Cortex®-M0+ Cycle Model

Version 9.6.0

User Guide

Non-Confidential

101164_0906_00 (ID103117)

Cortex-M0+ Cycle Model
User Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
November 2017	0906-00	Non-Confidential	Release with 9.6

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Product Status

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

Web Address

http://www.arm.com

Contents

Preface

Chapter 1.
Using the Cycle Model in SoC Designer

Cortex-M0+ Functionality 12
Adding and Configuring the SoC Designer Component 13
SoC Designer Component Files 13
Adding the Cycle Model to the Component Library 14
Adding the Component to the SoC Designer Canvas 14
ESL Ports 15
Available Component ESL Ports 15
Tied Pins 15
Setting Component Parameters 16
Debug Features 19
Memory Information 19

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 Cortex-A32 multi-processor 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 publication provides information that relates directly to SoC Designer:

- *SoC Designer User Guide* (100996)

The following publications provide reference information about Arm products:

- *Arm Cortex-M0+ Technical Reference Manual* (DDI 0484)
- *CoreSight Architecture Specification* (IHI 0029)

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.

CHI	The AMBA® 5 Coherent Hub Interface specification. A bus protocol with coherency channels designed to support high frequency, non-blocking data transfers between multiple coherent processors.
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	The full name is *SoC Designer*. A 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:

- Cortex-M0+ Functionality
- Adding and Configuring the SoC Designer Component
- ESL Ports
- Setting Component Parameters

1.1 Cortex-M0+ 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, see the *Cortex-M0+ Technical Reference Manual*.

The following features of the Cortex-M0+ hardware are fully implemented in the Cortex-M0+ Cycle Model:

- Cortex-M0+ Integer Core
- NVIC – Nested Vectored Interrupt Controller
- WIC – Wakeup Interrupt Controller
- AHB-Lite: System Bus Interface
- BPU– BreakPoint Unit
- DWT – Data Watchpoint and Trace
- ROM Table

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* (100996) 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 Carbon 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. The component's appearance may vary depending on your specific device configuration. Additional ports are provided depending on the Cycle Model RTL configuration file (*.conf*), used to create the Cycle Model.

1.3 ESL Ports

This section describes the differences between the pins listed in the *Arm Cortex-M0+ Technical Reference Manual* (DDI 0484) and those on the Cortex-M0+ Cycle Model. Certain hardware pins have been converted to init-time Cycle Model parameters.

- Available Component ESL Ports — Describes ports that have been added to the Cycle Model, such as clocks and resets required by SoC Designer Plus, or those created by wrapping multiple hardware pins into transactors.
- Tied Pins — Describes pins that are tied for performance reasons.

1.3.1 Available Component ESL Ports

Table 1-2 describes ports that have been added to the Cycle Model. Additional ports are visible in SoC Designer Plus, which correspond to the hardware pins. See the *Arm Cortex-M0+ Technical Reference Manual* (DDI 0484) or the *CoreSight MTB-M0+ Implementation and Integration Manual* (DIT 0031) for descriptions of these pins.

Note: Some ESL component port values can be set using a component parameter. In those cases, the parameter value is used whenever the ESL port is not connected. If the port is connected, the connection value takes precedence over the parameter value.

Table 1-2 ESL Component Ports

ESL Port	Description	Type
AHBLite_Master	AHB Lite Master transaction master port. This port implements the AMBA AHB-Lite interface. This transaction master port should be connected to an AHBv2 slaves using either an MxAHBv2 bus component (where one side is an AHB Lite Master and the other side is an AHB Lite Slave) or a PL301 in between. There are a few AHBv2 sideband signals defined specifically for the Cortex-M0+. See the *AHBv2 Protocol Bundle User Guide* (101027) for details on AHB Cortex-M0+ extension signals.	Transaction Master
DCLK	Clock for the processor debug domain.	Clock Slave
HCLK	Clock for the majority of the non-debug logic in the processor system domain.	Clock Slave
SCLK	Free running clock that clocks a small amount of logic in the processor system domain.	Clock Slave

1.3.2 Tied Pins

STCLKEN is tied low.

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 component's parameters:

1. In the Canvas, right-click on the component and select **Component Information**. You can also double-click the component. The *Edit Parameters* dialog box appears.

   The list of available parameters will be slightly different depending on the settings that you enabled in the configuration file when creating 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.

Refer to the *CoreSight MTB-M0+ Implementation and Integration Manual* (DIT 0031) for details about these parameters.

