

SoC Designer

Version 9.6

System Analyzer User Guide



SoC Designer

System Analyzer User Guide

Copyright © 2018 Arm Limited (or its affiliates). All rights reserved.

Release Information

Document History

Issue	Date	Confidentiality	Change
0906-00	28 February 2018	Non-Confidential	Release with 9.6

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. **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 third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

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 click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. 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 of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2018 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

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.

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

<http://www.arm.com>

Contents

SoC Designer System Analyzer User Guide

Preface

About this book	7
-----------------------	---

Chapter 1

Overview of System Analyzer

1.1 Features and benefits	1-10
1.2 Requirements	1-11
1.3 Restrictions and limitations	1-12
1.4 Suggested reading	1-13

Chapter 2

Using System Analyzer

2.1 Enabling System Analyzer profiling	2-15
2.2 System Analyzer GUI reference	2-18
2.3 Viewing profiling data	2-20
2.4 Comparing Multiple Simulations	2-28
2.5 Using the System Metrics Plugin	2-29

Appendix A

Developing plugins

A.1 Before you begin	Appx-A-33
A.2 Creating a plugin assembly	Appx-A-34
A.3 Enabling plugins to run in System Analyzer	Appx-A-39
A.4 Enabling plugins to run as a console application	Appx-A-40
A.5 Driving the program from a test driver	Appx-A-41
A.6 Plugin method reference	Appx-A-42
A.7 Running ValidateMetrics with the CHI Example	Appx-A-45

Appendix B

Metrics and formulas

B.1

AXI4 metrics and formulas table

Appx-B-47

B.2

CHI metrics and formulas table

Appx-B-50

Appendix C

Revisions

C.1

Revisions

Appx-C-53

Preface

This preface introduces the *SoC Designer System Analyzer User Guide*.

It contains the following:

- [About this book on page 7.](#)

About this book

This book describes how to use the System Analyzer plugin for SoC Designer. This book is written for experienced hardware engineers, software engineers and System-on-Chip (SoC) designers who have experience of Arm® products.

Using this book

This book is organized into the following chapters:

Chapter 1 Overview of System Analyzer

This chapter provides an overview of the features and benefits, system requirements, and any restrictions and limitations that should be noted when working with System Analyzer.

Chapter 2 Using System Analyzer

This chapter describes how to enable System Analyzer profiling, and how to view profiling data and compare multiple simulations. It also describes the GUI and how to use the System Metrics Plugin, which produces a System Metrics report.

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.

Appendix B Metrics and formulas

This appendix provides a list of AXI-related calculations and formulas that can be used to perform analysis, and a list of CHI formulas and variable definitions that are used with CHI metrics.

Appendix C Revisions

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

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](#) for more information.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

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

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`monospace`

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

<and>

Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:

- The title *SoC Designer System Analyzer User Guide*.
- The number 101143_0906_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information

- *Arm® Developer*.
- *Arm® Information Center*.
- *Arm® Technical Support Knowledge Articles*.
- *Technical Support*.
- *Arm® Glossary*.

Chapter 1

Overview of System Analyzer

This chapter provides an overview of the features and benefits, system requirements, and any restrictions and limitations that should be noted when working with System Analyzer.

It contains the following sections:

- [*1.1 Features and benefits*](#) on page 1-10.
- [*1.2 Requirements*](#) on page 1-11.
- [*1.3 Restrictions and limitations*](#) on page 1-12.
- [*1.4 Suggested reading*](#) on page 1-13.

1.1 Features and benefits

System Analyzer is easy to setup, integrates with SoC Designer, and captures all stream, transaction, and software profiling data for entire simulations.

In addition, System Analyzer:

- 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.
- Supports 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.
- Includes a prebuilt Validation Metrics Plugin which computes useful metrics such as latency, throughput, channel utilization, and bus utilization.

Related references

[Appendix A Developing plugins on page Appx-A-32.](#)

1.2 Requirements

Review this section for information about software and platform requirements for System Analyzer.

This section contains the following subsections:

- [1.2.1 Software requirements on page 1-11.](#)
- [1.2.2 Platform requirements on page 1-11.](#)

1.2.1 Software requirements

System Analyzer requires the following software:

- SoC Designer Version 9.6.0 or later.
- You can develop System Analyzer plugins using a Windows IDE such as Visual Studio 2013.

Alternatively, you can write your plugin independent of an IDE and use your system's native compiler to compile it.

Related concepts

[1.2.2 Platform requirements on page 1-11.](#)

Related references

[Appendix A Developing plugins on page Appx-A-32.](#)

1.2.2 Platform requirements

System Analyzer is supported on both Linux and Windows platforms. Refer to the *SoC Designer Installation Guide* (100975) for supported versions.

Related concepts

[1.2.1 Software requirements on page 1-11.](#)

1.3 Restrictions and limitations

Restrictions and limitations that should be noted when working with System Analyzer.

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

1.4 Suggested reading

The following additional Arm publications are related to profiling and working with System Analyzer:

- *System Analyzer API Reference Guide* (DUI1036)
- *SoC Designer User Guide* (100996)
- *AMBA5 CHI Architecture Specification (12 June 2014, Rev. A)* (IHI0050)
- *AMBA AXI and ACE Protocol Specification* (IHI0022)

Chapter 2

Using System Analyzer

This chapter describes how to enable System Analyzer profiling, and how to view profiling data and compare multiple simulations. It also describes the GUI and how to use the System Metrics Plugin, which produces a System Metrics report.

It contains the following sections:

- [2.1 Enabling System Analyzer profiling](#) on page 2-15.
- [2.2 System Analyzer GUI reference](#) on page 2-18.
- [2.3 Viewing profiling data](#) on page 2-20.
- [2.4 Comparing Multiple Simulations](#) on page 2-28.
- [2.5 Using the System Metrics Plugin](#) on page 2-29.

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 contains the following subsections:

- [2.1.1 Important operational notes on page 2-15.](#)
- [2.1.2 Enabling System Analyzer and generating data on page 2-15.](#)
- [2.1.3 Launching the System Analyzer GUI and viewing profiled data on page 2-15.](#)

2.1.1 Important operational notes

Review the following operational notes 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

Follow the instructions below to enable System Analyzer functionality in SoC Designer and generate data to use with System Analyzer.

Procedure

1. In SoC Designer, select **File > Preferences**.
2. In the **Preferences** dialog, select **Profile**.
3. Enable the checkbox **Use external profile database**. 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* (100996) if you need instructions.
6. Run the simulation to begin writing profiling data to the `.profile` file.

2.1.3 Launching the System Analyzer GUI and viewing profiled data

After running a simulation with the desired monitors enabled, launch System Analyzer to view the results. 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.

.Profile name and location

All profiling simulation data for monitored connections is captured to a profiling data file. On Linux, profiling data is stored in your working directory by default. On Windows, the profiling data files is stored in the same directory as your .mxd 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

You can launch System Analyzer from the SoC Designer GUI, from the Linux command line, or from the Windows Start menu.

