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

About This Guide ................................................................. 7
Audience ............................................................................. 7
Conventions ........................................................................ 8
Further reading .................................................................... 8
Glossary ............................................................................. 9

Chapter 1.

NIC-400 Cycle Model Functionality ........................................ 12
    Implemented Hardware Features ........................................ 12
    Hardware Features not Implemented ................................. 13
    Features Additional to the Hardware ............................... 13
    Adding and Configuring the SoC Designer Component .......... 14
        SoC Designer Component Files .................................. 14
        Adding the Cycle Model to the Component Library .......... 15
        Adding the Component to the SoC Designer Canvas ........ 15
    Available Component ESL Ports ..................................... 16
    Setting Component Parameters ......................................... 17
        Using the Track In-Flight Data parameter .................... 19
    Debug Features .................................................................. 20
        Register Information .................................................. 20
    Available Profiling Data .................................................... 22
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 NIC-400 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:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>courier</td>
<td>Commands, functions, variables, routines, and code examples that are set apart from ordinary text.</td>
<td><code>sparseMem_t SparseMemCreateNew();</code></td>
</tr>
<tr>
<td>italic</td>
<td>New or unusual words or phrases appearing for the first time.</td>
<td><code>Transactors provide the entry and exit points for data ...</code></td>
</tr>
<tr>
<td>bold</td>
<td>Action that the user performs.</td>
<td>Click <strong>Close</strong> to close the dialog.</td>
</tr>
<tr>
<td>&lt;text&gt;</td>
<td>Values that you fill in, or that the system automatically supplies.</td>
<td><code>&lt;platform&gt;/</code> represents the name of various platforms.</td>
</tr>
<tr>
<td>[ text ]</td>
<td>Square brackets [] indicate optional text.</td>
<td><code>$/CARBON_HOME/bin/modelstudio [ &lt;filename&gt; ]</code></td>
</tr>
<tr>
<td>[ text1</td>
<td>text2 ]</td>
<td>The vertical bar</td>
</tr>
</tbody>
</table>

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

The following publications provide information related to SoC Designer:

- *SoC Designer User Guide* (100996)

The following publications provide reference information about additional Arm products:

- *CoreLink NIC-400 Network Interconnect Technical Reference Manual* (DDI0475)

See [http://infocenter.arm.com/help/index.jsp](http://infocenter.arm.com/help/index.jsp) for access to Arm documentation.

The following publications provide additional information on simulation:

**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.
<table>
<thead>
<tr>
<th><strong>RTL</strong></th>
<th><em>Register Transfer Level.</em> A high-level hardware description language (HDL) for defining digital circuits.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SoC Designer</strong></td>
<td>The full name is <em>SoC Designer.</em> A high-performance, cycle accurate simulation framework which is targeted at System-on-a-Chip hardware and software debug as well as architectural exploration.</td>
</tr>
<tr>
<td><strong>SystemC</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Transactor</strong></td>
<td><em>Transaction adaptors.</em> You add transactors to your component to connect your component directly to transaction level interface ports for your particular platform.</td>
</tr>
</tbody>
</table>
Chapter 1

Using the Cycle Model Component 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:

• NIC-400 Cycle Model Functionality
• Adding and Configuring the SoC Designer Component
• Available Component ESL Ports
• Setting Component Parameters
• Debug Features
• Available Profiling Data
1.1 NIC-400 Cycle Model Functionality

The AMBA Network Interconnect is a highly configurable component that enables you to create a complete high performance, optimized AMBA-compliant network infrastructure. The possible configurations for the AMBA Network Interconnect can range from a single bridge component, for example an AHB to AXI protocol conversion bridge, to a complex infrastructure that consists of up to 128 masters and 64 slaves of a combination of different AMBA protocols. For information about these components, see the CoreLink NIC-400 Network Interconnect Technical Reference Manual (DD10475).

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.

- Implemented Hardware Features
- Hardware Features not Implemented
- Features Additional to the Hardware

1.1.1 Implemented Hardware Features

The NIC-400 is a highly configurable infrastructure component that supports:

- 1-64 slave interfaces that can be AXI3, AXI4, AHB-Lite slave or mirrored master.
- 1-128 master interfaces that can be AXI3, AXI4, AHB-Lite master or mirrored slave, APB3, or APB4.
- Hierarchical clock-gating to reduce idle or near-idle power.
- Configuration of:
  - an APB AMIB, which can have up to 16 subports. Each subport can be APB3 or APB4.
  - an AXI port to support four region control bits.
- Single-cycle arbitration.
- Full pipelining to prevent master stalls.
- Programmable control for FIFO transaction release.
- Multiple switch networks.
- Complex topologies, including Network On Chip (NOC) loop-back connections between switches.
- Up to five cascaded switch networks between any master and slave interface pair.
- AXI or AHB-Lite masters and slaves with:
  - an address width of 32-64 bits.
  - a data width of 32, 64, or 128 bits.
- Non-contiguous APB slave address map for a single master interface.
- Independent widths of user-defined sideband signals for each channel.
• Global Programmers View (GPV) for the entire infrastructure that you can configure so that any master, or a discrete configuration slave interface, can access it.

