SoC Designer
System Analyzer User Guide

Copyright © 2016 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2016</td>
<td>A</td>
<td>Non-Confidential</td>
<td>Rebrand update for 9.0.0</td>
</tr>
</tbody>
</table>

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM Limited (“ARM”). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any patents.

This document may include technical inaccuracies or typographical errors.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version shall prevail.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to ARM’s customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement specifically covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms.
A.2.3 Creating the Project ................................................................. A-4
A.3 Enabling Plugins to Run in System Analyzer .............................. A-15
A.4 Enabling Plugins to Run as a Console Application .................... A-16
   A.4.1 Wrapper Script approach ................................................. A-16
   A.4.2 Environment Variable approach ..................................... A-16
A.5 Driving the Program from a Test Driver ................................ A-17
   A.5.1 Visual Studio Instructions .............................................. A-17
   A.5.2 MonoDevelop Instructions ............................................ A-18
A.6 Plugin Method Reference .......................................................... A-20
   A.6.1 Name and Description Methods ..................................... A-20
   A.6.2 Accepts Method ......................................................... A-20
   A.6.3 Analyze Method .......................................................... A-23
A.7 Running ValidateMetrics with the CHI Example ......................... A-24
   A.7.1 Modifying ValidateMetrics to use with custom profiles ... A-24

Appendix B Metrics and Formulas
B.1 AXI4 Metrics and Formulas .................................................... B-2
B.2 CHI Metrics and Formulas ...................................................... B-6

Appendix C Revisions
Preface

This preface introduces the SoC Designer System Analyzer User Guide. It contains the following sections:

- About this document

About this document

This book describes how to use the System Analyzer plugin for SoC Designer.

Intended audience

This book is written for experienced hardware engineers, software engineers and System-on-Chip (SoC) designers who have experience of ARM products.

Glossary

The ARM® Glossary is a list of terms used in ARM documentation, together with definitions for those terms. The ARM® Glossary does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See the ARM® Glossary:

http://www.arm.com
Typographical conventions

The following typographical conventions are used in this book:

italic        Highlights important notes, introduces special terminology, 
denotes internal cross-references, and citations.

bold          Highlights interface elements, such as menu names. Denotes 
processor signal names. Also used for terms in descriptive lists, 
where appropriate.

monospace     Denotes text that can be entered at the keyboard, such as 
commands, file and program names, and source code.

monospace     Denotes a permitted abbreviation for a command or option. The 
underlined text can be entered instead of the full command or 
option name.

monospace italic Denotes arguments to commands and functions where the 
argument is to be replaced by a specific value.

monospace bold Denotes language keywords when used outside example code.

< and >       Enclose replaceable terms for assembler syntax where they appear 
in code or code fragments. For example:

```c
MRC pl5, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>
```

Further reading

This section lists publications by ARM.

Cycle Model publications

This book contains information that is specific to this product. See the following 
documents for other relevant information:

•  *SoC Designer Installation Guide* (ARM DUI 0953)

•  *SoC Designer User Guide* (ARM DUI 0956)
Chapter 1
Overview of System Analyzer

This chapter describes:
• Features and Benefits
• Requirements
• Restrictions and Limitations
• Suggested Reading
1.1 Features and Benefits

System Analyzer provides the following features and benefits:

- Easy setup and integration with SoC Designer.
- Captures all stream, transaction, and software profiling data for entire simulations.
- Supports AXIv2, AXI4, ACE, ACELite, ACELite+DVM, and CHI transactions.
- Allows viewing and comparison of multiple simulations.
- Supports profile inspection at all degrees of granularity, from high-level to fine-grained.
- Support for large datasets without expensive load time or memory burden.
- Fast redraw.
- Quick oversight of violated metrics.
- Data capture is configurable via the System Analyzer API and by creating custom plugins within SoC Designer; example plugins are included that you can modify. Refer to Appendix A for more information.
- Includes a prebuilt Validation Metrics Plugin which computes useful metrics such as latency, throughput, channel utilization, and bus utilization.
1.2 Requirements

This section describes software and platform requirements for System Analyzer.

1.2.1 Software Requirements

- SoC Designer Version 9.0.0 or later.
- You can develop System Analyzer plugins using a Windows IDE such as Visual Studio 2013 or a Linux IDE such as MonoDevelop .NET 4.0.

Alternatively, you can write your plugin independent of an IDE and use your system’s native compiler to compile it. Refer to Appendix A for more information.

1.2.2 Platform Requirements

System Analyzer is supported on both Linux and Windows platforms. Refer to the SoC Designer Plus Installation Guide (ARM DUI 0956) for supported versions.
1.3 Restrictions and Limitations

- The internal SoC Designer Profiler does not work with System Analyzer enabled.
- AHB and APB protocols are not supported.
1.4 Suggested Reading

Refer to the following ARM documentation for further information on profiling and System Analyzer:

• SoC Designer User Guide
• System Analyzer API Reference Guide
• ARM AMBA5 CHI Architecture Specification (12 June 2014, Rev. A)
• ARM AMBA AXI and ACE Protocol Specification
Chapter 2
Using System Analyzer

This chapter describes:

- Enabling System Analyzer Profiling
- System Analyzer GUI Reference
- Viewing Profiling Data
- Comparing Multiple Simulations
- Using the System Metrics Plugin
2.1 Enabling System Analyzer Profiling

To run System Analyzer on your design simulation, first enable System Analyzer functionality in SoC Designer. When you configure monitors and run your simulation, System Analyzer outputs profiling data to a .profile file, which you then open and view using the System Analyzer GUI.

This section includes:

• Important Operational Notes
• Enabling System Analyzer and Generating Data
• Launching the System Analyzer GUI and Viewing Profiled Data

2.1.1 Important Operational Notes

Before you begin, review the following important information about how System Analyzer operates in the SoC Designer environment:

• In the SoC Designer GUI, profiling using System Analyzer is referred to as external profiling.

• When external profiling is enabled in the SoC Designer GUI, enabling a monitor on a connection, then simulating, logs transaction data to a .profile file for use by System Analyzer. Enabling Profiler probes does not generate System Analyzer-accessible data; Profiler probes generate profiling data for SoC Designer standard/internal profiling.

  Note

For profiling of a particular connection to show up in the .profile file for use by System Analyzer, you must configure monitors.

• External profiling requires data to be periodically flushed from the internal SoC Designer Plus buffers. Because standard/internal SoC Designer profiling needs the data in these buffers, it no longer works when you enable System Analyzer.

• System Analyzer does not gather data on designs that are currently open. You must close and re-open System Analyzer to begin using System Analyzer with these designs.
## 2.1.2 Enabling System Analyzer and Generating Data

To enable System Analyzer functionality in SoC Designer, and generate data for use with System Analyzer:

1. In SoC Designer, select **File → Preferences**.
2. In the **Preferences** dialog, select **Profile**.
3. Enable the checkbox **Use external profile database** (see Figure 2-1). External profiling indicates System Analyzer.

4. Click **OK & Save**.

5. In SoC Designer Simulator, configure Monitor probes on the connections you are interested in. Refer to the **SoC Designer User Guide** if you need instructions.

