PrimeCell® High-Performance Matrix (PL301) Cycle Model

Version 9.1.0

User Guide

Non-Confidential

ARM®

ARM DUI 1071A (ID120516)

PrimeCell® High-Performance Matrix (PL301) 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

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

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

Web Address

http://www.arm.com

Contents

Preface

A Cycle Model component is a library developed from ARM intellectual property (IP) that is generated through Cycle Model Studio™. The Cycle Model then can be used within a virtual platform tool, for example, SoC Designer.

About This Guide

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

Audience

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

- SoC Designer
- Hardware design verification
- Verilog or SystemVerilog programming language

Conventions

This guide uses the following conventions:

Convention	Description	Example
`courier`	Commands, functions, variables, routines, and code examples that are set apart from ordinary text.	`sparseMem_t SparseMemCreate-New();`
italic	New or unusual words or phrases appearing for the first time.	*Transactors* provide the entry and exit points for data ...
bold	Action that the user performs.	Click **Close** to close the dialog.
<text>	Values that you fill in, or that the system automatically supplies.	<platform>/ represents the name of various platforms.
[text]	Square brackets [ ] indicate optional text.	`$CARBON_HOME/bin/modelstudio [ <filename> ]`
[text1 | text2]	The vertical bar | indicates "OR," meaning that you can supply text1 or text 2.	`$CARBON_HOME/bin/modelstudio [<name>.symtab.db | <name>.ccfg ]`

Also note the following references:

- References to C code implicitly apply to C++ as well.
- File names ending in .cc, .cpp, or .cxx indicate a C++ source file.

Further reading

This section lists related publications. The following publications provide information that relate directly to SoC Designer:

- *SoC Designer Installation Guide*
- *SoC Designer User Guide*
- *SoC Designer Standard Component Library Reference Manual*

The following publications provide reference information about ARM® products:

- *AMBA 3 AHB-Lite Overview*
- *AMBA Specification (Rev 2.0)*
- *AMBA AHB Transaction Level Modeling Specification*
- *Architecture Reference Manual*

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

The following publications provide additional information on simulation:

- IEEE 1666™ SystemC Language Reference Manual, (IEEE Standards Association)
- SPIRIT User Guide, Revision 1.2, SPIRIT Consortium.

Glossary

AMBA	*Advanced Microcontroller Bus Architecture*. The ARM open standard on-chip bus specification that describes a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC).
AHB	*Advanced High-performance Bus*. A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol.
APB	*Advanced Peripheral Bus*. A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports.
AXI	*Advanced eXtensible Interface*. A bus protocol that is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.
Cycle Model	A software object created by the Cycle Model Studio (or *Cycle Model Compiler*) from an RTL design. The Cycle Model contains a cycle- and register-accurate model of the hardware design.
Cycle Model Studio	Graphical tool for generating, validating, and executing hardware-accurate software models. It creates a Cycle Model, and it also takes a Cycle Model as input and generates a component that can be used in SoC Designer, Platform Architect, or Accellera SystemC for simulation.
CASI	*ESL API Simulation Interface*, is based on the SystemC communication library and manages the interconnection of components and communication between components.
CADI	*ESL API Debug Interface*, enables reading and writing memory and register values and also provides the interface to external debuggers.
CAPI	*ESL API Profiling Interface*, enables collecting historical data from a component and displaying the results in various formats.
Component	Building blocks used to create simulated systems. Components are connected together with unidirectional transaction-level or signal-level connections.
ESL	*Electronic System Level*. A type of design and verification methodology that models the behavior of an entire system using a high-level language such as C or C++.
HDL	*Hardware Description Language*. A language for formal description of electronic circuits, for example, Verilog.
RTL	*Register Transfer Level*. A high-level hardware description language (HDL) for defining digital circuits.
SoC Designer	High-performance, cycle accurate simulation framework which is targeted at System-on-a-Chip hardware and software debug as well as architectural exploration.
SystemC	SystemC is a single, unified design and verification language that enables verification at the system level, independent of any detailed hardware and software implementation, as well as enabling co-verification with RTL design.
Transactor	*Transaction adaptors*. You add transactors to your component to connect your component directly to transaction level interface ports for your particular platform.

