PrimeCell® Dynamic Memory Controller (PL340) Cycle Model

Version 9.1.0

User Guide

Non-Confidential

ARM®

ARM DUI 1074A (ID120616)

PrimeCell Dynamic Memory Controller (PL340) 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 PL340 AXI Memory Controller Cycle Model Functionality

The PL340 memory controller is a high-performance, area-optimized SDRAM or Mobile SDR memory controller compatible with the Advanced Microcontroller Bus Architecture (AMBA) AXI protocol. For a detailed description of the AXI protocol refer to the *AMBA AXI Protocol Specification*.

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 Dynamic Memory Controller (PL340) Technical Reference Manual*.

The following topics are discussed in this section:

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

1.1.1 Fully Functional and Accurate Features

The following features of the PL340 AXI Memory Controller hardware are fully implemented in the PL340 AXI Memory Controller Cycle Model:

- All features of the Cycle Model are both functionally and cycle accurate with the exception of those listed in Fully Functional and Approximate Features, below.

1.1.2 Fully Functional and Approximate Features

The following features of the PL340 AXI Memory Controller hardware are implemented in the PL340 AXI Memory Controller Cycle Model, but the exact behavior of the hardware implementation is not accurately reproduced because some approximations and optimizations have been made for simulation performance:

- The self refresh logic is functionally correct, but self refreshes may occur at slightly different times in the Cycle Model and in RTL.

1.1.3 Unsupported Hardware Features

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

- EBI interface
- AXI low-power interface
- Asynchronous aclk and mclk and clock ratio between the two clocks
- ATPG signals
- DFT
- Integration test registers
  - int_outputs
  - int_inputs
  - int_cfg
- The PL340 TRM shows multiple clocks: aclk, mclk, mclkn, mclkx2, and mclkx2n. The Cycle Model only has clk-in.

1.1.4 Features Additional to the Hardware

The following features that are implemented in the PL340 AXI Memory Controller Cycle Model to enhance usability do not exist in the PL340 AXI Memory Controller hardware:

- The PL340 Memory Controller Cycle Model has built-in memory, so you do not need to provide a memory.
- Debug and profiling features. For further information about debug and profiling features, refer to "Debug Features" on page 1-9 and "Available Profiling Data" on page 1-12 respectively.

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

The PL340 component has an APB transaction slave port, an AXI transaction slave port and additional signal slave ports as shown below. The APB port is for configuring the component, whereas the AXI port is for accessing the memory. Table 1-2 describes the ESL ports that are exposed in SoC Designer. See the *ARM PrimeCell® Dynamic Memory Controller (PL340) Technical Reference Manual* for more information.

Table 1-2 ESL Component Ports

ESL Port	Description	Direction	Type
apb	APB port for memory mapped register accesses. Refer to section A.4 of the PL340 Technical Reference Manual for the APB signal list.	slave	APB transaction slave
axi	AXI port for memory accesses. Refer to section A.3 of the PL340 Technical Reference Manual for the AXI signal list.	slave	AXI transaction slave
pclken	Clock enable for APB.	slave	Signal
qos_override	Signal port to override QoS.	slave	Signal
read_delay	Delay before the capture of read data from the memory device.	slave	Signal
user_status	General purpose APB-accessible port.	slave	Signal
clk-in	Clock slave port.	slave	Clock slave
user_config	General purpose APB-accessible port.	master	Signal

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 pclken, user_status, and read_delay (when available) 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 **Component Information**. 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 are described in Table 1-3.