To launch System Analyzer from:

- *SoC Designer* — In SoC Designer Simulator, click the **Analyzer** icon. If the icon is grayed out, enable the **Use external profile database** option in the Preferences, Profile dialog.

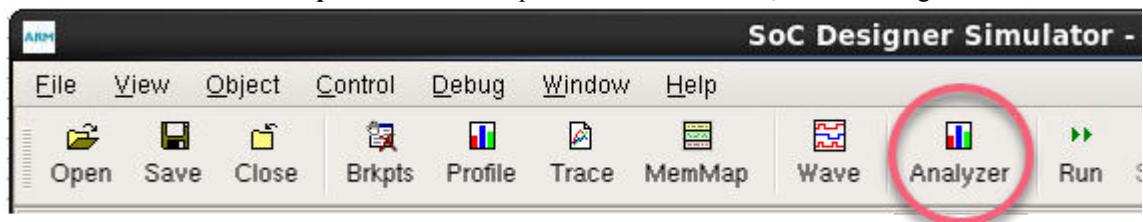


Figure 2-1 System Analyzer icon in SoC designer simulator toolbar

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

Related tasks

[2.1.2 Enabling System Analyzer and generating data on page 2-15.](#)

Related references

[Command line arguments on page 2-16.](#)

Command line arguments

This section describes the SoC Designer System Analyzer 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

The System Analyzer **File** menu shows the most recent .profile files, providing quick access to your profiles.

Open the System Analyzer GUI and select **File > Open** or select from the list of recently-opened files, as shown in the following figure.

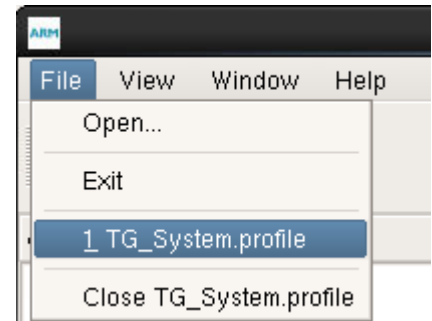


Figure 2-2 Opening a System Analyzer .profile file

2.2 System Analyzer GUI reference

The System Analyzer GUI has three main sections: the Plugins Panel, the Analyzer Explorer tree, and the main display window.

- **Plugins panel** — Displays buttons for launching plugins that you develop; the System Metrics plugin comes with SoC Designer and is available by default.
- **Analyzer Explorer.**
- **Main display window.**

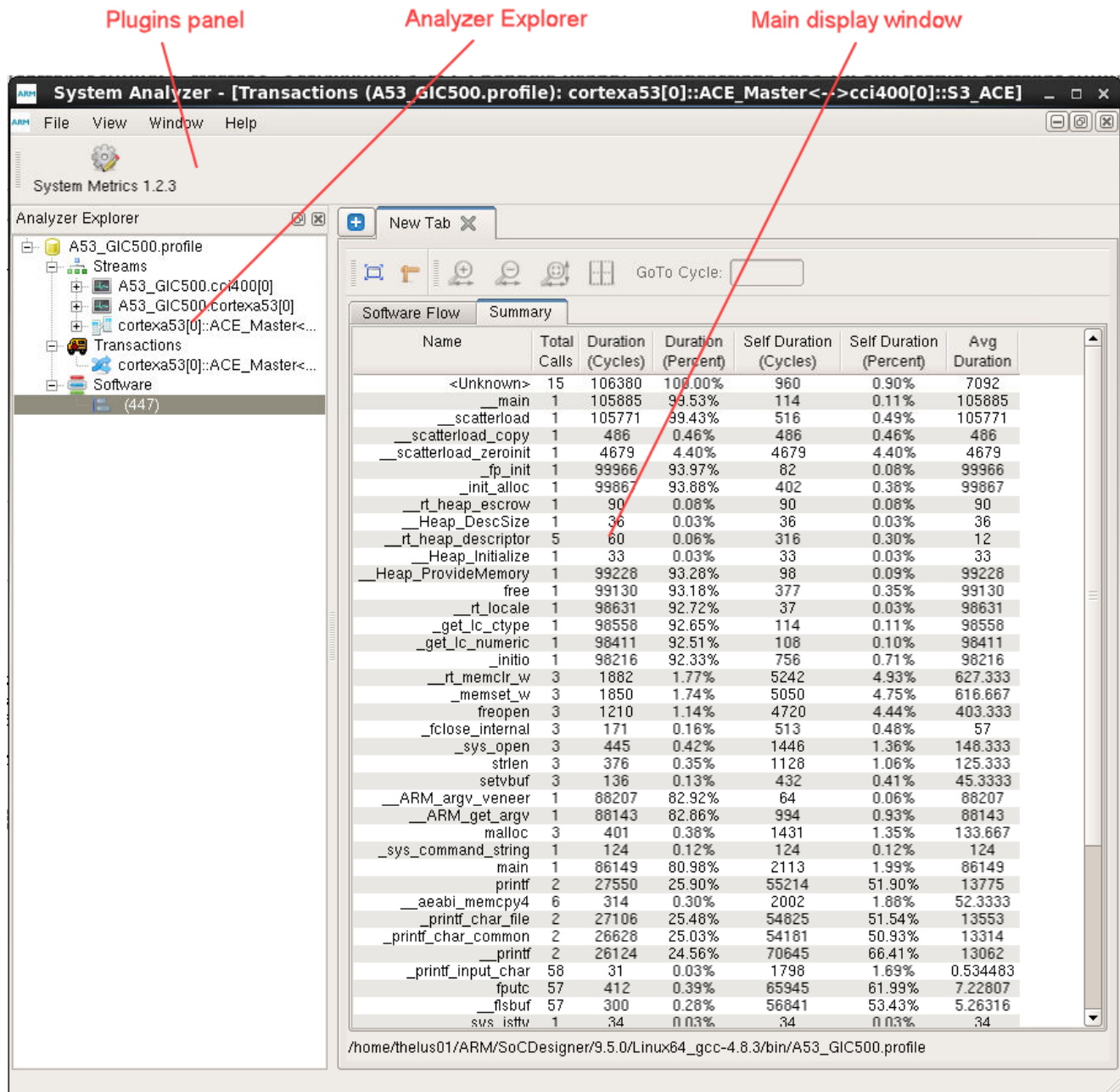


Figure 2-3 System Analyzer GUI

This section contains the following subsections:

- [2.2.1 Overview of the Analyzer Explorer on page 2-19.](#)

- [2.2.2 Overview of the main display window on page 2-19.](#)

2.2.1 Overview of the Analyzer Explorer

The Analyzer Explorer is a standard, hierarchical expansion tree. For each simulation file it displays applicable profiling data for streams, transactions, and software.

