PrimeCell® Generic Interrupt Controller (PL390) Cycle Model

Version 9.1.0

User Guide

Non-Confidential

ARM®

ARM DUI 1077A (ID120816)

PrimeCell Generic Interrupt Controller (PL390) 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

Confidentiality Status

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Contents

Chapter 1.
Using the Cycle Model in SoC Designer

PL390 Generic Interrupt Controller Cycle Model Functionality 1
Fully Functional and Accurate Features 2
Features Additional to the Hardware 2
Unsupported Hardware Features 2
Adding and Configuring the SoC Designer Component 3
SoC Designer Component Files 3
Adding the Cycle Model to the Component Library 4
Adding the Component to the SoC Designer Canvas 4
Available Component ESL Ports 5
Setting Component Parameters 7
Debug Features 11
Register Information 11
Available Profiling Data 14

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:

- PL390 Generic Interrupt Controller Cycle Model Functionality
- Adding and Configuring the SoC Designer Component
- Available Component ESL Ports
- Setting Component Parameters
- Debug Features
- Available Profiling Data

1.1 PL390 Generic Interrupt Controller Cycle Model Functionality

The GIC is an Advanced Microcontroller Bus Architecture (AMBA) and ARM Architecture compliant System-on-Chip (SoC) peripheral. It is a high-performance, area-optimized interrupt controller with on-chip AMBA bus interfaces that, depending on the configuration, conform to the AMBA Advanced eXtensible Interface (AXI) protocol or the AMBA AHB-Lite protocol.

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

- *GIC with AHB-Lite interfaces*. This configuration implements the AHBv2-Lite protocol and contains a single CPU Interface.
- *GIC with AXI interfaces*. This configuration implements the AXIv2 protocol, supports the Security Extensions, and contains up to eight CPU Interfaces.

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 PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual*.

- Fully Functional and Accurate Features
- Features Additional to the Hardware
- Unsupported Hardware Features

1.1.1 Fully Functional and Accurate Features

The following features of the PL390 Generic Interrupt Controller hardware are fully implemented in the PL390 Generic Interrupt Controller Cycle Model:

- Support for three interrupt types:
  - Software Generated Interrupt (SGI)
  - Private Peripheral Interrupt (PPI)
  - Shared Peripheral Interrupt (SPI)
- Programmable interrupts that enable you to set the:
  - security state for an interrupt
  - priority level of an interrupt
  - enabling or disabling of an interrupt
  - processors that receive an interrupt
- Enhanced security features, when the GIC is configured to support the Security Extensions.

1.1.2 Features Additional to the Hardware

The following features that are implemented in the GIC PL390 Cycle Model do not exist in the PL390 hardware. These features have been added to the Cycle Model for enhanced usability.

- The component supports positive and negative level *irq* and *fiq* signal. This is configurable using the *negLogic* component parameter (see the tables in the section "Setting Component Parameters" on page 1-7). By default, all the *irq*/*fiq* inputs and outputs are active-HIGH.
- The *match_x* and *enable_x* component parameters have been added to set the values of the corresponding ports: where *x* is either *c* (CPU interface) or *d* (Distributor).

1.1.3 Unsupported Hardware Features

The following hardware features have not been implemented in the Cycle Model:

- The following registers are not available:
  - Distributor Configuration registers: *ppi_enable_if (clear)*, *spi_enable (clear)*, *spi_pending (clear)*, *sgi_pending_if (clear)*, *ppi_pending_if (clear)*, *spi_config*, and *sgi_control*. For the registers with *(clear)*, use the corresponding versions of those registers (for example, *sgi_pending_if*) to set ***and*** to clear the register.

- The following registers are not available to be read / written via debug transactions — for example, in the SoC Designer Registers window, or by accessing them directly from RealView Debugger:
  - Control registers: *int_ack*, *eoi*, *run_priority*, and *hi_pending*.
  - The Integration Test registers.