Table 1-3 Component Parameters

Parameter Name	Description	Allowed Values	Default Value	Runtime[1]
apb Base Address	APB Region base address. The address must be on a 4KB boundary.	0x0 - 0xFFFFFFFF	0x0	No
apb Enable Debug Messages	When set to *true* writes APB debug messages onto the SoC Designer output window.	true, false	false	Yes
apb Size	APB region size.	0x0 - 0xFFFFFFFF	0x100000000	No
axi axi_size[0-5] axi axi_start[0-5]	These parameters are obsolete and should be left at the default values.[2]	n/a	size0 default is 0x100000000, size1-5 default is 0x0 0x00000000	No No
axi Enable Debug Messages	When set to *true* writes AXI debug messages onto the SoC Designer output window.	true, false	false	Yes
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

Table 1-3 Component Parameters (continued)

Parameter Name	Description	Allowed Values	Default Value	Runtime[1]
chip_size[0-3]	Size of DDR memory regions 0 through 3.	0x1000000 0x2000000 0x4000000	initialized to the correct size of the memory regions [3]	No
chip_start[0-3]	Initial start address, in each memory region, that is used for program loading and debug access before the simulation is started.	0x0 - 0xFFFFFFFF	0x00000000	No
Dump Waveforms	Whether SoC Designer dumps waveforms for this component.	true, false	false	Yes
Enable Debug Messages	When set to *true* writes the debug messages onto the SoC Designer output window.	true, false	false	Yes
Maximum amber read latency	Max value displayed as amber for read latency profiling.	>0	30	Yes
Maximum amber write latency	Max value displayed as amber for write latency profiling.	>0	30	Yes
Maximum green read latency	Max value displayed as green for read latency profiling.	>0	20	Yes
Maximum green write latency	Max value displayed as green for write latency profiling.	>0	20	Yes
pclken	Clock enable for APB domain.	0, 1	0x1	Yes
read_delay	Delay before the capture of read data from the memory device.	0 - 3	0x1	Yes
user_status	General purpose APB-accessible input pins.	0x0 - 0xFF	0x0	Yes
Waveform File [4]	Name of the waveform file.	*string*	arm_cm_pl340_ <*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.
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. The chip_size depends on the memory_width defined in AMBA Designer.

memory_width	chip_size
16	16M (0x1000000)
32	32M (0x2000000)
64	64M (0x4000000)

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 PL340 AXI Memory Controller Cycle Model has a debug interface (CADI) that allows the user to view, manipulate and control the registers and memory 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.

- Register Information
- Memory Information

1.5.1 Register Information

Figure 1-3 shows the register view of the PL340 AXI Memory Controller Cycle Model in the SoC Designer Simulator. Sub-fields can be accessed 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® Dynamic Memory Controller (PL340) Technical Reference Manual* for complete information.

The following Register tabs are supported:

- DMC Configuration Registers
- Chip Configuration Registers
- AXI ID Configuration Registers

1.5.1.1 DMC Configuration Registers

Table 1-4 shows the DMC Configuration registers. This tab shows the registers that configure the entire memory controller, including timing and ID registers.

Table 1-4 DMC Configuration Registers

Name	Description	Type
memc_status	The Memory Controller Status register provides information on the configuration of the memory controller, and also on the current state of the memory controller.	read-only
memc_cmd	The Memory Controller Command register enables the memory controller to be traversed.	write-only
direct_cmd	The Direct Command register passes commands to the external memory.	write-only
memory_cfg	The Memory Configuration register configures the memory.	read-write
refresh_prd	The Refresh Period register sets the memory refresh period.	read-write
cas_latency	The CAS Latency register sets the CAS latency in memory clock cycles.	read-write

Table 1-4 DMC Configuration Registers (continued)

