PrimeCell® DMA Controller (PL080/PL081) Cycle Model

Version 9.1.0

User Guide

Non-Confidential

ARM®

ARM DUI 1068A (ID120516)

PrimeCell® DMA Controller (PL080/PL081) 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, and how to use it in SoC Designer. It contains the following sections:

1.1 PL080/PL081 DMAC Cycle Model Functionality

The ARM® PrimeCell® Dynamic Memory Access Controller (DMAC) is an AMBA compliant System-on-Chip peripheral that provides AHB bus masters for DMA transfers. The DMAC PL080 has two master ports and eight DMA channels. The DMAC PL081 has just a single master port and two DMA channels. It supports data transfer between memory to memory, memory to peripheral, and peripheral to peripheral. It is programmed and controlled by an AHB slave interface.

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 Access Controller (PL080) Technical Reference Manual* or *ARM PrimeCell Dynamic Memory Access Controller (PL081) Technical Reference Manual*.

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.

1.1.1 Fully Functional and Accurate Features

The following features of the DMAC PL080/PL081 hardware are fully implemented in the DMAC PL080 and PL081 Cycle Model:

- Compliance to the AMBA (Rev 2.0) Specification.
- Eight (PL080) or two (PL081) DMA channels. Each channel can support a unidirectional transfer.
- Single DMA and burst DMA request signals. Each peripheral connected to the DMAC can assert either a burst DMA request or a single DMA request. You set the DMA burst size by programming the DMAC.
- Memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral transfers.
- Scatter or gather DMA support through the use of linked lists.
- Hardware DMA channel priority. Each DMA channel has a specific hardware priority. DMA channel 0 has the highest priority and channel 7 has the lowest priority. If requests from two channels become active at the same time, the channel with the highest priority is serviced first.
- AHB slave DMA programming interface. You program the DMAC by writing to the DMA control registers over the AHB slave interface.
- Two (PL080) or one (PL081) AHB bus masters for transferring data. Use these interfaces to transfer data when a DMA request goes active.
- 32-bit AHB master bus width.
- Incrementing or non-incrementing addressing for source and destination.
- Setable DMA burst size. You can program the DMA burst size to transfer data more efficiently. The burst size is usually set to half the size of the FIFO in the peripheral.
- Internal four word FIFO per channel.
- Supports 8, 16, and 32-bit wide transactions.
- Separate and combined DMA error and DMA count interrupt requests. You can generate an interrupt to the processor on a DMA error or when a DMA count has reached 0 (this is usually used to indicate that a transfer has finished). There are three interrupt request signals to do this:
  - DMACINTTC signals when a transfer has completed.
  - DMACINTERR signals when an error has occurred.
  - DMACINTR combines both the DMACINTTC and DMACINTERR.

You can use the DMACINTR interrupt request in systems that have few interrupt controller request inputs. The signal is driven by calling the function:

`driveSignal (MxU32 value, MxU32 *extValue)` from input signal port of the Interrupt controller or the core.

It is expected that the parameter “value” is set to the Interrupt line number and the “extvalue” to the signal value.

- Interrupt masking. You can mask the DMA error and DMA terminal count interrupt requests.
- Raw interrupt status. You can read the DMA error and DMA count raw interrupt status prior to masking.
- Identification registers that uniquely identify the DMAC. An operating system can use these to automatically configure itself.

1.1.2 Fully Functional and Approximate Features

The following features of the DMAC PL080/PL081 hardware are implemented in the DMAC PL080 and PL081 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 AHB interface is modeled as an AHB transaction interface.

  For more information, refer to the *SoC Designer AHBv2 Protocol Bundle User Guide*, which is provided with the SoC Designer installation.

- 16 DMA requests (peripheral side requests). The DMAC provides 16 peripheral DMA request lines. The request signals *breq*, *sreq*, *lbreq*, and *lsreq* are modeled as signal slave ports. Up to 16 peripherals can be connected. The signals can be driven by calling the function:

  ```
  driveSignal (MxU32 value, MxU32 *extValue);
  ```

  It is expected that the parameter “value” is set to the peripheral ID that has been assigned to the specific peripheral. The ID can be set in powers of 2 only.

  There is no need to encode the value of the signal itself, calling `driveSignal()` implicitly means that the signal is *asserted* for this cycle. On the other hand, not calling `driveSignal()` implicitly means that the signal is *low* for this cycle. The `driveSignal()` should be called for a single cycle for every transaction.