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.
- **Transactions** — Data associated with transactions on a connection, including Program Counter and Function information. The data varies depending on the protocol in use.
- **Software** — On a per-core basis, data such as program flow, total calls, and function duration.

You can toggle the Analyzer Explorer panel using **View > Analyzer Explorer**.

Related concepts

[2.3.1 Working in the Streams view on page 2-20.](#)

[2.3.2 Working in the Transactions view on page 2-23.](#)

[2.3.3 Working in the Software View on page 2-25.](#)

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 **Analyzer Explorer** tree.

Toolbar functions that persist regardless of context are shown in the following figure.

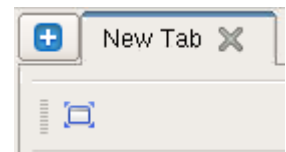


Figure 2-4 Persistent toolbar functions

The following buttons are available regardless of context:

- **Create New Tab** button (white plus sign on a blue field) — Creates a new tab. When you select a node in the Analyzer Explorer tree, its data populates the new tab rather than replacing the data in the existing tab.

You can rename any tab with a meaningful name. Right-click on the tab and select **Rename** from the context menu.

- **Float/Dock** button — This toggle button makes the window independent of the System Analyzer application, or re-docks the window.

Standard Tile and Cascade functionality are available under the **Window** menu.

Related concepts

[2.5 Using the System Metrics Plugin on page 2-29.](#)

Related references

[Appendix A Developing plugins on page Appx-A-32.](#)

2.3 Viewing profiling data

You can work with profiling data in the Streams view, Transactions view, or Software view.

This section contains the following subsections:

- [2.3.1 Working in the Streams view on page 2-20.](#)
- [2.3.2 Working in the Transactions view on page 2-23.](#)
- [2.3.3 Working in the Software View on page 2-25.](#)

2.3.1 Working in the Streams view

In **Streams** view you can specify plot characteristics to fine-tune your view of profiling data, and specify which event types to show.

The following figure shows the **Streams** view in the Analyzer 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.

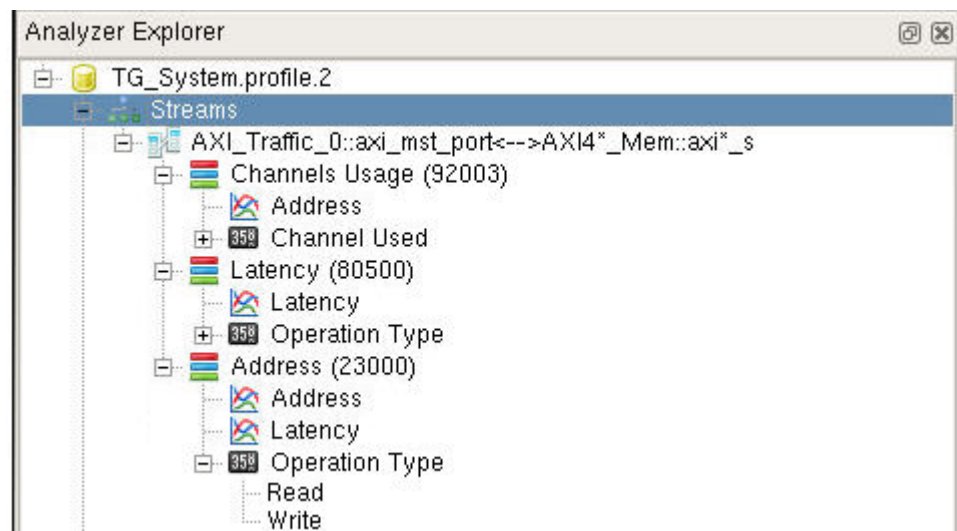


Figure 2-5 Streams view

Specifying plot characteristics

In the **Streams** view, you can fine-tune the view settings by selecting a stream from the tree.

When you click on a stream in the tree, the **Configure Plot** dialog appears (as shown in the figure). 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 from the pull.
- **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.

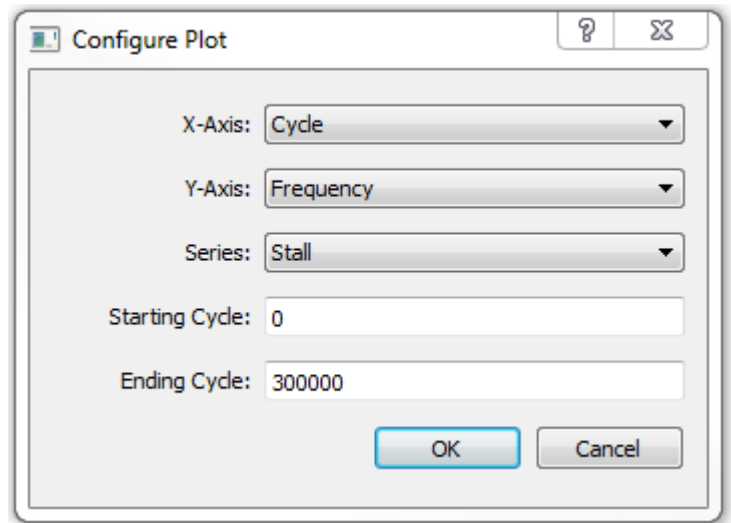


Figure 2-6 Configuring plot characteristics

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.

The Streams view toolbar

The Streams view toolbar provides tools for resizing the data view, zooming to a specific Cycle Range, adding a cursor to the view, changing the view granularity, and other options when viewing a data stream.

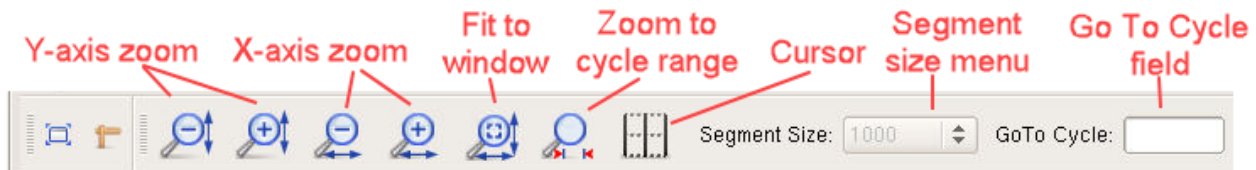


Figure 2-7 Streams view toolbar

- **Zoom out** and **Zoom in** along the Y Axis. Increases the vertical granularity of the view without changing the lateral granularity.
- **Zoom out** and **Zoom in** along the X Axis. Increases the lateral granularity of the view without changing the vertical granularity.
- **Fit to window**. Fits all available data into the view.
- **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.

- **Cursor**. Adds a cursor to the view. Dragging the cursor across the chart displays the function executing during each cycle.
- **Segment Size** menu. Increases or decreases the granularity of the display. For a granularity of 1, use **Zoom to Cycle Range**.
- **Go To Cycle** field. Causes the view to jump to the specified cycle number.