6. Run the simulation to begin writing profiling data to the .profile file.

Proceed to the next section, **Launching the System Analyzer GUI and Viewing Profiled Data**, for information about .profile files and instructions for opening them with System Analyzer.

![Figure 2-1 Enabling System Analyzer Profiling](image)
2.1.3 Launching the System Analyzer GUI and Viewing Profiled Data

After you have run your simulation with the desired monitors enabled, launch System Analyzer to view the results using the information in this section.

If you run SoC Designer simulations with System Analyzer open, System Analyzer refreshes its data automatically when simulation stops. If you insert a Monitor window, display a CAPI stream, or display a Software view in SoC Designer Plus, the corresponding view is opened automatically in System Analyzer.

This section includes the following subsections:

- .Profile Name and Location on page 2-4
- Launch Instructions on page 2-5
- Command line arguments on page 2-5
- Opening a profile from within System Analyzer on page 2-6

.Profile Name and Location

When you run your simulation, all profiling data for monitored connections is captured to a profiling data file stored by default in your working directory on Linux. On Windows, the profiling data files is stored by default in the same directory as your .mxp file.

The profiling data file is named according to the convention <design_name>.profile. For subsequent simulations of the same design, a new profiling data file is created, and an incremented number is appended to the file name (for example, A9-MP1-PL301.profile.1).
Launch Instructions

To launch System Analyzer from:

- **SoC Designer** — In SoC Designer Simulator, click the **Analyzer** icon (see Figure 2-2). If the icon is grayed out, ensure that you have enabled **Use external profile database** as described in Enabling System Analyzer and Generating Data.

![System Analyzer icon in SoC Designer Simulator toolbar](image)

**Figure 2-2 System Analyzer icon in SoC Designer Simulator toolbar**

System Analyzer launches the current profile database for the loaded simulation.

- **Linux command line** — Enter `systemanalyzer -d <filename> .profile` (with a known filename), or `$MAXSIM_HOME/bin/systemanalyzer` (with no filename specified).

Refer to **Command line arguments** on page 2-5 for a complete list of arguments.

- **Windows desktop** — From the Windows Start menu, launch System Analyzer from the ARM menu.

**Command line arguments**

The command line argument to load a specific database is `-d <database-filename>`. Use this argument in combination with the following arguments to load the required database, data, and data views:

- `-t <transaction-connection-name>`
- `-c <capi-connection-name>`
- `-s <stream-name>`
- `-sw <software-name>`
- `-tile <horizontal | vertical> | <cascade>`

For example:

- `systemanalyzer <database-filename> -d -t` loads the specified transaction connection data in the specified database.
- `systemanalyzer <database-filename> -d -c -s` loads the stream data for the specified connection in the specified database.
• `systemanalyzer <database-filename> -d sw` loads the specified software data for the specified database.

• `systemanalyzer <database-filename> -d t` loads the specified transaction data for the specified database.

**Opening a profile from within System Analyzer**

To open a `.profile` file from within the System Analyzer GUI, select **File → Open** or select from the list of recently-opened files (see Figure 2-3).

![Figure 2-3 Opening a System Analyzer .profile file](image)
2.2 System Analyzer GUI Reference

The System Analyzer GUI (Figure 2-4) has three main sections:

- Plugins Panel — Displays buttons for launching plugins that you develop; the System Metrics plugin comes with SoC Designer and is available by default. Refer to Using the System Metrics Plugin for instructions on using this prebuilt plugin. Refer to Appendix A for more about creating your own plugins.
- Solutions Explorer — Described in Overview of the Solutions Explorer.
- Main Display Window — Described in Overview of the Main Display Window.
Using System Analyzer

Figure 2-4 System Analyzer GUI
2.2.1 Overview of the Solutions Explorer

The Solutions Explorer is a standard, hierarchical expansion tree. For each simulation file, it displays three kinds of Profiling data (as applicable). Nested below each of these data types are its measurable components:

- **Streams** — Hardware profiling events. The events supported by each stream differ based on the processor in use; refer to your processor’s Cycle Model Guide or Technical Reference Manual for details about its supported hardware profiling. See Working in the Streams View.

- **Transactions** — Data associated with transactions on a connection, including Program Counter and Function information. The data varies depending on the protocol in use. See Working in the Transactions View.

- **Software** — On a per-core basis, data such as program flow, total calls, and function duration. See Working in the Software View.

You can toggle the Solutions Explorer panel using **View → Solutions Explorer**.

2.2.2 Overview of the Main Display Window

The System Analyzer main display window, and the main toolbar functions, are context-sensitive based on the object selected in the Solutions Explorer tree.

Toolbar functions that persist regardless of context are shown in Figure 2-5.

![Persistent Toolbar Functions](image)

**Figure 2-5 Persistent Toolbar Functions**
The following buttons are available regardless of context:

- **New Tab** button — Creates a new tab. When you select a node in the Solutions Explorer tree, its data populates the new tab rather than replacing the data in the existing tab.
  
  You can rename any tab. Right-click on the tab and select **Rename** from the context menu.

- **Close Tab** button — Closes the active tab.

- **Float** button — Makes the window independent of the System Analyzer application.

- **Dock** button — Docks the application in the main System Analyzer display window.
  
  Standard Tile and Cascade functionality are available under the Windows menu.
2.3 Viewing Profiling Data

This section describes:

- Working in the Streams View
- Working in the Transactions View
- Working in the Software View

2.3.1 Working in the Streams View

Figure 2-6 shows the Streams view in the Solutions Explorer expanded to show the event names for the A9-MP1-PL301.CortexA9MP stream. Buckets that contain no data from the simulation are grayed out. The number of events of each particular type is shown in parentheses beside the event type.

![Figure 2-6 Streams view](image-url)
This section describes:

- Specifying Plot Characteristics
- Using the Streams View Toolbar
- Specifying Event Types
- Streams View Displays

**Specifying Plot Characteristics**

When you click on a stream in the tree, the Configure Plot dialog appears (Figure 2-7), which allows you to fine-tune the view. The settings for which you are prompted depend on the selected stream, and may include:

- X-Axis and Y-Axis settings — Specify the type of data to plot.
- Series — Select the Series to view.
- Starting Cycle and Ending Cycle — Specify the cycle range to observe; by default, the entire set is defined. Be aware that selecting too large a range may affect rendering performance; this is also dependent on the amount of available memory and the quantity of data to be rendered.

![Configure Plot Dialog](image)

**Figure 2-7 Configuring Plot Characteristics**

When you click **OK**, System Analyzer launches a floating window to display the data.

For optimum performance, System Analyzer calculates an appropriate default segment size on a per-simulation basis, based on the number of cycles.
When you hover over a section of the display, the event type is displayed. In the lower right corner, System Analyzer displays the Cycle, Program Counter, and Function associated with the event.

