 GIC-400 Component Parameters (continued)

Parameter Name	Description	Allowed Values	Default Value	Runtime[a]
AXI USER Gen Rule [3]	User-defined rule to set the AXI transaction USER filed. To apply this setting, set the AXI Enable Change USER parameter to True.	string	Default rule; see Section 1.4.1, Using the USER Gen Rule.	Yes
Carbon DB Path	Sets the directory path to the database file.	not used	empty	No
Dump Waveforms	Whether SoC Designer Plus dumps waveforms for this component.	true, false	false	Yes
Enable Debug Messages	When set to *true* writes the debug messages to the SoC Designer Plus output window.	true, false	false	Yes
interface*X*.cluster_id	*X* = 1 to number of CPUs supported by the GIC-400. These parameters are used exclusively for generating the GIC-400 Fast Model; values are listed for your information. Do not change them.	n/a	n/a	n/a
interface*X*.core_id				
interface*X*.inout_port_number_to_use				
negLogic	Enables active low interrupts.	bool	false	Yes
Waveform File [d]	Name of the waveform file.	string	`arm_cm_GIC-400.vcd`	No
Waveform Format	The format of the waveform dump file.	VCD, FSDB	VCD	No
Waveform Timescale	Sets the timescale to be used in the waveform.	Set of values in pull-down menu.	1 ns	No

a. *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.
b. ARM recommends using the Memory Map Editor (MME) in SoC Designer Plus, 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 Plus User Guide*.
c. See Section 1.4.1, Using the USER Gen Rule for more information.
d. When enabled, SoC Designer Plus 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 USER Gen Rule

The USER Gen Rule is a string that defines the rule to manipulate the USER field of the transaction.

A valid rule looks like this:

```
Cluster_ID, Core_ID[4:3], CCI_Port[2:0]; Cluster_ID:0,
Core_ID:0, CCI_Port:3 = 0; Cluster_ID:1, Core_ID:0,
CCI_Port:4 = 1
```

and it can be divided into two parts. The first part:

```
Cluster_ID, Core_ID[4:3], CCI_Port[2:0];
```

is a list of contributing factors followed by the range of bits in the ID field where the value of that factor is stored. In the example above:

- `Core_ID[4:3]` means the value of Core_ID is stored in bit 4 to bit 3 in the ID field.
- `Cluster_ID` is an exception; its value is not in the ID field, so it is not followed by any bit range. `Cluster_ID` is included because it will be needed to generate Fast Model parameters. `Cluster_ID` does not affect the USER field.

The second part — the remainder of the rule — defines the configurations of each interface (how the ID fields are mapped to the USER fields).

`Cluster_ID:0, Core_ID:0, CCI_Port:3 = 0;` means Cluster 0 core 0 is using the S3 port of the CCI, and user interface 0 of the GIC.

The default value for this parameter is:

```
Cluster_ID, Core_ID[4:3], CCI_Port[2:0]; Cluster_ID:0,
Core_ID:0, CCI_Port:3 = 0; Cluster_ID:0, Core_ID:1,
CCI_Port:3 = 1; Cluster_ID:0, Core_ID:2, CCI_Port:3 = 2;
Cluster_ID:0, Core_ID:3, CCI_Port:3 = 3; Cluster_ID:1,
Core_ID:0, CCI_Port:4 = 4; Cluster_ID:1, Core_ID:1,
CCI_Port:4 = 5; Cluster_ID:1, Core_ID:2, CCI_Port:4 = 6;
Cluster_ID:1, Core_ID:3, CCI_Port:4 = 7
```

This is for a system containing two CPU clusters (0, 1), each cluster having four cores. Cluster 0 is connected to the S3 port of the CCI, and Cluster 1 is connected to the S4 port of CCI. Also, by default:

- Cluster 0 core 0 uses GIC interface 0
- Cluster 0 core 1 uses GIC interface 1
- Cluster 0 core 2 uses GIC interface 2
- Cluster 0 core 3 uses GIC interface 3
- Cluster 1 core 0 uses GIC interface 4
- Cluster 1 core 1 uses GIC interface 5
- Cluster 1 core 2 uses GIC interface 6