  The functionality of these registers, however, does exist and can be accessed by software running on the virtual platform.
- AXI and AHB-Lite debug read/write to registers is not supported.

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.

1.3 Available Component ESL Ports

Table 1-2 describes the PL390 ESL ports that are exposed in SoC Designer when you have defined an AHB-Lite component. See the *ARM PrimeCell® Generic Interrupt Controller (PL390) Technical Reference Manual* for more information.

Each AHB-Lite or AXI signal uses a suffix to identify the interface; either **d** for Distributor or **c** for CPU interface.

Table 1-2 AHB-Lite ESL Component Ports

ESL Port	Description	Direction	Type
ahb_c	AHB-Lite slave for the CPU interface.	Input	Transaction slave
ahb_d	AHB-Lite slave for the distributor.	Input	Transaction slave
greset	Reset for the GIC. This signal is active HIGH by default.	Input	Signal slave
legacy_irq_c0	Legacy IRQ interrupt for CPU interface. Available only if legacy has been configured. Active high/low is controlled by the `negLogic` parameter. This signal is active HIGH by default.	Input	Signal slave
spi	Shared peripheral interrupt inputs.	Input	Signal slave
clk-in	Clock slave port.	Input	Clock slave
irq_c0	IRQ interrupt for the processor that connects to the CPU interface. Active high/low is controlled by the `negLogic` parameter. This signal is active HIGH by default.	Output	Signal master

Table 1-3 describes the PL390 ESL ports that are exposed in SoC Designer when you have defined an AXI component. This configuration supports security extensions and can contain up to 8 CPU interfaces. Some other ports appear only when the GIC has been configured with certain functionality in AMBA Designer.

Table 1-3 AXI ESL Component Ports

ESL Port	Description	Direction	Type
axi_c	AXI slave for the CPU interface.	Input	Transaction slave
axi_d	AXI slave for the distributor.	Input	Transaction slave
enable_c<*n*> [1]	Provides a mask select to access registers in a CPU interface. Available only if more than one processor has been configured.	Input	Signal slave
enable_d<*n*> [1]	Provides a mask select to access banked registers in the Distributor. Available only if more than one processor has been configured.	Input	Signal slave
greset	Reset for the GIC. This signal is active HIGH by default.	Input	Signal slave

Table 1-3 AXI ESL Component Ports (continued)

ESL Port	Description	Direction	Type
legacy_fiq_c<*n*>[1]	Legacy FIQ interrupt for CPU interface <n>. Available only if legacy and Security Extensions have been configured. Active high/low is controlled by the `negLogic` parameter. This signal is active HIGH by default.	Input	Signal slave
legacy_irq_c<*n*>[1]	Legacy IRQ interrupt for CPU interface <n>. Available only if legacy has been configured. Active high/low is controlled by the `negLogic` parameter. This signal is active HIGH by default.	Input	Signal slave
match_c<*n*>[1]	The result from the *enable_c* mask select is compared with this signal. Available only if more than one processor has been configured.	Input	Signal slave
match_d<*n*>[1]	The result from the *enable_d* mask select is compared with this signal. Available only if more than one processor has been configured.	Input	Signal slave
ppi_c<*n*>[1]	Private peripheral interrupt (PPI) inputs. Available only if more than one processor and more than one PPI have been configured.	Input	Signal slave
spi	Shared peripheral interrupt inputs.	Input	Signal slave
clk-in	Clock slave port.	Input	Clock slave
fiq_c<*n*>[1]	FIQ interrupt for the processor that connects to CPU interface <n>. Available only if Security Extensions have been configured. Active high/low is controlled by the `negLogic` parameter. This signal is active HIGH by default.	Output	Signal master
irq_c<*n*>[1]	IRQ interrupt for the processor that connects to the CPU interface <n>. Active high/low is controlled by the `negLogic` parameter. This signal is active HIGH by default.	Output	Signal master

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

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 enable_c, enable_d, match_c, and match_d 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 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 that are available for an AHB-Lite version of the PL390 are described in Table 1-4. The parameters that are available for an AXI version are described in Table 1-5.