**Using the Streams View Toolbar**

The toolbar for the Streams view allows you to perform the following functions:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Zoom in and out" /></td>
<td>Zoom out and Zoom in along the Y Axis. Increases the vertical granularity of the view without changing the lateral granularity.</td>
</tr>
<tr>
<td><img src="image" alt="Zoom in and out" /></td>
<td>Zoom out and Zoom in along the X Axis . Increases the lateral granularity of the view without changing the vertical granularity.</td>
</tr>
<tr>
<td><img src="image" alt="Fit to window" /></td>
<td>Fit to window. Fits all available data into the view.</td>
</tr>
<tr>
<td><img src="image" alt="Zoom to cycle range" /></td>
<td>Zoom to Cycle Range. Constrains the view to a particular cycle range. Clicking this button prompts you to enter the Start and End Cycles for the range you wish to view, and the desired granularity (Segment Size). This is the method that System Analyzer provides for viewing data at a granularity of 1 cycle.</td>
</tr>
<tr>
<td><img src="image" alt="Cursor" /></td>
<td>Cursor. Adds a cursor to the view. Dragging the cursor across the chart displays the function executing during each cycle.</td>
</tr>
</tbody>
</table>
Specifying Event Types

By default, the Streams view shows all event types in its displays, which include the Stacked Bar Chart described below, as well as the Scatter Plot view described in Streams View Displays.

In the Stacked Bar Chart display, the Event Type panel (see Figure 2-8) lists the:

- Event types
- Minimum and maximum number of each event type within the specified range
- Total for each event type within the specified range

To show only certain event types in the display:

In the Event Type panel, check only the event types you want to view and Click Apply. System Analyzer redraws the display according to your specifications.
Streams View Displays

Depending on the type of profiling node selected, the Streams view displays two kinds of charts:

- Stacked Bar Chart. Figure 2-8, above, shows the Stacked Bar Chart that System Analyzer displays for the CPU_0 DCache stream.

- Scatter Plot. This format is used to display latency values, with Latency along the Y axis and Cycle number along the X axis (see Figure 2-9). The minimum and maximum number of each event type within the specified range is displayed in the Event Type panel.
Figure 2-9 Latency Scatter Plot

Event Type panel
2.3.2 Working in the Transactions View

This section describes:

- Transactions View Overview
- Constraining, Focusing, and Filtering the View
- Viewing Data Beats
- Displaying Software Trace Data

Transactions View Overview

When you click a Transactions-related node, System Analyzer displays data associated with particular transactions. Note that the data displayed varies depending on the protocol in use. Figure 2-10 shows a transaction view for CHI data.
The number of loaded transactions versus total transactions is displayed in the bottom left of the display window. The number of cycles is displayed in the bottom right corner of the display window.

**Figure 2-10 Transaction View**
Constraining, Focusing, and Filtering the View

Using the options in the toolbar, you can constrain or focus the view:

- **Start Cycle** — Specify the view to begin at a particular cycle. By default, the view displays transactions starting at cycle 0.

- **End Cycle** — Specify the view to end at a particular cycle. By default, the view displays transactions through the last transaction in the simulation.

- **Start Address** — View transactions associated with a particular start address.

- **End Address** — View transactions associated with a particular end address.

- **Command** menu — Allows you to filter the display by transaction type; by default, all transaction types are included.

For AXI, the menu options are Read, Write, Snoop, and Ignore.

For CHI, the menu options are Read, Dataless, Write, and Other. Refer to the AMBA 5 specification for information about the protocol transactions included in each category.

- **More/Less** button (CHI only) — Allows you to expand the view to include additional columns as needed, or restrict the view.

Viewing Data Beats

Expanding the view for a particular Transaction ID (Figure 2-11) displays its associated:

- Transaction Beat data (for AXI content)

- Flit data (for CHI)
Displaying Software Trace Data

In the Transactions view, you have the option of viewing the Program Counter and function information associated with each transaction. This is done by means of the Trace Software button on the toolbar (see Figure 2-12).
Figure 2-12 Software Trace Enabled
2.3.3 Working in the Software View

When you select a Software node in the tree, System Analyzer displays the details of the Software flow for the selected core. The display includes two tabs:

- **Software Flow Tab**
- **Software Summary Tab**

**Software Flow Tab**

The Software Flow tab displays a chart of the program flow by function name and cycle (see Figure 2-13).
Using the Cursor to Display the Executing Function

To display the name of the executing function at different cycles, enable the cursor using the Cursor button ( ) and drag it along the Software Flow.

Software Summary Tab

The Software Summary tab is a table format that shows the activity, such as total calls and function duration, related to each software function (see Figure 2-14),

![Software Flow Summary Table](image-url)

Figure 2-14 Software Summary Tab
2.4 Comparing Multiple Simulations

System Analyzer supports opening multiple simulations at once to compare performance.

To open additional .profile files, in System Analyzer select File → Open. The simulation profiling data file appears in the Solutions Explorer as a top-level node with a .profile suffix.

2.4.1 Locking Scroll for All Windows

When multiple windows are open, the Lock Scroll button synchronizes all views as you scroll in one window.

To enable Lock Scroll functionality:

1. Tile the windows you want to compare (Window → Tile).
2. Click the Lock Scroll button.

Disable scroll lock in the same window in which you enabled it.
2.5 Using the System Metrics Plugin

ARM provides a prebuilt plugin that returns a System Metrics report.

The System Metrics plugin is designed to run with SoC Designer Version 9.0.0 and later; these releases include version 3 or version 4 of the System Analyzer database. If you run the System Metrics plugin on data generated with an older version of SoC Designer Simulator, the following error message appears at the top of the report:

Warning .profile database version less than 3, results may be bad.

If this occurs, run a fresh simulation to update the .profile (database). Refer to the SoC Designer User Guide (ARM DUI 0956) for information about running simulations.

To use the System Metrics plugin:

1. In System Analyzer, open the .profile database to be analyzed.
2. Click the System Metrics icon at the upper left corner of the GUI (Figure 2-16).

![Figure 2-16 System Metrics Button in System Analyzer](image)

The System Metrics dialog opens (Figure 2-17).
Figure 2-17 System Metrics Stream Selection dialog

1. Select the streams you want to include in your report. If you select the top-level database (.profile), all subordinate data streams will be included.

2. By default, the Starting Cycle and Ending Cycle fields reflect the entire data set. Edit these fields to specify a more limited data set, if desired. Note that execution speed is dependent on the size of your data set.

3. Click OK. System Analyzer generates the report.

When finished, System Analyzer presents the report in a new tab.

A single report may include different sections for each protocol in use (if applicable). Refer to Appendix B for a description of the metrics presented. Report data may also differ based on the processor in use; refer to your processor’s Cycle Model Guide or Technical Reference Manual for details about profiled events.
Reports are also saved in a log file in the directory from which you launched System Analyzer. It is called `<design_name>.log`, where `<design_name>` is the prefix of the `.profile` database selected. For example if you selected A15-MP4.profile, the report can be found in A15-MP4.log.

A `.csv` file named `<design_name>.csv` is also generated. This `.csv` file contains summary data from a simulation run, not the calculated results. You can use this file to load the results into a spreadsheet application.

### 2.5.1 Sample Reports

Figure 2-18 shows a system containing an AXIv2 bus and Figure 2-19 shows one with a CHI bus.
If you specified a Starting Cycle and/or Ending Cycle other than the default (complete data set), the number of cycles detailed in the report is reflected as TOTAL SPECIFIED START/END RANGES. The number of cycles of the complete data set is given as TOTAL CYCLES.