Specifying event types

By default, the Streams view shows all event types in its displays, which include the Stacked Bar Chart and the Scatter Plot view.

In the **Stacked Bar Chart** view, the **Event Type** panel 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.

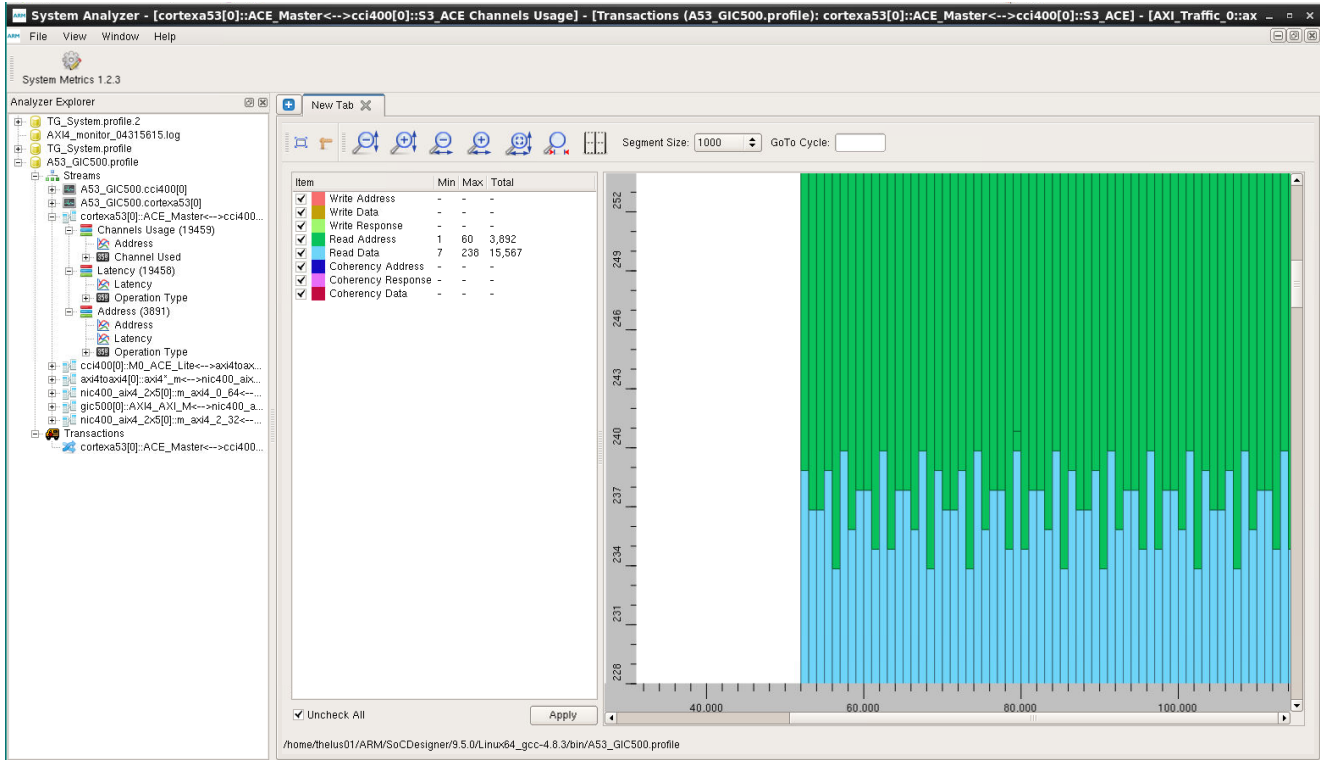


Figure 2-8 Event Type Panel with stacked bar chart

Related concepts

[Streams view displays on page 2-22.](#)

Streams view displays

Depending on the type of profiling node selected, the **Streams** view displays the **Stacked Bar** chart and the **Scatter Plot** chart.

- **Stacked Bar** chart, as shown in the previous figure.
- **Scatter Plot** chart. This format displays latency values, with Latency along the Y axis and Cycle number along the X axis, as shown in the following figure. The minimum and maximum number of each event type within the specified range is displayed in the Event Type panel.

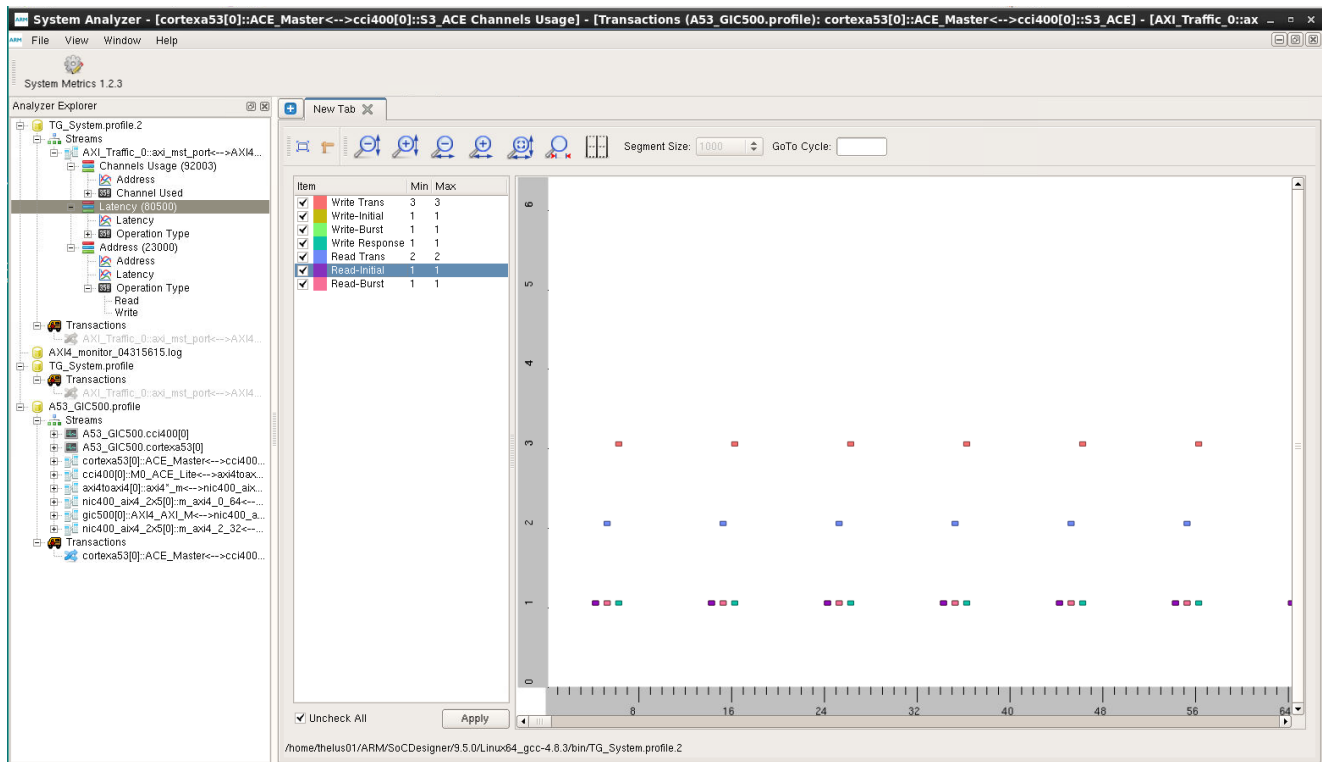


Figure 2-9 Latency scatter plot

2.3.2 Working in the Transactions view

The **Transactions** view allows you to constrain, focus and filter views, view data beats, and display software trace data.