Table 1-3 Component Parameters

Name	Description	Allowed Values	Default Value	Init/ Runtime
AHBLite_Master Align Data	Determines whether halfword and byte transactions will align data to the transaction size for this port. By default, data is not aligned.	true, false	false	Init
AHBLite_Master Big Endian	Determines whether AHB data is treated as big endian for this port. By default, data is not sent as big endian.	true, false	false	Init
AHBLite_Master Enable Debug Messages	Determines whether debug messages are logged for the *mem_D* port.	true, false	false	Runtime
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	Init
Carbon DB Path	Sets the directory path to the database file.	Not Used	empty	Init
CPUWAIT	1 — Causes the processor to wait for the signal to be LOW before coming out of reset. 0 — Allows processor to come out of reset regardless of signal status.	0, 1	0	Runtime
DBGEN	Enables and disables debug. 1 — Debug enabled 0 — Debug disabled	0, 1	0	Runtime

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Init/ Runtime
DBGRESTART	External restart request.	0, 1	0	Runtime
DFTSE	Scan-enable input.	0, 1	0	Runtime
Dump Waveforms	Determines whether SoC Designer dumps waveforms for this component.	true, false	false	Runtime
ECOREVNUM	Provides a way to implement engineering change order modification for certain bits in the architected ID registers in the processor, DAP and CoreSight MTB-M0+.[1]	20-bit integer	0	Runtime
EDBGRQ	External debug request.	0, 1	0	Runtime
Enable Debug Messages	Determines whether debug messages are logged for the component.	true, false	false	Runtime
IOMATCH	I/O address decoder response: 0 — LOW Address on IOCHECK uses AHB. 1 — HIGH Address on IOCHECK uses I/O port.	0, 1	0	Runtime
IORDATA	I/O port read data, for reads.	32-bit integer	0	Runtime
IRQ	External interrupt signals.	32-bit integer	0	Runtime
IRQLATENCY	Specifies the minimum number of cycles between an interrupt that becomes pended in the NVIC, and the vector fetch for that interrupt being issued on the AHB-Lite interface.	8-bit integer	0	Runtime
NIDEN	Disables and enables trace. — To permanently disable trace, tie this signal LOW. — To dynamically enable and disable trace, connect this signal to your own logic.	0, 1	0	Runtime
NMI	Non-Maskable Interrupt.	0, 1	0	Runtime
RXEV	A HIGH level on this input causes the Arm v6-M architecture defined Event Register to be set in the Cortex-M0+ processor. This causes a WFE instruction to complete. It also awakens the processor if it is sleeping as the result of encountering a WFE instruction when the Event Register is clear.		0	Runtime
SLEEPHOLDREQn	Request to extend the processor sleeping state regardless of wake-up events.	0, 1	0	Runtime

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Init/ Runtime
SLVADDR	Connect to the SLVADDR port of the Cortex-M0+ DAP, or the HADDR port of a CoreSight AHB-AP.	32-bit integer	0	Runtime
SLVPROT	Connect to the SLVPROT port of the Cortex-M0+ DAP, or the HPROT[3:0] port of a CoreSight AHB-AP.	4-bit integer or 0 - 15	0	Runtime
SLVSIZE	Connect to the SLVSIZE port of the Cortex-M0+ DAP, or the HSIZE[1:0] port of a CoreSight AHB-AP.	2-bit integer or 0 - 3	0	Runtime
SLVSTALL	Drive HIGH to stall accesses on the SLV port.	0, 1	0	Runtime
SLVTRANS	Connect to SLVTRANS port of the Cortex-M0+ DAP or the HTRANS port of a CoreSight AHB-AP.	2-bit integer	0	Runtime
SLVWDATA	Connect to the SLVWDATA port of the Cortex-M0+ DAP, or the HWDATA port of a CoreSight AHB-AP.	32-bit value	0	Runtime
SLVWRITE	Connect to the SLVWRITE port of the Cortex-M0+ DAP, or the HWRITE port of a CoreSight AHB-AP.	0, 1	0	Runtime
STCALIB	SysTime Clock Calibration	26-bit integer	0x207A11F	Init
Waveform File [2]	Name of the waveform file.	string	CortexM0Plus.vcd	Init
Waveform Format	The format of the waveform dump file.	FSDB, VCD	VCD	Init
Waveform Timescale	Sets the timescale to be used in the waveform.	Many values in drop-down	1 ns	Init
WICDSREQn	Active LOW request for deep sleep to be WIC-based deep sleep.	0, 1	0	Runtime

1. Refer to the *CoreSight MTB-M0+ Implementation and Integration Manual* (DIT 0031) for details.
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 Cortex-M0+ Cycle Model has a debug interface (CADI) that allows the user to view, manipulate, and control the memory. A view can be accessed in SoC Designer Simulator by right clicking on the Cycle Model and choosing the appropriate menu entry.

- Memory Information

1.5.1 Memory Information

The Cortex-M0+ supports a basic memory view, which shows the CPU's debug view of code memory. In SoC Designer Simulator, access the **Debug > View Memory for...** menu.

Third Party Software Acknowledgement

Arm acknowledges and thanks the respective owners for the following software that is used by our product:

- **ELF (Executable and Linking Format) Tool Chain Product**