Chapter 1

Using the Cycle Model in SoC Designer

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

1.1 PL301 HPM Cycle Model Functionality

The HPM is a highly configurable auto-generated AMBA 3 bus subsystem, based around a high-performance AXI cross-bar switch known as the AXI bus matrix, and extended by AMBA infrastructure components. For information about these components, see the *ARM PrimeCell High-Performance Matrix (PL301) Technical Reference Manual*.

This combination of IP provides support for other AMBA interface protocols, including AHB-Lite and APB.

Use the AMBA Designer Graphical User Interface configuration tool to design your bus matrix. 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. The PL301 HPM has:

- configurable number of slave interfaces (SIs) and master interfaces (MIs)
- multi-layer AXI routing, suitable for high-performance applications
- sparse connection options to reduce gate count and improve security
- configurable AXI data widths
- configurable AHB-Lite data widths
- configurable address widths on AXI and AHB-Lite interfaces
- support for an AHB-Lite to AXI bridge optimized for use with memory controllers
- support for AMBA 2 APB and AMBA 3 APB at 32-bit data width
- decoded address register that you can configure for each SI
- flexible register stages to aid timing closure
- an arbitration mechanism that you can configure (in AMBA Designer) for each MI, implementing:
  - a fixed Round-Robin (RR) scheme
  - a programmable RR scheme
  - a programmable scheme that provides prioritized groups of Least Recently Granted (LRG) arbitration
- a programmable Quality of Service (QoS) scheme
- an APB interface to provide access to programming registers
- support for multiple clock domains:
  - synchronous
  - asynchronous
- configurable cyclic dependency schemes to enable a master to have outstanding transactions to more than one slave
- PrimeCell ID register to aid self-discovery in systems

- a configurable memory map
- support for TrustZone® technology
- AXI and AHB USER signal support
- auto-generated Verilog
- auto-generated RTL testbench
- 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<*component_name*>.conf	SoC Designer configuration file
	lib<*component_name*>.mx.so	SoC Designer component runtime file
	lib<*component_name*>.mx_DBG.so	SoC Designer component debug file
Windows	maxlib.lib<*component_name*>.windows.conf	SoC Designer configuration file
	lib<*component_name*>.mx.dll	SoC Designer component runtime file
	lib<*component_name*>.mx_DBG.dll	SoC Designer component debug file

Additionally, this User Guide PDF file is provided with the component.

1.2.2 Adding the Cycle Model to the Component Library

The compiled Cycle Model component is provided as a configuration file (*.conf*). To make the component available in the Component Window in SoC Designer Canvas, perform the following steps:

1. Launch SoC Designer Canvas.
2. From the *File* menu, select **Preferences**.
3. Click on **Component Library** in the list on the left.
4. Under the *Additional Component Configuration Files* window, click **Add**.
5. Browse to the location where the SoC Designer Cycle Model is located and select the component configuration file:
   - `maxlib.lib<component_name>.conf` (for Linux)
   - `maxlib.lib<component_name>.windows.conf` (for Windows)
6. Click **OK**.
7. To save the preferences permanently, click the **OK & Save** button.

The component is now available from the SoC Designer *Component Window*.

1.2.3 Adding the Component to the SoC Designer Canvas

Locate the component in the *Component Window* and drag it out to the Canvas.

This figure shows three different components in order to show a variety of available ports. The AXI component includes two interface master ports and two interface slave ports. The AHB component includes two interface master ports and one interface slave port. The APB component includes two interface master ports and one interface slave port. Your component may appear with fewer or more ports, depending on how it was configured in AMBA Designer.