Transactions view overview

When you click a Transactions-related node, System Analyzer displays data associated with particular transactions. The data displayed varies depending on the protocol in use.

The following figure shows a transaction view for AXI4-ACE data.

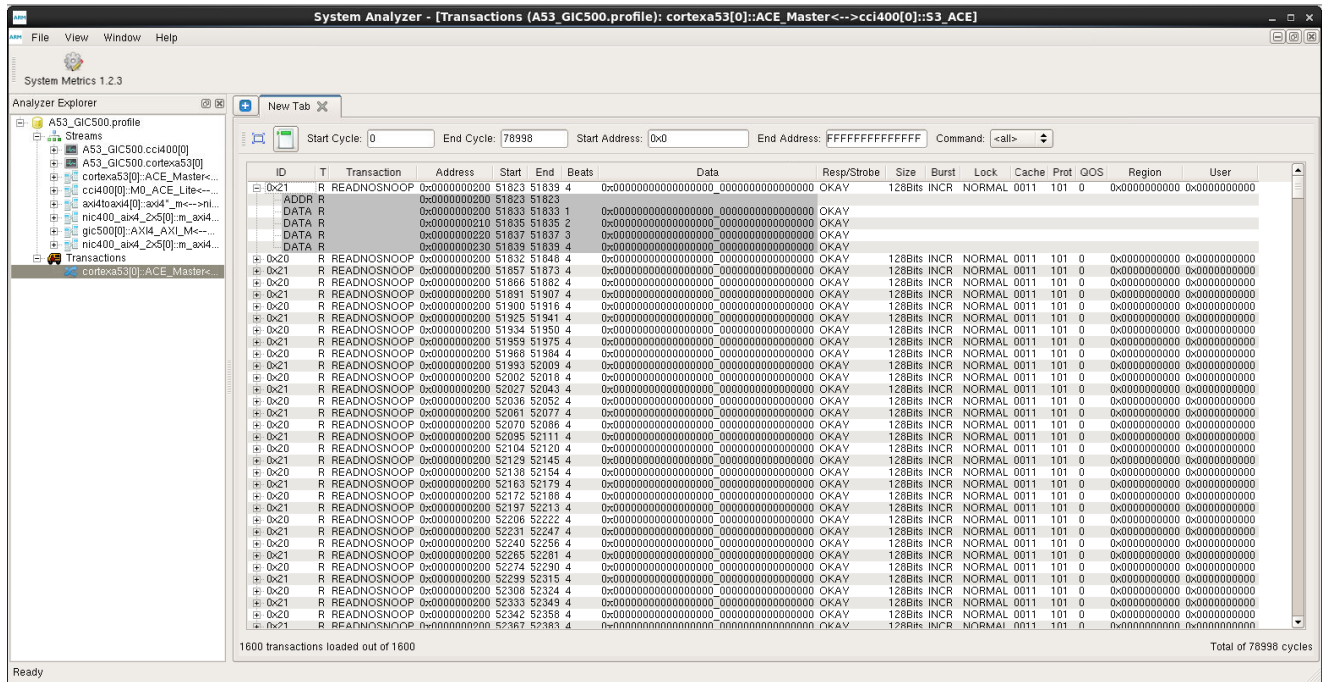


Figure 2-10 Transaction View

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.

Expand a transaction to display its data beats.

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.

Displaying software trace data

In the **Transactions** view, you have the option of viewing the Program Counter and function information associated with each transaction. Click the **Trace Software** button on the toolbar; PC and Function columns appear at the right of the display.