  The same scheme applies to the response signals *clr* and *tc*. To raise these signals for one cycle, the DMAC Cycle Model calls `driveSignal()` with the first argument containing the peripheral ID (powers of 2 only) of the target. All connected models with a different ID should ignore the call. For sequence of operation for various modes of DMAC, see the PL080 or PL081 Technical Reference Manual.

- INCR transfers of undefined length are accurate functionally, but they are not modeled accurately as RTL timing.

1.1.3 Unsupported Hardware Features

The following features of the DMAC PL080/PL081 hardware are not implemented in the DMAC PL080 and PL081 Cycle Model:

- Pack/Unpack logic and burst operations are functionally accurate.
- ERROR/SPLIT/RETRY response is not supported.
- Lock transactions are 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 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 of the component that are exposed in SoC Designer. See the *ARM PrimeCell DMA Controller Integration Manual* for more information.

Table 1-2 ESL Component Ports

Name	Description	Direction	Type
ahbs	AHB slave port (configuration).	Input	AHB transaction slave
breq	Burst Request signal for the peripheral.	Input	Signal slave
lbreq	Last Burst Request signal for the peripheral.	Input	Signal slave
lsreq	Last Single Request signal for the peripheral.	Input	Signal slave
reset	Input reset. Reset port for receiving reset signal.	Input	Signal slave
sreq	Single Request signal for the peripheral.	Input	Signal slave
clk-in	Input clock.	Input	Clock slave
ahbm0	AHB master port (DMA).	Output	AHB transaction master
ahbm1	AHB master port (DMA). (Exists only in the PL080)	Output	AHB transaction master
clr	Request Acknowledge Clear signal for the peripheral.	Output	Signal master
intp	Interrupt Master Port.	Output	Signal master
tc	Terminal Count signal for the peripheral.	Output	Signal master

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

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]
ahbm0 Align Data	Whether halfword and byte transactions will align data to the transaction size for the *ahbm0* port. By default, data is not aligned.	true, false	false	No
ahbm0 Big Endian	Whether AHB data is treated as big endian for the *ahbm0* port. By default, data is not sent as big endian.	true, false	false	No
ahbm0 Enable Debug Messages	Whether debug messages are logged for the *ahbm0* port.	true, false	false	Yes
ahbm1 Align Data	Whether halfword and byte transactions will align data to the transaction size for the *ahbm1* port. By default, data is not aligned.	true, false	false	No
ahbm1 Big Endian	Whether AHB data is treated as big endian for the *ahbm1* port. By default, data is not sent as big endian.	true, false	false	No
ahbm1 Enable Debug Messages	Whether debug messages are logged for the *ahbm1* port.	true, false	false	Yes
ahbs Align Data	Whether halfword and byte transactions will align data to the transaction size for the *ahbs* port. By default, data is not aligned.	true, false	false	No
ahbs Big Endian	Whether AHB data is treated as big endian for the *ahbs* port. By default, data is not sent as big endian.	true, false	false	No
ahbs Enable Debug Messages	Whether debug messages are logged for the non-secure *ahbs* port.	true, false	false	Yes
ahbs Filter HREADYIN	Whether the HREADYIN signal is filtered to prevent it from reaching the Cycle Model.	true, false	false	No
ahbs region size 0	Region size of the non-secure AHB interface.	0x0 - 0xFFFFFFFF	0x100000000	No
ahbs region size [1-5]	Unused	Not used	0x0	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
ahbs region start 0	Base address of the non-secure AHB interface.	0x0 - 0xffffffff	0x0	No
ahbs region start [1-5]	Unused	Not used	0x0	No
ahbs 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
ahbs Subtract Base Address Dbg	Same description as for *ahbs 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 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
intp int_1 id	The ID used to identify the DMACINTR interrupt.	user-defined *value*	1	Yes
intp int_2 id	The ID used to identify the DMACINTERR interrupt.	user-defined *value*	2	Yes
intp int_3 id	The ID used to identify the DMACINTTC interrupt.	user-defined *value*	3	Yes
Waveform File [2]	Name of the waveform file.	*string*	arm_cm_[*model_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.

1.5 Debug Features

The PL080/PL081 Cycle Model has a debug interface (CADI) that allows the user to view, manipulate, and control the registers and memory in SoC Designer Simulator, or any debugger that supports CADI. 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 PL080/PL081 Cycle Model has four sets of registers that are accessible via the debug interface. Registers are grouped into sets according to functional area. The DMAC PL081 Cycle Model has only two DMA channels, therefore, only the registers corresponding to the two DMA channels are present in the PL081 Cycle Model.