Table 1-4 PL390 AHB-Lite Component Parameters

Parameter Name	Description	Allowed Values	Default Value	Runtime[1]
ahb_c Align Data	Whether halfword and byte transactions will align data to the transaction size for the ahb_c port. By default, data is not aligned.	true, false	false	No
ahb_c Big Endian	Whether AHB data is treated as big endian for the ahb_c port. By default, data is not sent as big endian.	true, false	false	No
ahb_c Enable Debug Messages	When set to *true*, writes ahb_c debug messages onto the SoC Designer output window.	true, false	false	Yes
ahb_c Filter HREADYIN	Whether the HREADYIN signal is filtered to prevent it from reaching the Cycle Model.	true, false	false	No
ahb_c region size 0	Region size of the ahb_c interface.	0 - 0xFFFFFFFF	0x100000000	No
ahb_c region size [1-5]	Unused	0 - 0xFFFFFFFF	0x0	No
ahb_c region start 0	Base address of the ahb_c interface.	0x0 - 0xffffffff	0x0	No
ahb_c region start [1-5]	Unused	0x0 - 0xffffffff	0x0	No
ahb_c Subtract Base Address	Whether the Base Address parameter is subtracted from the actual transaction address before being passed to the component. By default, the transaction address is passed directly to the component.	true, false	false	No
ahb_c Subtract Base Address Dbg	Same description as for *ahb_c Subtract Base Address*, except this is for debug transactions.	true, false	true	No

Table 1-4 PL390 AHB-Lite Component Parameters (continued)

Parameter Name	Description	Allowed Values	Default Value	Runtime[1]
ahb_d Align Data	Whether halfword and byte transactions will align data to the transaction size for the ahb_d port. By default, data is not aligned.	true, false	false	No
ahb_d Big Endian	Whether AHB data is treated as big endian for the ahb_d port. By default, data is not sent as big endian.	true, false	false	No
ahb_d Enable Debug Messages	When set to *true*, writes ahb_d debug messages onto the SoC Designer output window.	true, false	false	Yes
ahb_d Filter HREADYIN	Whether the HREADYIN signal is filtered to prevent it from reaching the Cycle Model.	true, false	false	No
ahb_d region size 0	Region size of the ahb_d interface.	0 - 0xFFFFFFFF	0x100000000	No
ahb_d region size [1-5]	Unused	0 - 0xFFFFFFFF	0x0	No
ahb_d region start 0	Base address of the ahb_d interface.	0x0 - 0xffffffff	0x0	No
ahb_d region start [1-5]	Unused	0x0 - 0xffffffff	0x0	No
ahb_d Subtract Base Address	Whether the Base Address parameter is subtracted from the actual transaction address before being passed to the component. By default, the transaction address is passed directly to the component.	true, false	false	No
ahb_d Subtract Base Address Dbg	Same description as for *ahb_d Subtract Base Address*, except this is for debug transactions.	true, false	true	No
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

Table 1-4 PL390 AHB-Lite Component Parameters (continued)

Parameter Name	Description	Allowed Values	Default Value	Runtime[1]
Enable Debug Messages	When set to *true* writes the debug messages onto the SoC Designer output window.	true, false	false	Yes
negLogic	Sets IRQ/FIQ assertion to use negative logic. Default of *false* means 0=off and 1=on. *True* means 0=on and 1=off.	true, false	false	Yes
Waveform File [2]	Name of the waveform file.	*string*	arm_cm_PL390 *<component_name>*.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 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.

The parameters that are available for an AXI version of the PL390 are described in Table 1-5.