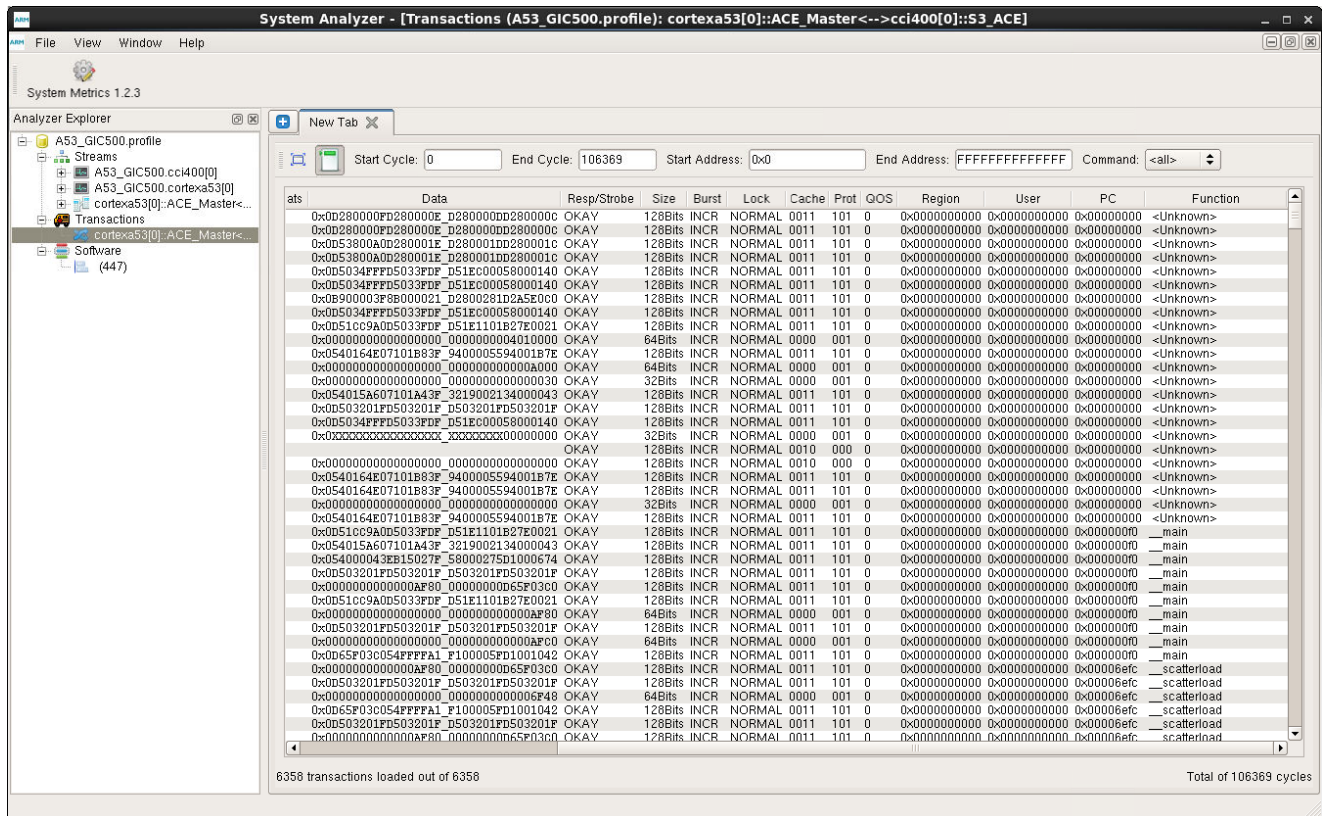


Figure 2-11 Software trace enabled

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 the **Software Flow** and **Software Summary** tabs.

Software Flow Tab

The **Software Flow** tab displays a chart of the program flow by function name and cycle.

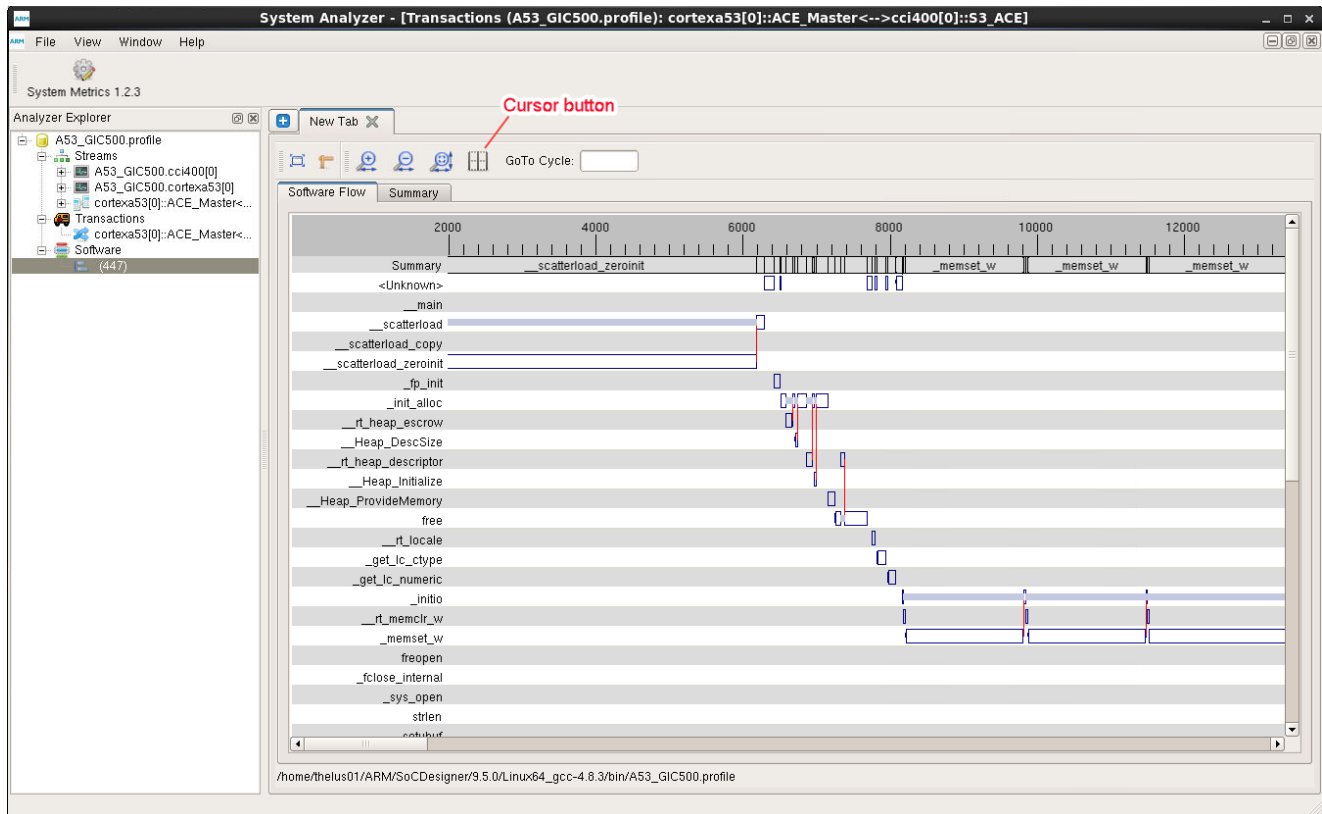


Figure 2-12 Software Flow tab

Using the Cursor to display the executing function

You can display the name of the executing function at different cycles with the **Cursor** button.

To display the name of the executing function at different cycles, click the **Cursor** button and drag the cursor 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.

