PrimeCell® Vectored Interrupt Controller (PL190) Cycle Model

Version 9.1.0

User Guide

Non-Confidential

ARM®

ARM DUI 1069A (ID120516)

PrimeCell® Vectored Interrupt Controller (PL190) 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 VIC PL190 Cycle Model Functionality

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 simulates, refer to the *ARM PrimeCell® Vectored Interrupt Controller (PL190) Technical Reference Manual*.

1.1.1 Fully Functional and Accurate Features

The following features of the VIC PL190 hardware implementation are fully implemented in the VIC PL190 Cycle Model.

- Compliance to AMBA AHB specification
- Support for 32 standard interrupts
- Support 16 vectored IRQ interrupts
- Hardware interrupts priority levels
- IRQ and FIQ generation
- Software interrupts generation
- Debug Register
- Raw interrupt status
- Interrupt request status
- Interrupt masking
- Privileged mode support
- Interrupt controller Daisy chaining

1.1.2 Unsupported Hardware Features

The PL190 Cycle Model does not support scan or other testing features.

1.1.3 Features Additional to the Hardware

The following features that are implemented in the VIC PL190 Cycle Model do not exist in the VIC PL190 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* parameter (see Table 1-3 on page 1-6).

1.2 Adding and Configuring the SoC Designer Component

The following topics briefly describe how to use the Cycle Model 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 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 ESL ports that are exposed in SoC Designer. See the *ARM PrimeCell® VIC (PL190) Technical Reference Manual* for more information.

Table 1-2 ESL Component Ports

ESL Port	Description	Direction	Type
clk-in	Input clock. This component must be connected to the clock.	Input	Clock slave
reset	Input reset. Reset port for receiving reset signal.	Input	Signal slave
ahb	AHB slave port.	Input	AHB_Slave_FT2S Transaction slave
isrc	Interrupt source.	Input	Interrupt slave
fiq_in	External interrupt controller FIQ interrupt. Active high/low is controlled by the `negLogic` parameter.	Input	Signal slave
irq_in	External interrupt controller IRQ interrupt. Active high/low is controlled by the `negLogic` parameter.	Input	Signal slave
VecAddrIn	Daisy-chained vector address input.	Input	Signal slave
fiq	FIQ interrupt. Active high/low is controlled by the `negLogic` parameter.	Output	Signal master
irq	IRQ interrupt. Active high/low is controlled by the `negLogic` parameter.	Output	Signal master
VecAddrOut	Daisy-chained vector address output.	Output	Signal master

All pins that are not listed in this table have been either tied or disconnected for performance reasons.

Note: It is necessary to disable the ports if they are not used.

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]
ahb Align Data	Whether halfword and byte transactions will align data to the transaction size. By default, data is not aligned.	true, false	false	No
ahb Big Endian	Whether AHB data is treated as big endian. By default, data is not sent as big endian.	true, false	false	No
ahb Enable Debug Messages	Enable or disable the capture of ahb debug messages.	true, false	false	Yes
ahb Filter HREADYIN	The HREADYIN signal indicates the state of other AHB devices on the bus. By default, this signal is not filtered - it is received by this component.	true, false	false	No
ahb region size 0	Address range size.	0 - 0xFFFFFFFF	0x100	No
ahb region size [1-5]	Unused	0 - 0xFFFFFFFF	0x0	No
ahb region start 0	Address range base.	0x0 - 0xffffffff	0x1000000	No
ahb region start [1-5]	Unused	0x0 - 0xffffffff	0x0	No
ahb Subtract Base Address	Whether the Base Address parameter is subtracted from the actual transaction address before being passed to the component. By default, the actual transaction address is passed directly to the component.	true, false	false	No
ahb Subtract Base Address Dbg	Same description as for *ahb Subtract Base Address*, except this is for debug transactions.	true, false	true	No

Table 1-3 Component Parameters (continued)

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
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	Enable or disable the capture of debug messages.	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	No
Waveform File [2]	Name of the waveform file.	*string*	arm_cm_pl1 90.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.

1.5 Debug Features

The VIC PL190 Cycle Model has a debug interface (CADI) that allows the user to view, manipulate and control the registers in the SoC Designer Simulator and Model Debugger.

1.5.1 Registers Information

This section lists the register views available for the VIC PL190 Cycle Model in the SoC Designer Simulator. The VIC PL190 Cycle Model has three sets of registers that are accessible via the debug interface. Registers are grouped into sets according to functional area.

- General Registers
- Peripheral ID Registers
- PrimeCell ID Registers

See the *ARM PrimeCell® VIC (PL190) Technical Reference Manual* for detailed descriptions of these registers.

1.5.1.1 General Registers

Table 1-4 shows the General registers.

Table 1-4 General Registers Summary

Register	Description	Type
VICIRQStatus	Provides the status of the interrupts after IRQ masking.	read-only
VICFIQStatus	Provides the status of the interrupts after FIQ masking.	read-only
VICRawIntr	Provides the status of the source interrupts, and software interrupts, to the interrupt controller.	read-only
VICIntSelect	Selects whether the corresponding interrupt source generates an FIQ or an IRQ interrupt. 0 = IRQ interrupt 1 = FIQ interrupt	read-write
VICIntEnable	Enables the interrupt request lines, by masking the interrupt sources for the IRQ interrupt. A write to the debug interface will forcibly set the value. It will not behave as a write to this register through the AHB bus. See section 3.3.5 of the TRM.	read-write
VICIntEnClear	Set to 0	read-only
VICSoftInt	Generates software interrupts. A write to the debug interface will forcibly set the value. It will not behave as a write to this register through the AHB bus. See section 3.3.7 of the TRM.	read-write
VICSoftIntClear	Set to 0	read-only
VICProtection	Enables or disables protected register access.	read-write
VICVectAddr[0-15]	Contains the ISR vector address; from 0 to 15.	read-write

Table 1-4 General Registers Summary (continued)

Register	Description	Type
VICVectCntrl[0-15]	Selects the interrupt source for the vectored interrupt; from 0 to 15.	read-write
VICVectAddr	Contains the Interrupt Service Routine (ISR) address of the currently active interrupt.	read-only
VICDefVectAddr	Contains the address of the default ISR handler.	read-write

1.5.1.2 Peripheral ID Registers

Table 1-5 shows the Peripheral Identification registers.

Table 1-5 Peripheral ID Registers Summary

Register	Description	Type
VICPeriphID0	Identifies the part number of the peripheral.	read-only
VICPeriphID1	Identifies the part number and designer of the peripheral.	read-only
VICPeriphID2	Identifies the revision and designer of the peripheral.	read-only
VICPeriphID3	Identifies the configuration of the peripheral.	read-only

1.5.1.3 PrimeCell ID Registers

Table 1-6 shows the PrimeCell Identification registers.

Table 1-6 PrimeCell ID Registers Summary

Register	Description	Type
VICPCellID0	Determines the reset value.	read-only
VICPCellID1	Determines the reset value.	read-only
VICPCellID2	Determines the reset value.	read-only
VICPCellID3	Determines the reset value.	read-only

1.6 Available Profiling Data

The VIC PL190 Cycle Model component has no profiling capabilities.