Name	Description	Type
t_dqss	The t_dqss register writes to DQS in memory clock cycles.	read-write
t_mrd	The t_mrd register sets the mode register command time in memory clock cycles.	read-write
t_ras	The t_ras register sets the RAS to precharge delay in memory clock cycles.	read-write
t_rc	The t_rc register sets the Active bank x to Active bank x delay in memory clock cycles.	read-write
t_rcd	The t_rcd register sets the RAS to CAS minimum delay in memory clock cycles.	read-write
t_rfc	The t_rfc register sets the auto-refresh command time in memory clock cycles.	read-write
t_rp	The t_rp register sets the precharge to RAS delay in memory clock cycles.	read-write
t_rrd	The t_rrd register sets the Active bank x to Active bank y delay in memory clock cycles.	read-write
t_wr	The t_wr register sets the write to precharge delay in memory clock cycles.	read-write
t_wtr	The t_wtr register sets the write to read delay in memory clock cycles.	read-write
t_xp	The t_xp register sets exit power-down command time in memory clock cycles.	read-write
t_xsr	The t_xsr register sets exit self-refresh command time in memory clock cycles.	read-write
t_esr	The t_esr register sets self-refresh command time in memory clock cycles.	read-write
memory_cfg2	The memory_cfg2 register determines the operating state of the memory controller and the memory.	read-write
memory_cfg3	The memory_cfg3 register determines the operating state of the memory controller and the memory.	read-write
user_status	This register returns the state of the user_status[7:0] primary inputs.	read-only
user_config	This register sets the value of the user_config[7:0] primary outputs.	write-only
periph_id	The periph_id registers are four 8-bit read-only registers, that span address locations 0xFE0-0xFEC. The registers can conceptually be treated as a single register that holds a 32-bit peripheral ID value.	read-only
pcell_id	The pcell_id registers are four 8-bit wide read-only registers, that span address locations 0xFF0-0xFFC. The registers can be treated conceptually as a single register that holds a 32-bit PrimeCell identification value.	read-only

1.5.1.2 Chip Configuration Registers

Table 1-5 shows the CHIP_CFG Chip Configuration registers.

Table 1-5 Chip Configuration Registers

Name	Description	Type
chip_0_cfg	These registers set up the external memory device configuration. The number of external chips supported, and therefore the number of these registers, depends on your configuration. They span address locations 0x200-0x300.	read-write
chip_1_cfg		read-write
chip_2_cfg		read-write
chip_3_cfg		read-write

*Note: Component parameter values are used for debug access before the start of simulation, so program loading will use the "chip_start**x**" parameter values. Once the simulation starts (when cycle count is greater than 0), debug access will always use the register values. Note that debug access is invalid during the CONFIG state as register values can change.*

1.5.1.3 AXI ID Configuration Registers

Table 1-6 shows the ID_CFG ID Configuration registers.

Table 1-6 ID Configuration Registers

Name	Description	Type
id_0_cfg	These registers are 16 registers that set the QoS (Quality of Service) and span address locations 0x100-0x200.	read-write
id_1_cfg		read-write
id_2_cfg		read-write
id_3_cfg		read-write
id_4_cfg		read-write
id_5_cfg		read-write
id_6_cfg		read-write
id_7_cfg		read-write
id_8_cfg		read-write
id_9_cfg		read-write
id_10_cfg		read-write
id_11_cfg		read-write
id_12_cfg		read-write
id_13_cfg		read-write
id_14_cfg		read-write
id_15_cfg		read-write

1.5.2 Memory Information

The Memory view does not display any values when the PL340 is in the CONFIG state, and it uses the CHIP_CFG register values for displaying the Memory view once the simulation starts (when cycle count is greater than 0).

1.6 Available Profiling Data

Profiling data is enabled, and can be viewed using the Profiling Manager, which is accessible via the Debug menu in the SoC Designer Simulator. The profiling events are the ones that can be monitored in the hardware using counters. The PL340 Cycle Model profiles the events shown in Table 1-7.

Table 1-7 PL340 Profiling Streams and Events

Stream	Events	X axis	Y axis
Memory Command	Read	Cycle	Memory Command
	Write		
	Precharge		
	Precharge All		
	Self Recharge		
	Self Recharge Exit		
	Auto Refresh		
	Mode Reg		
Write Latency	Green	Cycle	Write Latency
	Amber		
	Red		
Read Latency	Green	Cycle	Read Latency
	Amber		
	Red		

The Write Latency is measured from the time of the write request on the AW channel to the time of the response on the B channel. The Read Latency is measured in the same way, using the AR and R channels. The latency values are assigned to buckets based on thresholds you can set using component parameters.