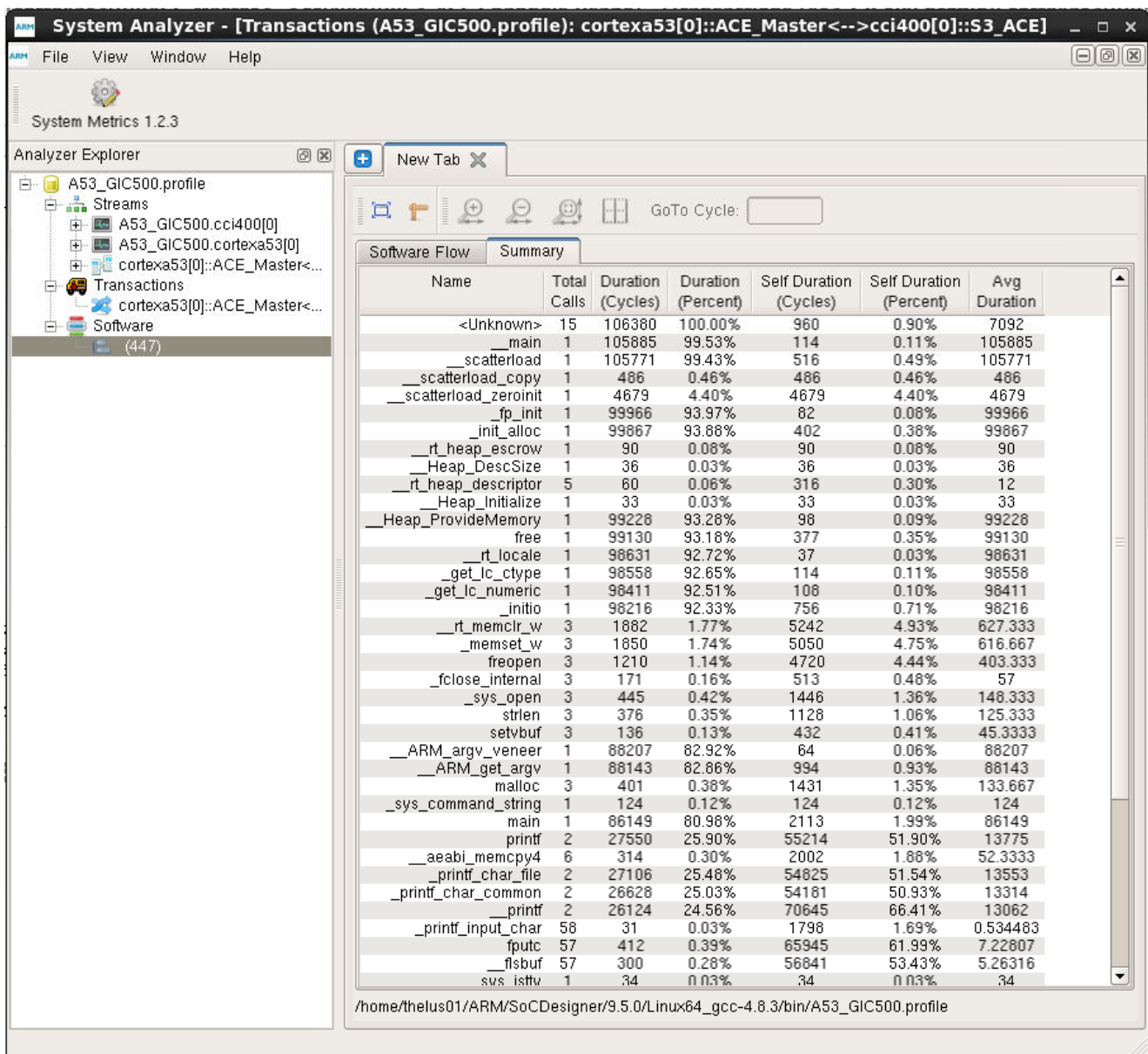


Figure 2-13 Software Summary view

Related concepts

[Software Flow Tab on page 2-25.](#)

[Software Summary tab on page 2-26.](#)

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 Analyzer Explorer as a top-level node with a .profile suffix.

This section contains the following subsection:

- [2.4.1 Locking Scroll for All Windows on page 2-28](#).

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.



Figure 2-14 Lock Scroll button

To enable scroll lock functionality:

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

To disable scroll lock, toggle the **Lock Scroll** 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* (100996) 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.

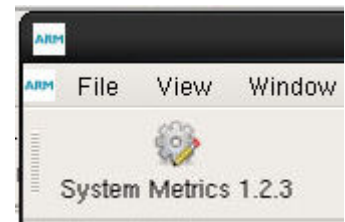


Figure 2-15 System Metrics Button in System Analyzer

The System Metrics dialog opens, as shown in the following figure.

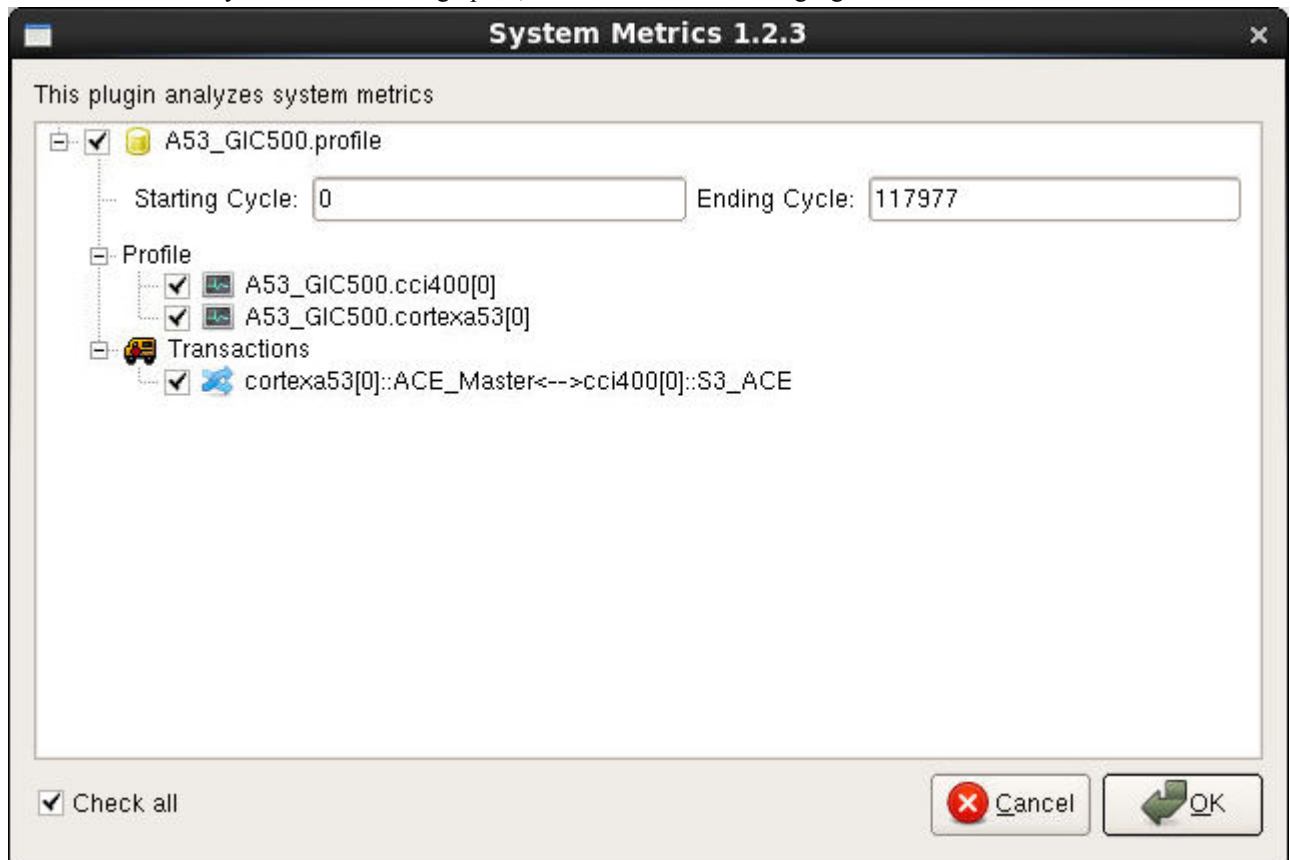


Figure 2-16 System Metrics selection dialog

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). 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.

This section contains the following subsection:

- [2.5.1 Sample Reports on page 2-30.](#)

2.5.1 Sample Reports

This section contains a sample report generated by the System Metrics plugin..

The following sample report is for a Cortex-A53 system with an AXI4-ACE bus.