Additionally, the port names shown are AMBA Designer default names. Depending on the configuration in AMBA Designer the default port names may be different. Also, you can override the default names in AMBA Designer so that the names are fully customizable.

This naming distinction determines the ESL port names and the component parameter names used in this manual.

1.3 Available Component ESL Ports

Table 1-2 describes the ESL ports of the component, created by AMBA Designer, that are exposed in SoC Designer. See the *AMBA Designer (FD001) User Guide Supplement for the PL301*. See also the *ARM PrimeCell High Performance Matrix (PL301) Integration Manual* for more information.

Many ports in the table use the naming format *[m/s]<port#> <port_name>*, where the protocol type is part of the name. For example, *s00_axi_64*, *m01_apb_32*, *PCLKENm00_apb_32*, etc.

Table 1-2 ESL Component Ports

ESL Port	Description	Direction	Type
APB_in	The APB interface created by AMBA Designer.	input	APB transaction slave
PCLK	Clock for the APB programming interface. This clock must be synchronous to CLK.	input	Clock slave
PCLKEN<*port_name*>	The Clock enable signal for the APB interface named <*port_name*>.	input	signal
PRESETn	APB reset signal. The reset signal is: – asserted asynchronously – de-asserted synchronously to the rising edge of PCLK.	input	Signal slave
PSELEN<*port_name*>	The SEL enable signal for the APB interface named <*port_name*>.	input	Signal slave
REMAP	This vector is added by AMBA Designer when a memory map is defined.	input	Signal slave
RESETn [1]	AXI reset signal. The reset signal is: – asserted asynchronously – de-asserted synchronously to the rising edge of CLK.	input	Signal slave
CLK	AXI core clock. A clock port is created for each clock input in a multiple clock domain. See the *AMBA Designer User Guide Supplement for the PL301* for information about using multiple clock domains.	input	Clock slave
CLK<*port_name*>	Additional clock signal if your configuration contains frequency conversion AXI infrastructure components, for example, a downward-synchronizing bridge.	slave/master	signal
CLKEN<*port_name*>	Additional clock enable signal if your configuration contains frequency conversion AXI infrastructure components, for example, a downward-synchronizing bridge.	slave/master	signal

Table 1-2 ESL Component Ports (continued)

ESL Port	Description	Direction	Type
RESETn<*port_name*> [1]	Additional reset signal if your configuration contains frequency conversion AXI infrastructure components, for example, a downward-synchronizing bridge.	slave/master	signal
TZPCDECPROT0 <*port_name*>	TrustZone security programming registers for APB peripherals 0-7.	input	Signal slave
TZPCDECPROT1 <*port_name*>	TrustZone security programming registers for APB peripherals 8-15.	input	Signal slave
TZPROT	Global bus for TrustZone control.	input	Signal slave
SYNCMODEREQ <*port_name*>	Synchronous bypass mode request. Only appears if the master port is defined as asynchronous (Async).	input	Signal slave
Protocol Type Slave Ports	There is a slave port created for each port defined in AMBA Designer. For example, *s00_axi_32*, *s01_axi_64*, etc.	slave	AXI, AHB_LITE, APB
clk-in	Input clock. When using the *CLK* port, the *clk-in* port should not be connected.	slave	Clock slave
Protocol Type Master Ports	There is a master port created for each port defined in AMBA Designer. For example, *m00_axi_64*, *m01_apb_32*, *m00_ahb_54*, etc.	master	AXI, AHB_LITE, APB
SYNCMODEACK <*port_name*>	Synchronous bypass mode acknowledge. Only appears if the master port is defined as asynchronous (Async).	output	Signal master

1. The RESETn signal is active low input. If it is left unconnected, then it will be driven to a 1 (the non-reset-value).