The registers are listed below:

- General Registers
- Peripheral Identification Registers
- PrimeCell Identification Registers
- Test Registers

See the *ARM PrimeCell Dynamic Memory Access Controller Technical Reference Manual* for detailed descriptions of these registers.

1.5.1.1 General Registers

The General group contains registers that control the DMAC.

Table 1-4 General Registers Summary

Name	Description	Type
DMACIntStatus	Interrupt Status. Shows the status of the interrupts after masking.	read-only
DMACIntTCStatus	Interrupt Terminal Count Status. Indicates the status of the terminal count after masking.	read-only
DMACIntTCClear	Interrupt Terminal Count Clear. Clears a terminal count interrupt request.	write-only
DMACIntErrorStatus	Interrupt Error Status. Indicates the status of the error request after masking.	read-only
DMACIntErrClr	Interrupt Error Clear. Clears the error interrupt requests.	write-only
DMACRawIntTCStatus	Raw Interrupt Terminal Count Status. Indicates the DMA channels that are requesting a transfer complete, terminal count interrupt, prior to masking.	read-only
DMACRawIntErrorStatus	Raw Interrupt Error Status. Indicates the DMA channels that are requesting an error interrupt prior to masking.	read-only
DMACEnbldChns	Enabled Channels. Indicates the DMA channels that are enabled.	read-only

Table 1-4 General Registers Summary (continued)

Name	Description	Type
DMACSoftBReq	Software Burst Transfer Request. Enables DMA burst requests to be generated by software.	read-write
DMACSoftSReq	Software Single Transfer Request. Enables DMA single requests to be generated by software.	read-write
DMACSoftLBReq	Software Last Burst Transfer Request. Enables software to generate DMA last burst requests.	read-write
DMACSoftLSReq	Software Last Single Transfer Request. Enables software to generate DMA last single requests.	read-write
DMACConfiguration	DMAC Configuration. Configures the operation of the DMAC.	read-write
DMACSync	DMAC Synchronization. Enables or disables synchronization logic for the DMA request signals.	read-write
DMACC0SrcAddr - DMACC7SrcAddr [1]	Channel Source Address Registers. These 8 registers contain the current source address, byte-aligned, of the data to be transferred.	read-write
DMACC0DestAddr - DMACC7DstAddr [1]	Channel Destination Address Registers. These 8 registers contain the current destination address, byte-aligned, of the data to be transferred.	read-write
DMACC0LLI - DMACC7LLI [1]	Channel Linked List Item Registers. These 8 registers contain a word-aligned address of the next LLI.	read-write
DMACC0Control - DMACC7Control [1]	Channel Control Registers. These 8 registers contain DMA channel control information such as the transfer size, burst size, and transfer width.	read-write
DMACC0Config - DMACC7Config [1]	Channel Configuration Registers. These 8 registers configure the DMA channel.	read-write

1. The PL081 provides only two of each of these registers (DMACC0 and DMACC1) because it has only two DMA channels.

1.5.1.2 Peripheral Identification Registers

The Peripheral ID group contains registers that enable system firmware to identify a peripheral.

Table 1-5 Peripheral Registers Summary

Name	Description	Type
DMACPeriphID0	Peripheral ID 0	read-only
DMACPeriphID1	Peripheral ID 1	read-only
DMACPeriphID2	Peripheral ID 2	read-only
DMACPeriphID3	Peripheral ID 3	read-only

1.5.1.3 PrimeCell Identification Registers

The PrimeCell ID group contains registers that enable system firmware to identify a PrimeCell component.

Table 1-6 PrimeCell Registers Summary

Name	Description	Type
DMACPCellID0	PrimeCell ID 0	read-only
DMACPCellID1	PrimeCell ID 1	read-only
DMACPCellID2	PrimeCell ID 2	read-only
DMACPCellID3	PrimeCell ID 3	read-only

1.5.1.4 Test Registers

The Test group contains the DMAC test registers.

Table 1-7 Test Registers Summary

Name	Description	Type
DMACITCR	Test control. Enables you to test the DMAC.	read-write
DMACITOP1	Integration test output register 1.	read-only
DMACITOP2	Integration test output register 2.	read-only
DMACITOP3	Integration test output register 3.	read-only

1.6 Available Profiling Data

The PL080/PL081 Cycle Model component has no profiling capabilities.