---

**Figure 2-18 System Metrics Report (AXIv2)**

<table>
<thead>
<tr>
<th>Interconnect</th>
<th>CortexA9MP::axi_n0&lt;--&gt;L2C310::axi_s0</th>
</tr>
</thead>
<tbody>
<tr>
<td>THROUGHPUT (bytes/cycle)</td>
<td>Min</td>
</tr>
<tr>
<td>AXIv2 Read Throughput</td>
<td>4</td>
</tr>
<tr>
<td>AXIv2 Write Throughput</td>
<td>8</td>
</tr>
<tr>
<td>UTILIZATION (%)</td>
<td></td>
</tr>
<tr>
<td>AXIv2 Read Bus Utilization</td>
<td>15.11%</td>
</tr>
<tr>
<td>AXIv2 Write Bus Utilization</td>
<td>7.99%</td>
</tr>
<tr>
<td>AXIv2 Read Channel Utilization</td>
<td>6.67%</td>
</tr>
<tr>
<td>AXIv2 Write Channel Utilization</td>
<td>3.45%</td>
</tr>
<tr>
<td>TRANSACTIONS</td>
<td></td>
</tr>
<tr>
<td>Read Transactions</td>
<td>392</td>
</tr>
<tr>
<td>Write Transactions</td>
<td>215</td>
</tr>
<tr>
<td>LATENCY (latency/transaction cycles)</td>
<td>Min</td>
</tr>
<tr>
<td>AXIv2 Read-Trans (Addr) Latency</td>
<td>1</td>
</tr>
<tr>
<td>AXIv2 Write-Trans (Addr) Latency</td>
<td>1</td>
</tr>
<tr>
<td>AXIv2 Initial Read Latency</td>
<td>5</td>
</tr>
<tr>
<td>AXIv2 Initial Write Latency</td>
<td>1</td>
</tr>
<tr>
<td>AXIv2 Subsequent Read Latency</td>
<td>1</td>
</tr>
<tr>
<td>AXIv2 Subsequent Write Latency</td>
<td>---</td>
</tr>
<tr>
<td>AXIv2 Read Burst Latency</td>
<td>5</td>
</tr>
<tr>
<td>AXIv2 Write Burst Latency</td>
<td>1</td>
</tr>
</tbody>
</table>
### System Metrics Report (CHI)

#### CYCLES
- System: CH75AXI_sub256

#### EXT MEMORY & INTERCONNECT

<table>
<thead>
<tr>
<th>Interconnect</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHI Read Throughput</td>
<td>864.3333</td>
</tr>
<tr>
<td>CHI Write Throughput</td>
<td>864.6667</td>
</tr>
<tr>
<td>CHI SNF Throughput</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

#### THROUGHPUT (bytes/cycle)

<table>
<thead>
<tr>
<th>Throughput</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHI Read Throughput</td>
<td>8</td>
<td>64</td>
<td>26.3333</td>
</tr>
<tr>
<td>CHI Write Throughput</td>
<td>8</td>
<td>64</td>
<td>46.6667</td>
</tr>
<tr>
<td>CHI SNF Throughput</td>
<td>0</td>
<td>0</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

#### UTILIZATION (%)

<table>
<thead>
<tr>
<th>Utilization</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHI Read Bus Utilization</td>
<td>11.50%</td>
</tr>
<tr>
<td>CHI Write Bus Utilization</td>
<td>41.50%</td>
</tr>
<tr>
<td>CHI SNF Bus Utilization</td>
<td>6.00%</td>
</tr>
</tbody>
</table>

#### Channel Utilization

<table>
<thead>
<tr>
<th>Channel Utilization</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Utilization RXDAT</td>
<td>2.00%</td>
</tr>
<tr>
<td>Channel Utilization TXDAT</td>
<td>5.00%</td>
</tr>
<tr>
<td>Channel Utilization RXMP</td>
<td>6.00%</td>
</tr>
<tr>
<td>Channel Utilization TXMP</td>
<td>6.00%</td>
</tr>
<tr>
<td>Channel Utilization RXQ</td>
<td>4.50%</td>
</tr>
<tr>
<td>Channel Utilization TXQ</td>
<td>6.00%</td>
</tr>
<tr>
<td>Channel Utilization RXSNP</td>
<td>6.00%</td>
</tr>
<tr>
<td>Channel Utilization TXSNP</td>
<td>6.00%</td>
</tr>
</tbody>
</table>

#### LATENCY (latency/total_transactions)

<table>
<thead>
<tr>
<th>Latency</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHI Initial Read Latency</td>
<td>5</td>
</tr>
<tr>
<td>CHI Initial Write Latency</td>
<td>5</td>
</tr>
<tr>
<td>CHI Subsequent Read Latency</td>
<td>5</td>
</tr>
<tr>
<td>CHI Subsequent Write Latency</td>
<td>5</td>
</tr>
<tr>
<td>CHI Read Burst Latency</td>
<td>1</td>
</tr>
<tr>
<td>CHI Write Burst Latency</td>
<td>11</td>
</tr>
<tr>
<td>CHI Read Transactions Latency</td>
<td>5</td>
</tr>
<tr>
<td>CHI Write Transactions Latency</td>
<td>9</td>
</tr>
</tbody>
</table>

#### EFFICIENCY

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXDAT Channel Efficiency</td>
<td>0.0104</td>
</tr>
<tr>
<td>RXDAT Channel Efficiency</td>
<td>0.1196</td>
</tr>
<tr>
<td>SNF Channel Efficiency</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

#### INTERLEASE

<table>
<thead>
<tr>
<th>Interleave</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ (R/W) Channel Interleave Count</td>
<td>53</td>
</tr>
<tr>
<td>SNF Channel Interleave Count</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
Appendix A
Developing Plugins

Plugins allow you to perform specialized analysis on profiling data. This chapter provides instructions for creating a plugin assembly to work with System Analyzer:

- Before You Begin
- Creating a Plugin Assembly
- Enabling Plugins to Run in System Analyzer
- Enabling Plugins to Run as a Console Application
- Driving the Program from a Test Driver
- Plugin Method Reference
A.1 Before You Begin

This section provides some recommendations to help you get started with writing your own plugins.

A.1.1 Reviewing the .profile file

During simulation, System Analyzer outputs profiling data to a file called `<design_name>.profile` (for example, A9-MP1-PL301.profile.1). This file contains the raw data that your plugin will use. Before you can design a useful plugin, use a database browser (for example, SQLite) to take a look at the .profile file and become familiar with the data it collects as well as its structure.

See “Enabling System Analyzer Profiling” on page 2--2 for more about the .profile file.

A.1.2 Reviewing the System Analyzer API

You will be using the System Analyzer API exclusively to create your plugin. The System Analyzer API Reference Guide provides a complete description of the functions available. This document (SoCDesigner_Tools_API_Reference_Manual.pdf) is available in the /doc directory of your SoC Designer Plus installation.

A.1.3 Modifying an Example

System Analyzer includes example plugins that you can walk through for a better understanding of their structure and functionality. You can copy one of these examples to use as a basis for your own customized plugin.