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 RESETn, PRESETn, PSELEN, PCLKEN, TZPCDECPROT0, and TZPCDECPROT1 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 Cycle Model and select **Edit Parameters...**. You can also double-click the component.
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]
APB_in Base Address	Base address	0x0 - 0xffffffff	0x0	No
APB_in Enable Debug Messages	Whether debug messages are logged for the APB_in port.	true, false	false	Yes
APB_in Size	Size	0x0 - 0x100000000	0x100000000	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
Enable Debug Messages	Whether debug messages are logged for the component.	true, false	false	Yes
<*ahb master ports*> Align Data	Whether halfword and byte transactions will align data to the transaction size for the master ports. By default, data is not aligned.	true, false	false	No
<*ahb master ports*> Big Endian	Whether AHB data is treated as big endian for the master ports. By default, data is not sent as big endian.	true, false	false	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
<*master ports*> Enable Debug Messages	Whether debug messages are logged for the master ports. There is one parameter for each master port.	true, false	false	Yes
<*master ports*> PReady Default High	The transfer is extended if PREADY is held low during an access phase. There is one parameter for each master port.	true, false	true	Yes
PCLKEN<*port_name*>	Default value for PCLKEN<*port_name*>.	0, 1	1	Yes
PRESETn	Default value for PRESET - active high or active low.	0x0, 0x1	0x1	Yes
PSELEN<*port_name*>	Default value for PSELEN<*port_name*>. Low indicates absence of an APB device in that slot. High indicates that an APB slave device exists in that slot.	0x0 - 0xFFFF	0xFFFF	Yes
RESETn	Default value for RESET - active high or active low.	0x0, 0x1	0x1	Yes
TZPCDECPROT0 <*port_name*>	Default value for TZPCDECPROT0<*port_name*>. Low indicates that all AXI access to that APB peripheral must be secure. High indicates that access can be secure or non-secure.	0x0 - 0xFF	0	Yes
TZPCDECPROT1 <*port_name*>	Default value for TZPCDECPROT1<*port_name*>. Low indicates that all AXI access to that APB peripheral must be secure. High indicates that access can be secure or non-secure.	0x0 - 0xFF	0	Yes
s0[0-1]_<*port_name*> axi_size[0-5] [2]	These parameters are obsolete and should be left at their default values.[3]	0x0 - 0x100000000	size0 default is 0x100000000, size1-5 default is 0	No
s0[0-1]_<*port_name*> axi_start[0-5] [2]		0x0 - 0xffffffff	0x00000000	No
<*ahb slave ports*> ahbm port ID	ID of the slave port.	*value*	0	No
<*ahb slave ports*> Align Data	Whether halfword and byte transactions will align data to the transaction size for the slave ports. By default, data is not aligned.	true, false	false	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
<*ahb slave ports*> Big Endian	Whether AHB data is treated as big endian for the slave ports. By default, data is not sent as big endian.	true, false	false	No
<*slave ports*> Enable Debug Messages	Whether debug messages are logged for slave ports. There is one parameter for each slave port.	true, false	false	Yes
<*ahb slave ports*> region size [0-5]	*size 0* is the size of the region occupied by this AHB slave. *size 1* - *size 5* are unused.	0x0 - 0x100000000	size 0 default is 0x100000000, size1-5 default is 0x0	No
<*ahb slave ports*> region start [0-5]	*start 0* is the base address of the memory region occupied by this AHB slave. *start 1* - *start 5* are unused.	0x0 - 0xffffffff	0x0	No
Waveform File [4]	Name of the waveform file.	*string*	arm_cm_pl301.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. The square brackets used in parameter names specify a range of numbers that are available for the Cycle Model. The parameter name for the start addresses “s0[0-1]_axi_64 axi_start[0-5]” for example will be expanded to 12 possible parameter name combinations that range from “s00_axi_64 axi_start0” to “s01_axi_64 axi_start5”. The size of a memory region depends on the “s[N]_axi_64 axi_start[M]” and “s[N]_axi_64 axi_size[M]” parameters. The end address is calculated as StartAddr +Size -1. The size of the memory region must not exceed the value of 0x100000000. If the sum of StartAddr+Size is greater than 0x100000000, the size of the memory region is reduced to the difference: 0x100000000-StartAddr.
3. 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*.
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 PL301 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 PL301 Cycle Model has a minimum of three sets of registers that are accessible via the debug interface.