• Highly flexible timing closure options.

1.1.2 Hardware Features not Implemented

The following features of the NIC-400 hardware are not implemented in the Cycle Model:

• The hardware can support a data width choice of 256 bits; the Cycle Model supports only 32, 64, or 128 bits.

• Remap.

• Quality of Service (QoS) signalling.

• APB2 Protocol support.

1.1.3 Features Additional to the Hardware

The following features that are implemented in the NIC-400 Cycle Model do not exist in the NIC-400 hardware. These features have been added to the Cycle Model for enhanced usability.

• For debug access, the Cycle Model supports read-write accessibility for certain registers that are write-only in the hardware (the Registers view in SoC Designer shows Read/Write for these registers). NIC-400 Cycle Accurate Cycle Model behavior is identical to that of the hardware.
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>
<thead>
<tr>
<th>Platform</th>
<th>File Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>maxlib.lib&lt;component_name&gt;.conf</td>
<td>SoC Designer configuration file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx.so</td>
<td>SoC Designer component runtime file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx_DBG.so</td>
<td>SoC Designer component debug file</td>
</tr>
<tr>
<td>Windows</td>
<td>maxlib.lib&lt;component_name&gt;.windows.conf</td>
<td>SoC Designer configuration file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx.dll</td>
<td>SoC Designer component runtime file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx_DBG.dll</td>
<td>SoC Designer component debug file</td>
</tr>
</tbody>
</table>

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<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.

The “AXI” component includes three AXI slave interface ports and two AXI master interface ports. The “AXI, AHB, and APB” component includes two AXI slave interface ports and one AHB-Lite slave interface port. It also has one AXI master interface port, one AHB master interface port, and one APB master interface port. Your component may appear with fewer or more ports, depending on how it was configured.

Depending on how you configured the ports, the port names may be more descriptive. The names are fully customizable.
### 1.3 Available Component ESL Ports

Table 1-2 describes the ESL ports of the component that are exposed in SoC Designer.

**Table 1-2 ESL Component Ports**

<table>
<thead>
<tr>
<th>ESL Port</th>
<th>Description</th>
<th>Direction</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;clock_domain_name&gt;clk</code></td>
<td>Clock signal, or signals, for each defined clock domain.</td>
<td>input</td>
<td>Clock slave</td>
</tr>
<tr>
<td><code>&lt;clock_domain_name&gt;clk_r</code></td>
<td>This type of port is available if a transactor port was configured to have a view to the GPV. It must have the same clock frequency as the clock domain clock and must be synchronous to it.</td>
<td>input</td>
<td>Clock slave</td>
</tr>
<tr>
<td><code>&lt;clock_domain_name&gt;resetn</code></td>
<td>Reset signal, or signals, for each defined clock domain.</td>
<td>input</td>
<td>Signal slave</td>
</tr>
<tr>
<td><code>&lt;clock_domain_name&gt;resetn_r</code></td>
<td>Reset signal corresponding to the <code>&lt;clock_domain_name&gt;clk_r</code>, if one exists.</td>
<td>input</td>
<td>Signal slave</td>
</tr>
<tr>
<td><code>clk-in</code></td>
<td>Input clock. When using the <code>&lt;clock_domain_name&gt;clk</code> port, the <code>clk-in</code> port should not be connected.</td>
<td>input</td>
<td>Clock slave</td>
</tr>
</tbody>
</table>

**Protocol Type Master Interface Ports**

There is a master interface port created for each port defined in AMBA Designer. For example, `AXI4_slave_00`.

**Protocol Type Slave Interface Ports**

There is a slave interface port created for each port defined in AMBA Designer. For example, `AXI4_master_00`.

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

**Note:** You can set some ESL component port values using a component parameter. This includes the `<clock_domain_name>resetn` and `<clock_domain_name>resetn_r` 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 component’s parameters:

1. In the Canvas, right-click on the component and select **Edit Parameters**. 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default Value</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Align Waveforms</td>
<td>When set to <em>true</em>, 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 <em>false</em>, the reset sequence is dumped to the waveform data, however, the component time is not aligned with the SoC Designer time.</td>
<td>true, false</td>
<td>true</td>
<td>No</td>
</tr>
<tr>
<td>ARM CycleModels DB Path</td>
<td>Sets the directory path to the database file.</td>
<td>Not Used</td>
<td>empty</td>
<td>No</td>
</tr>
<tr>
<td>Dump Waveforms</td>
<td>Whether SoC Designer dumps waveforms for this component.</td>
<td>true, false</td>
<td>false</td>
<td>Yes</td>
</tr>
<tr>
<td>Enable Debug Messages</td>
<td>Whether debug messages are logged for the component.</td>
<td>true, false</td>
<td>false</td>
<td>Yes</td>
</tr>
<tr>
<td>Fast Application Load Support</td>
<td>Identifies that the component supports fast debug access for application load.</td>
<td>Not configurable</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td><code>&lt;clock_domain_name&gt;</code> resetn</td>
<td>Sets the value for the specified Reset signal, or signals, corresponding to the specified clock domain.</td>
<td>0x0, 0x1</td>
<td>0x1</td>
<td>Yes</td>
</tr>
<tr>
<td><code>&lt;clock_domain_name&gt;</code> resetn_r</td>
<td>If a <code>&lt;clock_domain_name&gt;</code> resetn_r signal exists, this parameter sets the value for the specified Reset signal, or signals, corresponding to the specified clock domain.</td>
<td>0x0, 0x1</td>
<td>0x1</td>
<td>Yes</td>
</tr>
<tr>
<td><code>&lt;port_protocol&gt;</code>&lt;slave port name&gt; _size[0-5]²</td>
<td>Sizes of memory regions.</td>
<td>0x0 - 0x100000000</td>
<td>size0 default is 0x100000000, size1-5 default is 0</td>
<td>No</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Allowed Values</td>
<td>Default Value</td>
<td>Runtime¹</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>----------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td><code>&lt;port_protocol&gt;_&lt;slave port name&gt;_&lt;start[0-5]</code></td>
<td>Start addresses of memory regions.</td>
<td>0x0 - 0xffffffff</td>
<td>0x00000000</td>
<td>No</td>
</tr>
<tr>
<td><code>&lt;port_protocol&gt;_&lt;master port name&gt; Enable Debug Messages</code></td>
<td>Whether debug messages are logged for the master ports. There is one parameter for each master port.</td>
<td>true, false</td>
<td>false</td>
<td>Yes</td>
</tr>
<tr>
<td><code>&lt;port_protocol&gt;_&lt;slave port name&gt; Enable Debug Messages</code></td>
<td>Whether debug messages are logged for the slave ports. There is one parameter for each slave port.</td>
<td>true, false</td>
<td>false</td>
<td>Yes</td>
</tr>
<tr>
<td>Track In-Flight Data³</td>
<td>Enables or disables debug access for in-flight transactions.</td>
<td>true, false</td>
<td>true</td>
<td>No</td>
</tr>
<tr>
<td>Waveform File ⁴</td>
<td>Name of the waveform file.</td>
<td>string</td>
<td><code>arm_cm_pl400_NIC400&lt;component_name&gt;.vcd</code></td>
<td>No</td>
</tr>
<tr>
<td>Waveform Format</td>
<td>The format of the waveform dump file.</td>
<td>VCD, FSDB</td>
<td>VCD</td>
<td>No</td>
</tr>
<tr>
<td>Waveform Timescale</td>
<td>Sets the timescale to be used in the waveform.</td>
<td>Many values in drop-down</td>
<td>1 ns</td>
<td>No</td>
</tr>
</tbody>
</table>

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 is taken into account only at the next reset.

2. The square brackets indicate the memory regions available for the Cycle Model. For example, the parameter name for the start addresses “s_[0-1]_start[0-5]” will be expanded to 12 possible parameter name combinations that range from “s_0_start0” to “s_1_start5”. The size of a memory region depends on the “s[N]_start[M]” and “s[N]_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. Refer to the Section “Using the Track In-Flight Data parameter” below for more about this parameter.

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.
1.4.1 Using the Track In-Flight Data parameter

The Track In-Flight Data parameter should be set to True (this is the default) if you are using any `debugAccess` functions in the NIC-400. `debugAccess` functions are used to implement the Memory view of a processor and any Disassembly view of memory. If in-flight tracking is disabled, then the information in one of these views may be incorrect while there are incomplete write transactions within the NIC-400.

Disabling Track In-Flight Data may be appropriate if the accuracy of a processor’s Disassembly view or Memory view is not critical. Whether or not Track In-Flight Data is enabled, opening a Memory view from the desired memory always provides an accurate view of its contents.
1.5 Debug Features

The NIC-400 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 NIC-400 Cycle Model registers (Table 1-4) are grouped by master. The register view shows one tab per master that has register visibility. In this tab, all the registers visible to that master are shown.