See “Understanding the Data Structures” on page A--3 for more information about available examples.
A.2 Creating a Plugin Assembly

This section describes:

- Understanding the Data Structures
- Plugin Coding Requirements
- Creating the Project (includes Visual Studio, MonoDevelop, and IDE-independent instructions)

A.2.1 Understanding the Data Structures

For your convenience, example plugins are included with System Analyzer. Walking through these examples will help you understand the underlying data structures with which System Analyzer plugins interact.

The example plugins are located in your SoC Designer installation area under examples/Analyzer/plugin_examples:

- /CPU — Contains plugin files designed to iterate through processor data.
- /Transactions — Contains plugin files designed to iterate through interconnect transaction data.
- /Validate_Metrics — Contains the source code and build files for the pre-built System Metrics plugin that ships with System Analyzer. ARM provides the source so you can customize the System Metrics report. Refer to Using the System Metrics Plugin for information about using this plugin, and Running ValidateMetrics with the CHI Example for more information about using the ValidateMetrics script.
- Refer to the Readme.txt file in each directory for instructions on building these examples, and to the information in this appendix for information about modifying them to suit your needs.
CHI Methods

In addition to the methods that apply to CHI connections that exist in the System Analyzer API, the following additional, higher-level methods facilitate CHI metric computation:

- UniqueTransactionsCount() — How many unique transactions are on a particular CHI connection.
- FlitsOnChannelCount(Int64 channel) — Returns the number of flits on a particular channel.
- Ienumerator<ICHITransactionFlit> DataItems() — The collection of CHITransactionFlits that happen on the data channel for a particular transaction.
- Int64 Max/MinFlitCycle() — Maximum/Minimum Flit Cycle on a particular Transaction.
- Int64 FirstDATFlitCycle() — The first DATA flit cycle for this transaction. (returns 0 if none).
- Int64 LastDATFlitCycle() — The last DATA flit cycle for this transaction. (returns 0 if none).

A.2.2 Plugin Coding Requirements

This section describes some basic coding requirements for your Plugin program, regardless of programming platform. The section “Creating the Project” on page A--4 includes sample code for your reference.

- Your plugin must add a reference to $MAXSIM_HOME/lib/Linux/release/CarbonAutoEnums.dll and CarbonAnalyzerAPI.dll.
- Your assembly must contain at least one class which derives from Carbon.Analyzer.Plugins.ICarbonAnalyzerPlugin.
- If you are using an IDE, you must specify .NET 4.0-compatible output.

A.2.3 Creating the Project

This section describes how to create your Plugin project in different environments:

- Visual Studio Instructions
- MonoDevelop Instructions
- Standalone Instructions
Visual Studio Instructions

This section describes how to create a plugin assembly using C# in Visual Studio:

1. Open Visual Studio and create a new Console project. Refer to Section A.3 for information about the best location for your plugin and making it accessible for use by the System Analyzer GUI.

2. Access the Application options and set the target framework to .NET 4.0 (see Figure A-1).

3. Add references to the files CarbonAnalyzerAPI.dll and CarbonAutoEnums.dll (Figure A-2). These are located in either $MAXSIM_HOME/lib/Linux/release or $MAXSIM_HOME/lib/Win32/Release.
4. Add using statements for the following:

   using CarbonAnalyzerAPI;
   using CarbonAnalyzerAPI.Interfaces;
   using CarbonAnalyzer;

5. Ensure your plugin assembly contains at least one class which derives from Carbon.Analyzer.Plugins.ICarbonAnalyzerPlugin; for example:

   public plugin:Carbon.Analyzer.Plugins.ICarbonAnalyzerPlugin


7. Right click on ICarbonAnalyzerPlugin and implement an interface. This implements interfaces for the methods Accepts, Analyze, Description and Name.

8. Populate the interfaces with the functionality you want.

9. Build the project.
The following is an example of a simple Plugin program:

```csharp
public string Description
{
    get { return "Plugin1 Description Here"; }
}

public string Name
{
    get { return "Plugin1"; }
}

{
    get { return Carbon.Analyzer.Plugins.AcceptableSourceTypes.Transaction; }
}

public string Analyze(Carbon.Analyzer.Plugins.AnalyzerArgument[] arguments)
{
    {
        var ds = Analyzer.Open(arg.DatabaseFileName);
        var ts = ds.TransactionStreams[arg.SourceName];
        if (ts == null)
        {
            Console.WriteLine(string.Format("Error: No source found: \{0\}", arg.SourceName));
            return "";
        }
        Int64 maxLatency = 0;
        Int64 maxReadLatency = 0;
        Int64 maxWriteLatency = 0;
        Int64 numReads = 0;
```
Int64 numWrites = 0;

foreach ( IAXIv2Transaction tx in ts.AXIv2Transactions.Items( CycleRange .All))
{
    foreach (IAXIv2TransactionBeat b in tx.TransactionsBeats.Items())
    {
        if (b.DataBeat == 0)
        {
            Int64 latency = b.BeatComplete - tx.AddressStart;
            {
                {
                    latency = tx.ResponseComplete - tx.AddressStart;
                    if (latency > maxWriteLatency)
                    {
                        numWrites++;
                    }
                }
                if (latex > maxReadLatency)
                {
                    maxReadLatency = latency;
                    numReads++;
                }
            }
            {
                latency = tx.ResponseComplete - tx.AddressStart;
                if (latency > maxWriteLatency)
                {
                    numWrites++;
                }
            }
        }
    }
}

// CHI section. Traversal through streams, transactions and transaction flits
foreach (ITransactionStream ts2 in ds.TransactionStreams)
{

    // Counts total Transactions and total Transaction Flits.
    int countTX = 0;
    int countTXFLITS = 0;

    foreach ( ICHITransaction ctx in ts2.CHITransactions.Items( CycleRange .All))
    {
        countTX++;  

        foreach ( ICHITransactionFlit tf in ctx.TransactionsFlits.Items())
        {
            countTXFLITS++;
        }
    }

    return "";
}
MonoDevelop Instructions

This section describes how to create a plugin assembly using MonoDevelop on Linux.

--- Note ---

Before beginning, set the `LD_LIBRARY_PATH` environment variable as follows:

```
LD_LIBRARY_PATH=$MAXSIM_HOME/lib/Linux/release:$MAXSIM_HOME/deps/gcc/4.7.2-binutil-2.25/lib64
```

To create the project in MonoDevelop:

1. Open MonoDevelop and create a new Console project. Refer to Section A.3 for information about the best location for your plugin and making it accessible for use by the System Analyzer GUI.

2. Access the project Options and on the Build panel, set the Target Framework to Mono/.NET 4.0 (Figure A-3):
3. Add references to CarbonAnalyzerAPI.dll and CarbonAutoEnums.dll (Figure A-4):

4. Include the following additional using statements:

   using CarbonAnalyzerAPI;

   using CarbonAnalyzerAPI.Interfaces;

   using Carbon.Analyzer;

5. To create a plugin, your assembly must contain at least one class which derives from Carbon.Analyzer.Plugins.ICarbonAnalyzerPlugin.