- For each slave port that is defined in the Cycle Model, there is one register tab that can be used to track signals (for example, *s00_axi_64_sig*).
- For each master port defined in the Cycle Model, there are two register tabs that can be used to track signals and registers (for example, *m00_axi_64_sig* and *m00_axi_64_reg*, respectively).
- The APB_in signals are available in one register tab.

The registers are grouped into sets according to slave and master ports. The registers are listed below:

- Slave Port Signals Registers
- Master Port Quality of Service Registers
- Master Port Signals Registers
- APB_in Signal Registers

1.5.1.1 Slave Port Signals Registers

The Slave Port Signals group contains a list of slave port signals and their current values. There is one register tab for each slave port that is part of your component. Slave signals for AXI and AHB are listed in separate tables below.

Table 1-4 shows the AXI slave port signals. See the *ARM AMBA AXI Protocol Specification* for more information about these signals.

Table 1-4 AXI Slave Port Signal Registers

Name	Description	Type
ARID	The read address ID.	read-only
ARADDR	The read address.	read-only
ARVALID	Indicates whether the read address is available.	read-only
ARREADY	Indicates whether the slave is ready to accept the read address.	read-only
ARLEN	The burst length.	read-only
ARSIZE	The burst size.	read-only
ARBURST	The burst type.	read-only
ARLOCK	The lock type.	read-only
ARCACHE	The cache type.	read-only
ARPROT	The protection type.	read-only
AWID	The write address ID.	read-only
AWADDR	The write address.	read-only
AWVALID	Indicates whether the write address is available.	read-only
AWREADY	Indicates whether the slave is ready to accept the write address.	read-only
AWLEN	The burst length.	read-only
AWSIZE	The burst size.	read-only
AWBURST	The burst type.	read-only
AWLOCK	The lock type.	read-only
AWCACHE	The cache type.	read-only
AWPROT	The protection type.	read-only
WID	The write ID tag.	read-only
WDATA	The write data.	read-only
WSTRB	The write strobes.	read-only
WLAST	The last transfer in a write burst.	read-only
WVALID	Indicates whether the write data and strobes are available.	read-only
WREADY	Indicates whether the slave is ready to accept the write data.	read-only
RID	The read ID tag.	read-only
RDATA	The read data.	read-only

Table 1-4 AXI Slave Port Signal Registers (continued)

Name	Description	Type
RLAST	The last transfer in a read burst.	read-only
RVALID	Indicates whether the read data is available.	read-only
RREADY	Indicates whether the master is ready to accept the read data and response information.	read-only
RRESP	The read response.	read-only
BID	The response ID.	read-only
BRESP	The write response.	read-only
BVALID	Indicates whether the write response is available.	read-only
BREADY	Indicates whether the master is ready to accept the response information.	read-only

Table 1-5 shows the AHB slave port signals. See the *ARM AMBA AHB Protocol Specification* for more information about these signals.

Table 1-5 AHB Slave Port Signal Registers

Name	Description	Type
HADDR	The 32-bit system address bus.	read-only
HBURST	Indicates if the transfer forms part of a burst. Four, eight, and sixteen beat bursts are supported.	read-only
HMASTLOCK	Indicates that the current master is performing a locked sequence of transfers.	read-only
HPROT	The protection control signals provide additional information about a bus access, and are primarily intended for use by any module that wishes to implement some level of protection.	read-only
HRDATA	The read data bus is used to transfer data from bus slaves to the bus master during read operations.	read-only
HREADY	When HIGH, the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer.	read-only
HRESP	The transfer response provides additional information on the status of a transfer. Four responses are provided, OKAY, ERROR, RETRY, and SPLIT.	read-only
HSIZE	Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit), or word (32-bit).	read-only
HTRANS	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE, or BUSY.	read-only
HWDATA	The write data bus is used to transfer data from the master to the bus slaves during write operations.	read-only
HWRITE	When HIGH, this signal indicates a write transfer. When LOW, it indicates a read transfer.	read-only