Table 1-4 NIC-400 Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>remap</td>
<td>Remap register, up to eight global remap states are available.</td>
<td>read-write</td>
</tr>
<tr>
<td>security0</td>
<td>Slave 0 security setting. 1 bit for non-virtual slaves, up to 16 for virtual or APB master interfaces, and you can configure it as follows: 0 Secure 1 Non-secure</td>
<td>read-write</td>
</tr>
<tr>
<td>security1</td>
<td>Slave 1 security setting. 1 bit for non-virtual slaves, up to 16 for virtual or APB master interfaces, and you can configure it as follows: 0 Secure 1 Non-secure</td>
<td>read-write</td>
</tr>
<tr>
<td>security&lt;n&gt;</td>
<td>Slave n security setting. 1 bit for non-virtual slaves, up to 16 for APB master interfaces.</td>
<td>read-write</td>
</tr>
<tr>
<td>Peripheral_ID0</td>
<td>Peripheral Identification Register 0 - Part Number</td>
<td>read-only</td>
</tr>
<tr>
<td>Peripheral_ID1</td>
<td>Peripheral Identification Register 1 - JEP106 and Part Number</td>
<td>read-only</td>
</tr>
<tr>
<td>Peripheral_ID2</td>
<td>Peripheral Identification Register 2 - Revision, JEP106 code flag, JEP106</td>
<td>read-only</td>
</tr>
<tr>
<td>Peripheral_ID3</td>
<td>Peripheral Identification Register 3 - Can be set using the AMBA Designer Graphical User Interface (GUI)</td>
<td>read-only</td>
</tr>
<tr>
<td>Peripheral_ID4</td>
<td>Peripheral Identification Register 4 - 4KB count, JEP106 continuation code</td>
<td>read-only</td>
</tr>
<tr>
<td>Peripheral_ID5</td>
<td>Peripheral Identification Register 5 - Reserved</td>
<td>read-only</td>
</tr>
<tr>
<td>Peripheral_ID6</td>
<td>Peripheral Identification Register 6 - Reserved</td>
<td>read-only</td>
</tr>
<tr>
<td>Peripheral_ID7</td>
<td>Peripheral Identification Register 7 - Reserved</td>
<td>read-only</td>
</tr>
<tr>
<td>Component_ID0</td>
<td>Component Identification Register 0 - Preamble</td>
<td>read-only</td>
</tr>
<tr>
<td>Component_ID1</td>
<td>Component Identification Register 1 - Generic IP component class, preamble</td>
<td>read-only</td>
</tr>
</tbody>
</table>
### Table 1-4  NIC-400 Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component_ID2</td>
<td>Component Identification Register 2 - Preamble</td>
<td>read-only</td>
</tr>
<tr>
<td>Component_ID3</td>
<td>Component Identification Register 3 - Preamble</td>
<td>read-only</td>
</tr>
<tr>
<td>sync_mode</td>
<td>This 3-bit register is valid only with a FIFO for all channels. You can configure the bits to create different clock domain boundaries as follows: 0: sync 1:1, 1: sync n:1, 2: sync 1:n, 3: sync m:n, 4: async, 5: reserved, 6: reserved, 7: reserved</td>
<td>read-write</td>
</tr>
<tr>
<td>fn_mod</td>
<td>This 1-bit register is the issuing functionality modification register. Issuing override sets block issuing capability to be forced to one transaction, and you can configure the bit as follows: 0: Read issuing, read_iss_override, 1: Write issuing, write_iss_override</td>
<td>read-write</td>
</tr>
<tr>
<td>fn_mod2</td>
<td>Bypass merge, only if upsizing or downsizing. See the Upsizing data width function and Downsizing data width function sections in the TRM for more information.</td>
<td>read-write</td>
</tr>
<tr>
<td>fn_mod_ahb</td>
<td>This 3-bit register is valid for AHB interfaces only. You can configure the bits of this register as follows: bit 0: rd_incr_override, bit 1: wr_incr_override, bit 2: lock_override. In the TRM, see Lock transactions for information on overriding locks, and see Combination 4 for information on wr_incr_override and rd_incr_override.</td>
<td>read-write</td>
</tr>
<tr>
<td>fn_mod_iss_bm</td>
<td>See the TRM for more information.</td>
<td>read-write</td>
</tr>
<tr>
<td>ahb_cntl</td>
<td>This 2-bit register is valid for AHB interfaces only. You can configure the bits as follows: bit 0: decerr_en, bit 1: force_incr. See AHB master interfaces in the TRM for more information.</td>
<td>read-write</td>
</tr>
<tr>
<td>wr_tidemark</td>
<td>This 4-bit register is valid only with a FIFO for the WFIFO channel, and if not an AHB slave interface. See the FIFO and clocking function section in the TRM for more information.</td>
<td>read-write</td>
</tr>
</tbody>
</table>
1.6 Available Profiling Data

The NIC-400 Cycle Model component has no profiling capabilities.