7. Populate the interfaces with the functionality you want.

8. Build the project.
The following is an example of a simple Plugin program:

```csharp
using System;

using CarbonAnalyzerAPI;
using CarbonAnalyzerAPI.Interfaces;
using Carbon.Analyzer;

namespace Plugin2
{
    class MyPlugin : Carbon.Analyzer.Plugins.ICarbonAnalyzerPlugin
    {
        {
            {
                Analyzer a = new Analyzer(arg.DatabaseFileName);
                // Perform Metrics Here
            }
            return "<html><body>test results</body></html>";
        }

        string Carbon.Analyzer.Plugins.ICarbonAnalyzerPlugin.Name {
            get {
                return "My Plugin";
            }
        }

        string Carbon.Analyzer.Plugins.ICarbonAnalyzerPlugin.Description {
```
```
Standalone Instructions

For IDE-independent plugin development, you can use either:

- an ARM-supplied wrapper script (see Enabling Plugins to Run as a Console Application)
- `dmcs`, a CSharp compiler/linker that runs on Linux) for compilation (instructions below).

**Using dmcs**

On Linux, if you are developing your plugin independent of a particular IDE, use the instructions in this section to compile the program using `dmcs`.

1. Set the `LD_LIBRARY_PATH` environment variable to
   
   ```
   LD_LIBRARY_PATH=$MAXSIM_HOME/lib/Linux/release:$MAXSIM_HOME/deps/gcc/4.7.2-binutil-2.25/lib64.
   ```

2. `dmcs` must be set up to use the version of Mono that is shipped with the SoC Designer Plus distribution. To do so, add the following to your PATH:
   ```
   $MAXSIM_HOME/etc/mono/bin.
   ```

3. When your program is ready to compile, you must include references to `CarbonAnalyzerAPI.dll` and `CarbonAutoEnums.dll` on the command line.

```csharp
get {
    return "Description";
}
}
}

    get {
        return Carbon.Analyzer.Plugins.AcceptableSourceTypes.All;
    }
}
```
The following example builds **Main.cs** with the `dmcs` compiler and includes the necessary references:

```
> dmcs Main.cs -reference:/x/work/build/master/package/SoCD/lib/Linux/release/CarbonAnalyzerAPI.dll
   -reference:/x/work/build/master/package/SoCD/lib/Linux/release/CarbonAutoEnums.dll
```

Refer to the `dmcs` documentation for additional help
(http://www.mono-project.com/CSharp_Compiler).

--- **Note** ---

If you are using C# on Windows, there is a C# requirement that all the resources be copied with the plugin executable. For example, to copy a plugin to `/fred/bin`, copy `SystemMetrics.exe` as well as `CarbonAnalyzerAPI.dll` and `CarbonAutoEnums.dll` to `/fred/bin`. In general, copy all the files that Visual Studio includes in the project's `bin/Debug` or `bin/Release` directory.

---


A.3 Enabling Plugins to Run in System Analyzer

There are two ways to make a plugin accessible for use by System Analyzer:

- After compiling the project, copy the built .exe or .dll into the directory $MAXSIM_HOME/etc/plugins/analyzer.

- Set the environment variable CARBON_ANALYZER_PLUGINS and define its value as the directory containing the additional plugins. Doing this allows System Analyzer to identify any .dll or .exe stored in that directory as a plugin.

To include more than one plugin directory in the CARBON_ANALYZER_PLUGINS environment variable, separate each directory using a colon (:) on Linux and semicolon (;) on Windows.

Note that all *.dll and *.exe plugins located in the directory will be loaded.
A.4 Enabling Plugins to Run as a Console Application

To run a plugin in a console application outside of System Analyzer, use one of the following approaches:

- Wrapper Script approach
- Environment Variable approach

A.4.1 Wrapper Script approach

The Carbon-provided wrapper script `systemmetrics.bat` is available in `$MAXSIM_HOME/bin` (Linux) or `%MAXSIM_HOME%\bin` (Windows). This script is for the Carbon-provided System Metrics plugin; copy the file and modify it for custom use.

A.4.2 Environment Variable approach

Set the environment variable MONO_PATH to the location of the library files `CarbonAnalyzerAPI.dll` and `CarbonAutoEnums.dll`. This is usually `$MAXSIM_HOME/lib/Linux/release`.

For example:

- In csh (or ksh) — `setenv MONO_PATH $MAXSIM_HOME/lib/Linux/release`
- In bash (or sh) — `MONO_PATH=$MAXSIM_HOME/lib/Linux/release; export MONO_PATH`

If your application requires other libraries, they should be included as well. The process is similar to setting LD_LIBRARY_PATH (see Standalone Instructions).
A.5 Driving the Program from a Test Driver

To drive your plugin program from a test driver, use the instructions in this section. Visual Studio and MonoDevelop instructions are provided below.

A.5.1 Visual Studio Instructions

1. Inside Visual Studio, select the top level solution in the Solution Explorer.
2. Add a New Console Project Application and give it a name similar to PluginLibraryTest.
3. Set this subproject to be your startup project.
4. Add a reference to the classLibrary that you just created.
5. Also add a reference to CarbonAnalyzerAPI.dll and CarbonAutoEnums.dll.
6. Add an existing item to the project: sqlite3.dll.
7. Select sqlite3.dll and choose Properties. Indicate that you will "copy if newer" to your exe directory. This ensures that that dll is in the directory where your Console Application is built.
8. Double click on Program.cs to open it.
9. Add a Using statement to refer to your PluginLibrary (e.g. using classLibrary1).
10. Inside your Main program, create a new instance of the class that was used in the original assembly above. For example:

```csharp
namespace PluginLibraryTest
{
    static void Main(string[] args) {
        Plugin1 p1 = new Plugin1();

        arg.DatabaseFileName = @"A9.profile";
        Console.WriteLine(html);
    }
}
```
Developing Plugins

return;
}
}
}

11. If running on Linux, set your LD_LIBRARY_PATH as follows:

   LD_LIBRARY_PATH=$MAXSIM_HOME/lib/Linux/release:$MAXSIM_HOME/deps/gcc/4.7.2-binutil-2.25/lib64.

Now you are able to debug your assembly outside of the System Analyzer program.

A.5.2 MonoDevelop Instructions

Before beginning, ensure that your LD_LIBRARY_PATH is set as follows:
LD_LIBRARY_PATH=$MAXSIM_HOME/lib/Linux/release:$MAXSIM_HOME/deps/gcc/4.7.2-binutil-2.25/lib64.

1. In MonoDevelop, start a new solution and give it a name (e.g., NewConsoleApp).
3. In your .cs file, add a Using statement to refer to your Plugin.
4. Inside your Main program, create a new instance of your plugin class and specify any arguments.
5. Call your plugin’s Analyze function.

Following is an example of a simple test driver:

```csharp
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Plugin2;
using System.IO;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Plugin1 c1 = new MyPlugin();
        
```
arg.DatabaseFileName = @"A9-MP1-TRAINING.profile";
Console.WriteLine(html);
return;
}
A.6 Plugin Method Reference

This section describes:

- Name and Description Methods
- Accepts Method
- Analyze Method

A.6.1 Name and Description Methods

The Name and Description methods indicate the plugin name that appears in System Analyzer and the description of the plugin. In Figure A-5, System Metrics 1.0.3 is the Name of the plugin.

![Figure A-5 Plugin Button in System Analyzer](image)

A.6.2 Accepts Method

The Accepts method indicates the kind of data that this plugin presents when the Plugin button is clicked in the System Analyzer GUI. The following are available options for the Accepts method: All, Transaction, Connection, Database, Profile, Software and Transaction.

For example, if Transactions are of interest, the Accepts method in the plugin code would specify Transactions for AcceptableSourceTypes; for example:

```plaintext
AcceptableSourceTypes = Transactions;
```

    {
        get { return Carbon.Analyzer.Plugins.AcceptableSourceTypes.Transaction; }
    }

When the plugin is implemented in the System Analyzer GUI, the dialog that displays when clicking the Plugin button appears as shown in Figure A-6. From this view, you can pick connections of interest related to analyzing Transactions.

These options are described below.

- **All** — Plugin dialog presents all data that was profiled in SoC Designer.
- **Connection** and **Transaction** — Plugin dialog presents data from profiled connections in SoC Designer. Refer to the SoC Designer User Guide for information on configuring Monitor and Profiler probes on system connections.
- **Database** — If not passed in, the plugin operates on all available databases. If passed in, the plugin allows selection of individual databases.
- **Profile** — Plugin presents data from profiled hardware streams. Figure A-7 shows hardware streams enabled for profiling in the SoC Designer Profiling Manager.
Software — Plugin presents choices related to software that was captured from SoC Designer. Figure A-8 shows a software stream (as well as hardware streams) enabled for profiling in the SoC Designer Profiling Manager.
A.6.3 Analyze Method

The Analyze method is used to produce and return the results of an API interrogation of the database. In the example below, this routine has produced information about maximum read and write latency.

The example in Figure A-9 iterates over the transactions and produces the maximum latency information, producing HTML output similar to that shown in Figure A-9:

Note that the code within the Analyze method can interrogate the database file name and sourcename (connection). It does this via the methods arg.DataBaseFileName and arg.SourceName.

A full description of all of the API routines can be found in the System Analyzer API Reference Guide.
A.7 **Running ValidateMetrics with the CHI Example**

ValidateMetrics.csh is a customizable test script that is shipped with System Analyzer, located in your SoC Designer installation area under examples/Analyzer/plugin_examples/ValidateMetrics. You can use it to compare CHI simulation results against expected results, or modify it for use with your own plugin (see Modifying ValidateMetrics to use with custom profiles).

ValidateMetrics compares the results of the Carbon CHI example shipped with System Analyzer (CHItoAXI_sub256.profile) with expected (gold) results. Refer to the comments within ValidateMetrics.csh for specific instructions.

A.7.1 **Modifying ValidateMetrics to use with custom profiles**

If you are creating your own plugin, you may want to modify the ValidateMetrics script by specifying the names of your own .profile and gold files within ValidateMetrics.csh. Refer to the comments in the code for instructions.
Appendix B
Metrics and Formulas

This appendix contains the following sections:

• AXI4 Metrics and Formulas on page B-2
• CHI Metrics and Formulas
# AXI4 Metrics and Formulas

Table B-1 describes AXI-related calculations and formulas you may wish to use to perform your analysis. This is a partial list.

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Formula</th>
</tr>
</thead>
</table>
| channel utilization       | Write channel<sup>a</sup> \[
\frac{\sum (D_{n\_End}-D_{1\_Start}+1)}{cycles}
\]
  where:  
  * \(D_{n\_End}\) — Last Data Beat, end cycle.  
  * \(D_{1\_Start}\) — First Data Beat, start cycle.  

| Calculate channel        | Read channel<sup>a</sup> \[
\frac{\sum (D_{x\_End}-D_{x\_Start}+1)}{cycles}
\]
  where:  
  * \(D_{x\_End}\) — First Data beat, start cycle.  
  * \(D_{x\_Start}\) — Last Data beat, start cycle.  
  * \(x\) holds the beat number.  

| bus utilization          | Write<sup>b</sup> \[
\frac{\text{(number of cycles a transaction is active (AW\_Start without Rsp\_End))} \times \text{BUS\_WIDTH}}{cycles}
\]
  where:  
  * \(AW\_Start\) — Write Address, start cycle.  
  * \(Rsp\_End\) — Write Response end (B Channel).  
  * This (response) is the final step for a write transaction.  

| bus utilization          | Read<sup>b</sup> \[
\frac{\text{(number of cycles a transaction is active (AR\_Start without Dn\_End))} \times \text{BUS\_WIDTH}}{cycles}
\]
  where:  
  * \(AR\_Start\) — \(x\) represents Read or Write. For example, for Reads it translates to AR\_Start.  
  * \(Dn\_End\) — Last Data Beat, end cycle.  

| channel throughput       | Read/Write channel \[
\frac{\sum (Size\times Beats)}{cycles}
\]

| channel efficiency       | Read/Write channel \[
\frac{\sum (Size\times Beats)}{cycles\times channel\_utilization\times BUS\_WIDTH}
\]
### Table B-1 AXI4 Formulas and Variable Definitions (continued)

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Formula</th>
</tr>
</thead>
</table>
| Write-trans (AW)     | \[
|                      | \( \frac{\sum (\text{AW}_\text{End}-\text{AW}_\text{Start}+1))}{\text{total\_transactions}} \) \]  
|                      | where: \text{AW}_\text{End} — Write Address, end cycle \text{AW}_\text{Start} — Write Address, start cycle |
| Write-initial        | \[
|                      | \( \frac{\sum (\text{D1}_\text{End}-\text{D1}_\text{Start}+1))}{\text{total\_transactions}} \) \]  
|                      | where: \text{D1}_\text{Start} — First Data Beat, start cycle \text{D1}_\text{End} — First Data Beat, end cycle |
| Write-subsequent\text{c} | \[
|                      | \( \frac{\sum (\text{Dx}_\text{End}-\text{Dx}_\text{Start}+1\_\text{End}))}{(\text{total\_beats} - \text{total\_transactions})} \) \]  
|                      | where: \text{Dx}_\text{End} — First Data beat, start cycle. \text{x} holds the beat number. \text{Dx}_\text{Start} — Last Data beat, start cycle. \text{x} holds the beat number. |
| Write-burst          | \[
|                      | \( \frac{\sum (\text{Dn}_\text{End}-\text{D1}_\text{Start}+1))}{\text{total\_transactions}} \) \]  
|                      | where: \text{Dn}_\text{End} — Last Data Beat, end cycle \text{D1}_\text{Start} — First Data Beat, start cycle |
| write transactions   | \[
|                      | \( \frac{\sum (\text{Rsp}_\text{End}-\text{AW}_\text{Start}))}{\text{total\_transactions}} \) \]  
<p>|                      | where: \text{Rsp}<em>\text{End} — Write Response end (B Channel). This (response) is the final step for a write transaction. \text{AW}</em>\text{Start} — Write Address, start cycle |</p>
<table>
<thead>
<tr>
<th>Calculation</th>
<th>Formula</th>
</tr>
</thead>
</table>