1.5.1.2 Master Port Quality of Service Registers

The Master Port Quality of Service (QoS) group contains registers that dynamically configure the quality of service parameters. There is one register tab for each master interface (MI) port that is part of your component. The MI port must be configured with QoS functionality (`prog_qos==true`), otherwise these registers are read-only with values of 0.

See the *ARM PrimeCell High-Performance Matrix (PL301) Technical Summary* for detailed descriptions of these registers.

Table 1-6 Master Port QoS Access Control Registers

Name	Description	Type
qos_tidemark	The number of outstanding transactions at a particular MI port before the QoS scheme becomes active and accepts transactions from the SI ports. If a value of 0 is written to this register, then the QoS scheme is turned off for this MI.	read-write
qos_idmatch	A 1 in any bit of the QoS access control register indicates that the SI corresponding to the bit position is permitted to use the reserved slots of the connected combined acceptance capability of the slaves.	read-write
priority_for_AR_<*SI_port*>	Used to set the priority value (0-255) of the AR channel for each SI that is connected to the MI. This register applies only if the MI channel was configured with the Least Recently Granted (LRG) arbitration scheme. If LRG was not selected for this MI, then this register is a read-only value of zero.	read-write
priority_for_AW_<*SI_port*>	Used to set the priority value (0-255) of the AW channel for each SI that is connected to the MI. This register applies only if the MI channel was configured with the Least Recently Granted (LRG) arbitration scheme. If LRG was not selected for this MI, then this register is a read-only value of zero.	read-write

1.5.1.3 Master Port Signals Registers

The Master Port Signals group contains a list of master port signals and their current values. There is one register tab for each master port that is part of your component. Master signals for AXI, AHB, and APB are listed in separate tables below.

Table 1-7 shows the AXI master port signals. See the *ARM AMBA AXI Protocol Specification* for more information about these signals.

Table 1-7 AXI Master Port Signal Registers

Name	Description	Type
ARID	The read address ID.	read-only
ARADDR	The read address.	read-only
ARVALID	Indicates whether the read address is available.	read-only
ARREADY	Indicates whether the slave is ready to accept the read address.	read-only
ARLEN	The burst length.	read-only
ARSIZE	The burst size.	read-only

Table 1-7 AXI Master Port Signal Registers (continued)

Name	Description	Type
ARBURST	The burst type.	read-only
ARLOCK	The lock type.	read-only
ARCACHE	The cache type.	read-only
ARPROT	The protection type.	read-only
AWID	The write address ID.	read-only
AWADDR	The write address.	read-only
AWVALID	Indicates whether the write address is available.	read-only
AWREADY	Indicates whether the slave is ready to accept the write address.	read-only
AWLEN	The burst length.	read-only
AWSIZE	The burst size.	read-only
AWBURST	The burst type.	read-only
AWLOCK	The lock type.	read-only
AWCACHE	The cache type.	read-only
AWPROT	The protection type.	read-only
WID	The write ID tag.	read-only
WDATA	The write data.	read-only
WSTRB	The write strobes.	read-only
WLAST	The last transfer in a write burst.	read-only
WVALID	Indicates whether the write data and strobes are available.	read-only
WREADY	Indicates whether the slave is ready to accept the write data.	read-only
RID	The read ID tag.	read-only
RDATA	The read data.	read-only
RLAST	The last transfer in a read burst.	read-only
RVALID	Indicates whether the read data is available.	read-only
RREADY	Indicates whether the master is ready to accept the read data and response information.	read-only
RRESP	The read response.	read-only
BID	The response ID.	read-only
BRESP	The write response.	read-only
BVALID	Indicates whether the write response is available.	read-only
BREADY	Indicates whether the master is ready to accept the response information.	read-only

Table 1-8 shows the AHB master port signals. See the *ARM AMBA AHB Protocol Specification* for more information about these signals.