Table 1-5 PL390 AXI Component Parameters

Parameter 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
axi_c axi_size[0-5] axi_c axi_start[0-5]	These parameters are obsolete and should be left at their default values.[2]	n/a	size0 default is 0x100000000, size1-5 default is 0x0 0x00000000	No No
axi_c Enable Debug Messages	When set to *true*, writes axi_c debug messages onto the SoC Designer output window.	true, false	false	Yes

Table 1-5 PL390 AXI Component Parameters (continued)

Parameter Name	Description	Allowed Values	Default Value	Runtime[1]
axi_d axi_size[0-5] axi_d axi_start[0-5]	These parameters are obsolete and should be left at their default values.	n/a	size0 default is 0x100000000, size1-5 default is 0x0 0x00000000	No No
axi_d Enable Debug Messages	When set to *true*, writes axi_d debug messages onto the SoC Designer output window.	true, false	false	Yes
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_c<*n*> [3]	Sets the *enable_c* value.	0x0 - 0xffffffff	0x0	Yes
enable_d<*n*> [3]	Sets the *enable_d* value.	0x0 - 0xffffffff	0x0	Yes
Enable Debug Messages	When set to *true* writes the debug messages onto the SoC Designer output window.	true, false	false	Yes
match_c<*n*> [3]	Sets the *match_c* value.	0x0 - 0xffffffff	0x0	Yes
match_d<*n*> [3]	Sets the *match_c* value.	0x0 - 0xffffffff	0x0	Yes
negLogic	Sets IRQ/FIQ assertion to use negative logic. Default of *false* means 0=off and 1=on. *True* means 0=on and 1=off.	true, false	false	Yes
Waveform File [4]	Name of the waveform file.	*string*	arm_cm_PL390<*component_name*>.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 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> represents the number of CPU interfaces, from 0 to7.
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.

1.5 Debug Features

The PL390 Generic Interrupt Controller Cycle Model has a debug interface (CADI) that allows the user to view, manipulate and control the registers in the SoC Designer Simulator or any debugger that supports the CADI, for example, Model Debugger. A view can be accessed in the SoC Designer Simulator or an instance of the Model Debugger can be attached by right clicking on the Cycle Model and choosing the appropriate menu entry. The views shown in this section are for the SoC Designer Simulator.

1.5.1 Register Information

Register views are available in SoC Designer Simulator. Access sub-fields by clicking on the plus sign to the left of a register name. Registers are grouped into different sets according to functional area.

The registers are described briefly in this section. See the *ARM PrimeCell® Generic Interrupt Controller (PL390) Technical Reference Manual* and *ARM Generic Interrupt Controller Architecture Specification* for complete information.

The following Register tabs are supported:

- Distributor Configuration Registers
- INTID Configuration Registers
- Signal Status Registers
- Control Registers
- Implementor Registers
- PrimeCell Configuration Registers

Note: All possible registers are shown on the following pages. Depending on how you configured the Cycle Model in AMBA Designer, some of the registers may not be available.

1.5.1.1 Distributor Configuration Registers

Table 1-6 shows the Distributor Configuration registers. These registers are used to determine the global configuration of the Distributor and control its operating state.

Table 1-6 Distributor Configuration Registers

Name	Description	Type
enable (Secure)	Secure Interrupt Control Register (ICDICR)	read-write
enable (Non-secure)	Non-Secure Interrupt Control Register (ICDICR)	read-write
ic_type	Interrupt Controller Type Register (ICDICTR)	read-only
dist_ident	Distributor Implementor Identification Register (ICDDIIR)	read-only

1.5.1.2 INTID Configuration Registers

Table 1-7 shows the INTID Configuration registers. These registers provide the operating parameters for each INTID.