- Cluster 1 core 3 uses GIC interface 7

This format can support an unlimited number of factors which affect the USER field. For example:

```
AA[11:8], BB[5:3], Cluster_ID, Core_ID[3:2], CC[1:0]; AA:10,
BB:4, Cluster_ID:0, Core_ID:0, CC:3 = 5
```

1.5 Debug Features

The GIC-400 Cycle Model has a debug interface (CADI) that allows you to view, manipulate, and control the registers and memory. You can access a view in SoC Designer Plus by right clicking on the Cycle Model and choosing the appropriate menu entry.

1.5.1 Register Information

The GIC-400 Cycle Model has many sets of registers that are accessible via the debug interface. Registers are grouped into sets according to functional area.

See the *ARM CoreLink GIC-400 Generic Interrupt Controller Technical Reference Manual* for detailed descriptions of these registers.

Note: Displayed register values are accurate only at debuggable points. While SoC Designer Plus grays out the register view when the processor is not at a debuggable point, values are still visible. Due to the speculative nature of the processor pipeline, these values are not guaranteed to be accurate when they are displayed as gray.

In general, you can write to a register only at a debuggable point. If a value is deposited at any other point, it may not be correctly propagated.

To find information about the read/write capabilities of registers, load the component in SoC Designer Plus and check the specific register. In general, any register defined as read/write in the *ARM CoreLink GIC-400 Generic Interrupt Controller Technical Reference Manual* is also read/write in the Cycle Model. However, some registers defined as read-only in the *CoreLink GIC-400 Generic Interrupt Controller TRM* are read/write in the Cycle Model in order to support Swap & Play.

The following sections describe registers that differ from those documented in the *ARM CoreLink GIC-400 Generic Interrupt Controller TRM*. These differences include registers that:

- Are unsupported (see Unsupported Registers).
- Appear as two separate registers (see Registers in which Secure/Non-Secure Access is Banked).

Registers not specifically listed in this section are supported, and you can find information about them in the *CoreLink GIC-400 Generic Interrupt Controller TRM*.

1.5.1.1 Unsupported Registers

Table 1-4 lists registers that are not supported by the the GIC-400 Cycle Model. These registers that are *not* available to be read/written via debug transactions — for example, in the SoC Designer Plus Registers window, or by accessing them directly from your debugger. The functionality of these registers, however, does exist and can be accessed by software running on the virtual platform.

Table 1-4 Unsupported Registers

Register Name	Notes
GICC_APR1	Although these registers are not listed as supported in the *ARM® CoreLink™ GIC-400 Generic Interrupt Controller Technical Reference Manual*, they are listed as *supported* or *optional* in the *ARM Generic Interrupt Controller Architectural Specification, Architecture Version 2.0*.
GICC_APR2	
GICC_APR3	
GICH_EISR1	
GICH_ELSR1	
GICD_NSACR*n*	
GICC_EOIR	The names of these registers are viewable in the SoC Designer Plus Register window or via debugger, but they are not writeable, and their true values are not shown. Their values are displayed as zero.
GICC_AEOIR	
GICC_DIR	
GICV_EOIR	
GICV_AEOIR	
GICV_DIR	

1.5.1.2 Registers in which Secure/Non-Secure Access is Banked

For the following registers, Secure/Non-Secure access is banked, so the Cycle Model implements them as two separate registers.

Table 1-5 Banked Registers

Name	Shows As...
GICD_CTLR	Secure: GICD_CTLR_S Non-secure: GICD_CTLR_NS
GICD_ICFGR	Secure: GICD_ICFGR_S Non-secure: GICD_ICFGR_NS
GICC_RPR	Secure: GICC_RPR_S Non-secure: GICC_RPR_NS

1.6 Available Profiling Data

The GIC-400 Cycle Model component has no profiling capabilities.