Table 1-8 AHB Master Port Signal Registers

Name	Description	Type
HADDR	The 32-bit system address bus.	read-only
HBURST	Indicates if the transfer forms part of a burst. Four, eight, and sixteen beat bursts are supported.	read-only
HMASTLOCK	Indicates that the current master is performing a locked sequence of transfers.	read-only
HPROT	The protection control signals provide additional information about a bus access, and are primarily intended for use by any module that wishes to implement some level of protection.	read-only
HRDATA	The read data bus is used to transfer data from bus slaves to the bus master during read operations.	read-only
HREADY	When HIGH, the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer.	read-only
HREADYOUT	When HIGH, the HREADYOUT signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer.	read-only
HRESP	The transfer response provides additional information on the status of a transfer. Four responses are provided, OKAY, ERROR, RETRY, and SPLIT.	read-only
HSEL	Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave.	read-only
HSIZE	Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit), or word (32-bit).	read-only
HTRANS	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE, or BUSY.	read-only
HWDATA	The write data bus is used to transfer data from the master to the bus slaves during write operations.	read-only
HWRITE	When HIGH, this signal indicates a write transfer. When LOW, it indicates a read transfer.	read-only

Table 1-9 shows the APB master port signals. See the *ARM AMBA 3 APB Protocol Specification* for more information about these signals.

Table 1-9 APB Master Port Signal Registers

Name	Description	Type
PRDATA	The selected slave drives the Read Data bus during read cycles when PWRITE is LOW. This bus can be up to 32-bits wide.	read-only
PADDR	The APB address bus. It can be up to 32 bits wide and is driven by the peripheral bus bridge unit.	read-only
PWDATA	The Write data bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is HIGH. This bus can be up to 32 bits wide.	read-only
PWRITE	The Direction signal indicates an APB write access when HIGH and an APB read access when LOW.	read-only
PSEL	The APB bridge unit generates the Select signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. There is a Select signal for each slave.	read-only
PENABLE	The Enable signal indicates the second and subsequent cycles of an APB transfer.	read-only
PREADY	The slave uses the Ready signal to extend an APB transfer.	read-only
PSLVERR	The PSLVERR signal indicates a transfer failure. APB peripherals are not required to support the PSLVERR pin. Where a peripheral does not include this pin, then the appropriate input to the APB Bridge is tied LOW.	read-only

1.5.1.4 APB_in Signal Registers

Table 1-10 shows the APB_in signals. See the *ARM AMBA 3 APB Protocol Specification* for more information about these signals.

Table 1-10 APB_in Signal Registers

Name	Description	Type
PRDATA	The selected slave drives the Read Data bus during read cycles when PWRITE is LOW. This bus can be up to 32-bits wide.	read-only
PADDR	The APB address bus. It can be up to 32 bits wide and is driven by the peripheral bus bridge unit.	read-only
PWDATA	The Write data bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is HIGH. This bus can be up to 32 bits wide.	read-only
PWRITE	The Direction signal indicates an APB write access when HIGH and an APB read access when LOW.	read-only
PSEL	The APB bridge unit generates the Select signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. There is a Select signal for each slave.	read-only
PENABLE	The Enable signal indicates the second and subsequent cycles of an APB transfer.	read-only
PREADY	The slave uses the Ready signal to extend an APB transfer.	read-only
PSLVERR	The PSLVERR signal indicates a transfer failure. APB peripherals are not required to support the PSLVERR pin. Where a peripheral does not include this pin, then the appropriate input to the APB Bridge is tied LOW.	read-only

1.6 Available Profiling Data

The PL301 Cycle Model component has no profiling capabilities.