Table 1-7 INTID Configuration Registers

Name	Description	Type
sgi_security_if<n> [1]	Interrupt Security Register (ICDISR)	read-write
ppi_security_if<n> [1]	Interrupt Security Register (ICDISR)	read-write
spi_security<x> [2]	Interrupt Security Register (ICDISR)	read-write
ppi_enable_if<n> [1]	Enable Set/Clear Register (ICDISER)	read-write
spi_enable<x> [2]	Enable Set/Clear Register (ICDISER)	read-write
sgi_pending_if<n> [1]	Pending Set/Clear Register (ICDISPR)	read-only
ppi_pending_if<n> [1]	Pending Set/Clear Register (ICDISPR)	read-only
spi_pending<x> [2]	Pending Set/Clear Register (ICDISPR)	read-only
sgi_active_if<n> [1]	Active Status Register (ICDABR)	read-only
ppi_active_if<n> [1]	Active Status Register (ICDABR)	read-only
spi_active<x> [2]	Active Status Register (ICDABR)	read-only
priority_sgi_<INTID>_if<n> [1]	Priority Level Register (ICDIPR)	read-write
priority_ppi_<INTID>_if<n> [1]	Priority Level Register (ICDIPR)	read-write
priority_spi_<INTID>	Priority Level Register (ICDIPR)	read-write
targets_spi_<INTID>	Target Register (ICDIPTR)	read-write

1. <n> represents the number of CPU interfaces, from 0 to7.
2. <x> represents the number of SPIs configured through AMBA Designer.

1.5.1.3 Signal Status Registers

Table 1-8 shows the Signal Status registers. These registers return the present logic status of the *ppi_c<n>* and *spi* inputs.

Table 1-8 Signal Status Registers

Name	Description	Type
ppi_if<n> [1]	PPI Status Register	read-only
spi<x> [2]	SPI Status Register	read-only

1. <n> represents the number of CPU interfaces, from 0 to7.
2. <x> represents the number of SPIs configured through AMBA Designer.

1.5.1.4 Control Registers

Table 1-9 shows the Control registers. Use these registers to control the operating state of the CPU Interface.

Table 1-9 Control Registers

Name	Description	Type
control<n> (Secure) [1]	Secure CPU Interface Control Register (ICCICR)	read-write
control<n> (Non-secure) [1]	Non-Secure CPU Interface Control Register (ICCICR)	read-write
pri_msk_c_<n> [1]	Priority Mask Register (ICCIPMR)	read-write
bp_c<n> (Secure) [1]	Secure Binary Point Register (ICCBPR)	read-write
nsbp_c<n> (Non-secure) [1]	Non-Secure Binary Point Register (ICCBPR)	read-write
alias_nsbp_c<n> [1]	Aliased Binary Point Register (ICCABPR) for secure access	read-write

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

1.5.1.5 Implementor Registers

Table 1-10 shows the Implementor registers. These registers identify the implementor, and revision, of the CPU Interface.

Table 1-10 Implementor Registers

Name	Description	Type
cpu_if_ident	CPU Interface Implementor Identification Register (ICCIIR)	read-only

1.5.1.6 PrimeCell Configuration Registers

Table 1-11 shows the PrimeCell Configuration registers. These registers enable the identification of system components by software.

Table 1-11 PrimeCell Configuration Registers

Name	Description	Type
periph_id_0	Peripheral Identification Register 0	read-only
periph_id_1	Peripheral Identification Register 1	read-only
periph_id_2	Peripheral Identification Register 2	read-only
periph_id_3	Peripheral Identification Register 3	read-only
periph_id_4	Peripheral Identification Register 4	read-only
periph_id_5	Peripheral Identification Register 5	read-only
periph_id_6	Peripheral Identification Register 6	read-only
periph_id_7	Peripheral Identification Register 7	read-only
periph_id_8 (Distributor)	Peripheral Identification Register 8	read-only

Table 1-11 PrimeCell Configuration Registers (continued)

Name	Description	Type
periph_id_8 (CPU Interface)	Peripheral Identification Register 8	read-only
component_id_<n>	PrimeCell Identification Registers	read-only

1.6 Available Profiling Data

The PL390 Cycle Model component has no profiling capabilities.