| Read-trans (AR)        | $(\sum \frac{(AR_{\text{End}}-AR_{\text{Start}}+1)}{\text{total\_transactions}})$  
  where:
  AR_{\text{End}} — Address Read channel, end cycle
  AR_{\text{Start}} — Address Read channel, start cycle |
| Read-initial           | $(\sum \frac{(D1_{\text{Start}}-AR_{\text{End}})}{\text{total\_transactions}})$  
  where:
  D1_{\text{Start}} — First Data Beat, start cycle
  AR_{\text{End}} — Address Read channel, end cycle |
| Read-subsequent        | read-subsequent $(\sum \frac{(Dx_{\text{Start}}-D[x-1]_{\text{Start}})}{(\text{total\_beats} - \text{total\_transactions})})$  
  where:
  Dx_{\text{Start}} — Last Data beat, start cycle.
  $x$ holds the beat number. |
| Read-burst             | $(\sum \frac{(Dn_{\text{End}}-AR_{\text{End}})}{\text{total\_transactions}})$  
  where:
  Dn_{\text{End}} — Last Data Beat, end cycle
  AR_{\text{End}} — Address Read channel, end cycle |
| Read-transactions      | $(\sum \frac{(Dn_{\text{End}}-AR_{\text{Start}})}{\text{total\_transactions}})$  
  where:
  Dn_{\text{End}} — Last Data Beat, end cycle
  AR_{\text{Start}} — Address Read channel, start cycle |
### Table B-1 AXI4 Formulas and Variable Definitions (continued)

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel back pressure</td>
<td>For a given cycle: number of Ax_Starts without Ax_Ends where: Ax_Start — x represents Read or Write. For example, for Reads it translates to AR_Start. Ax_End — x represents Read or Write. For example, for reads it translates to AR_End.</td>
</tr>
<tr>
<td>rd/wr</td>
<td>For a given cycle: number of beats not transferred for open transactions.</td>
</tr>
<tr>
<td>B</td>
<td>For a given cycle: number of RSP_Starts without RSP_Ends where: Rsp_Start — Write Response start (B Channel), Rsp_End — Write Response end (B Channel). This (response) is the final step for a write transaction.</td>
</tr>
<tr>
<td>channel interleave counts</td>
<td>Number of open transactions.</td>
</tr>
</tbody>
</table>

- Ignore AW, AR, and B channels; these are not interesting.
- Do not double count when transactions overlap.
- $\sum$ is for $x = 2^{-n}$; in other words, first beats ignored.
### B.2 CHI Metrics and Formulas

When used with CHI metrics, note that cycles = total cycles in simulation or total cycles in sample period.

#### Table B-2 CHI Formulas and Variable Definitions

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel utilization</td>
<td>all channels</td>
</tr>
<tr>
<td></td>
<td>number of flits on channel / cycles</td>
</tr>
<tr>
<td>bus utilization</td>
<td>write&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>( ( \sum ) (number of cycles a transaction REQ write is active (end flit cycle - begin flit cycle + 1)) ) / cycles</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>(count of cycles where open write transactions &gt; 0) / cycles</td>
</tr>
<tr>
<td></td>
<td>read&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>(count of cycles where open write transactions &gt; 0) / cycles</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>( ( \sum ) (number of cycles a transaction REQ read is active (end flit cycle - begin flit cycle + 1)) ) / cycles</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>(count of cycles where open read transactions &gt; 0) / cycles</td>
</tr>
<tr>
<td>SNP</td>
<td>( ( \sum ) (number of cycles a transaction SNP is active (end flit cycle - begin flit cycle + 1)) ) / cycles</td>
</tr>
<tr>
<td>bus throughput</td>
<td>write</td>
</tr>
<tr>
<td></td>
<td>( ( \sum ) TXREQ Size&lt;sup&gt;b&lt;/sup&gt;) / cycles * For the TXREQ opcodes that begin with &quot;Write&quot;</td>
</tr>
<tr>
<td></td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>( ( \sum ) TXREQ Size&lt;sup&gt;c&lt;/sup&gt;) / cycles * For the TXREQ opcodes that begin with &quot;Read&quot;</td>
</tr>
<tr>
<td>SNP</td>
<td>((Number of TXDAT&lt;sup&gt;d&lt;/sup&gt; Snoop response flits) * bus_width (bytes)) / cycles</td>
</tr>
</tbody>
</table>
### Table B-2 CHI Formulas and Variable Definitions (continued)

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel efficiency</td>
<td>TXDAT: $(\sum \frac{\text{TXREQ Size}}{\text{cycles} \times \text{bus utilization} \times (\text{BUS WIDTH}/8)})$</td>
</tr>
<tr>
<td></td>
<td>RXDAT: $(\sum \frac{\text{TXREQ Size}}{\text{cycles} \times \text{bus utilization} \times (\text{BUS WIDTH}/8)})$</td>
</tr>
<tr>
<td></td>
<td>SNP: $(\frac{\text{Number of TXDAT Snoop response flits} \times (\text{BUS WIDTH}/8)}{\text{cycles} \times \text{bus utilization} \times (\text{BUS WIDTH}/8)})$</td>
</tr>
<tr>
<td>latency</td>
<td>write: write-trans (AW) — N/A</td>
</tr>
<tr>
<td></td>
<td>write-initial — $(\sum \frac{(\text{first DAT flit cycle})-(\text{first flit cycle})+1)}{\text{total_transactions}}$</td>
</tr>
<tr>
<td></td>
<td>write-subsequent — $(\sum \frac{(\text{subsequent DAT flit cycle})-(\text{previous DAT flit cycle})+1)}{\text{total_transactions}}$</td>
</tr>
<tr>
<td></td>
<td>write-burst — $(\sum \frac{(\text{last DAT flit cycle})-(\text{first DAT flit cycle})+1)}{\text{total_transactions}}$</td>
</tr>
<tr>
<td></td>
<td>write-transactions — $(\sum \frac{(\text{last flit cycle})-(\text{first flit cycle})+1)}{\text{total_transactions}}$</td>
</tr>
<tr>
<td>read</td>
<td>read-trans (AR) — N/A</td>
</tr>
<tr>
<td></td>
<td>read-initial — $(\sum \frac{(\text{first DAT flit cycle})-(\text{first flit cycle})+1)}{\text{total_transactions}}$</td>
</tr>
<tr>
<td></td>
<td>read-subsequent — $(\sum \frac{(\text{subsequent DAT flit cycle})-(\text{previous DAT flit cycle})+1)}{\text{total_transactions}}$</td>
</tr>
<tr>
<td></td>
<td>read-burst — $(\sum \frac{(\text{last DAT flit cycle})-(\text{first DAT flit cycle})+1)}{\text{total_transactions}}$</td>
</tr>
<tr>
<td></td>
<td>read-transactions — $(\sum \frac{(\text{last flit cycle})-(\text{first flit cycle})+1)}{\text{total_transactions}}$</td>
</tr>
<tr>
<td>channel interleave counts</td>
<td>REQ: Average number of open transactions per cycle</td>
</tr>
<tr>
<td></td>
<td>SNP: Average number of open transactions per cycle</td>
</tr>
</tbody>
</table>

- a. Do not double count when transactions overlap.
- b. For the TXREQ opcodes that begin with "Write."
- c. For the TXREQ opcodes that begin with "Read."
- d. TXDAT snoop resp opcodes include SnpRespData and SnpRespDataPtl.
Appendix A
Revisions

This appendix describes the technical changes between released issues of this book.

<table>
<thead>
<tr>
<th>Change</th>
<th>Location</th>
<th>Affects</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Restamped release.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>