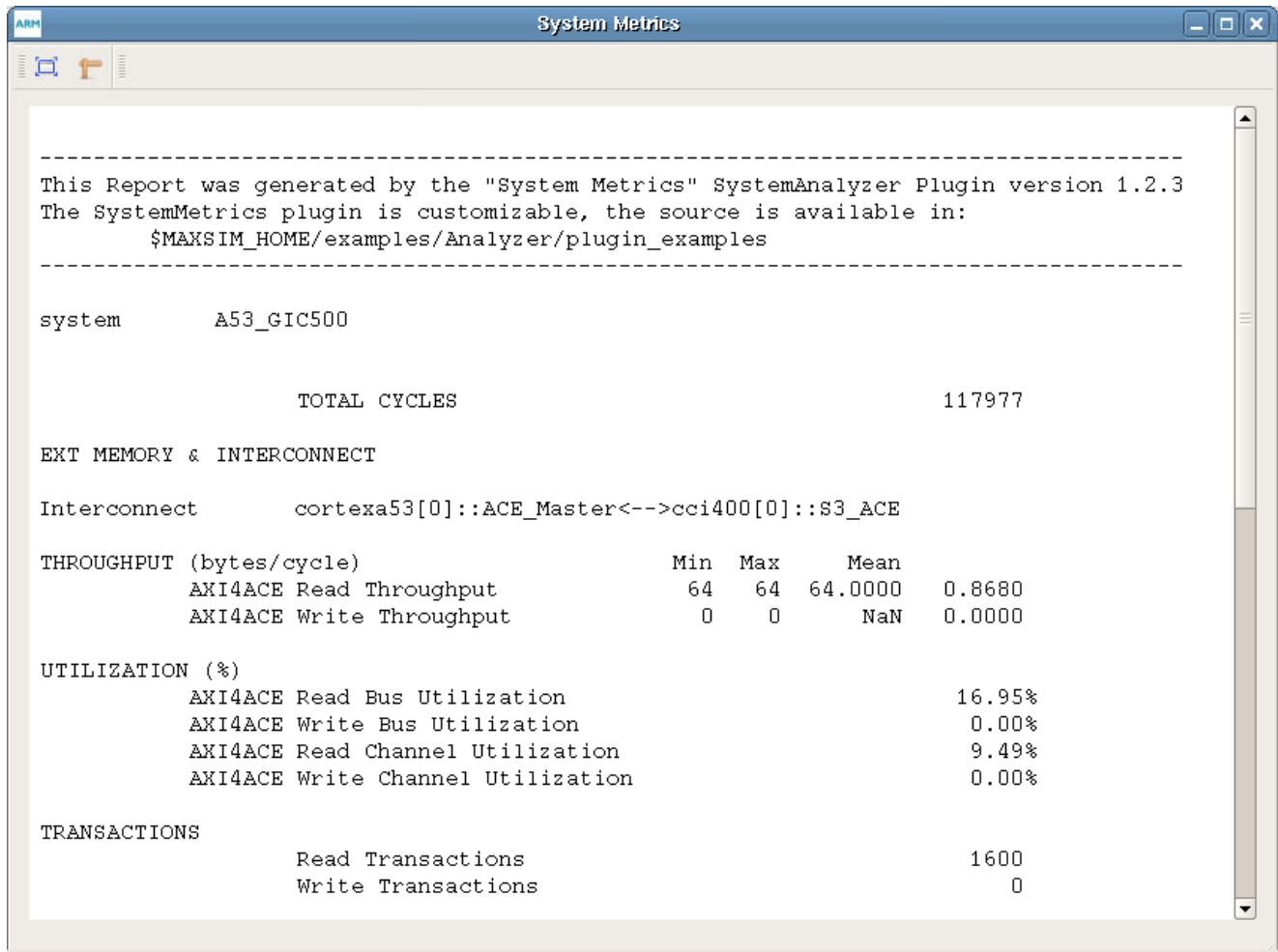


Figure 2-17 System Metrics Report (AXIv2)

If you specified a Starting Cycle 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.

Related references

[Appendix B Metrics and formulas on page Appx-B-46.](#)

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.

It contains the following sections:

- *A.1 Before you begin* on page Appx-A-33.
- *A.2 Creating a plugin assembly* on page Appx-A-34.
- *A.3 Enabling plugins to run in System Analyzer* on page Appx-A-39.
- *A.4 Enabling plugins to run as a console application* on page Appx-A-40.
- *A.5 Driving the program from a test driver* on page Appx-A-41.
- *A.6 Plugin method reference* on page Appx-A-42.
- *A.7 Running ValidateMetrics with the CHI Example* on page Appx-A-45.

A.1 Before you begin

Before you begin writing your own plugins, study the System Analyzer's `.profile` file to understand its structure and see what data it collects. You should also familiarise yourself with the functions that are available in the System Analyzer API. Furthermore, it is often easier to modify an existing plugin rather than creating one from scratch yourself. Several example plugins have been provided with the System Analyzer which you can modify for your own purposes.

This section contains the following subsections:

- [A.1.1 Reviewing the `.profile` file on page Appx-A-33.](#)
- [A.1.2 Reviewing the System Analyzer API on page Appx-A-33.](#)
- [A.1.3 Modifying an example on page Appx-A-33.](#)

A.1.1 Reviewing the `.profile` file

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.

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.

Related concepts

[2.1 Enabling System Analyzer profiling on page 2-15.](#)

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.

Related concepts

[A.2.1 Understanding the data structures on page Appx-A-34.](#)

A.2 Creating a plugin assembly

Before you can create a plugin assembly, you need to understand the data structures System Analyzer plugins interact with, and the coding requirements for plugins.

This section contains the following subsections:

- [A.2.1 Understanding the data structures on page Appx-A-34.](#)
- [A.2.2 Plugin coding requirements on page Appx-A-34.](#)
- [A.2.3 Creating the project on page Appx-A-34.](#)

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 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

List of higher-level CHI methods facilitate CHI metric computation. These methods are in addition to the methods that apply to CHI connections that exist in the System Analyzer API.

- `UniqueTransactionsCount()` — How many unique transactions are on a particular CHI connection.
- `FlitsOnChannelCount(Int64 channel)` — Returns the number of flits on a particular channel.
- `Ienumerable<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).

Related concepts

[2.5 Using the System Metrics Plugin on page 2-29.](#)

[A.7 Running ValidateMetrics with the CHI Example on page Appx-A-45.](#)

A.2.2 Plugin coding requirements

This section describes the basic coding requirements for your Plugin program, regardless of programming platform.

- 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

You can create a project using an IDE, such as Visual Studio, or you can create one standalone, that is, without the use of an IDE.

Creating a project in Visual Studio

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

Procedure

1. Open Visual Studio and create a new Console project.
2. Select **Application** in the left panel and set **Target framework** to **.NET 4.0**:

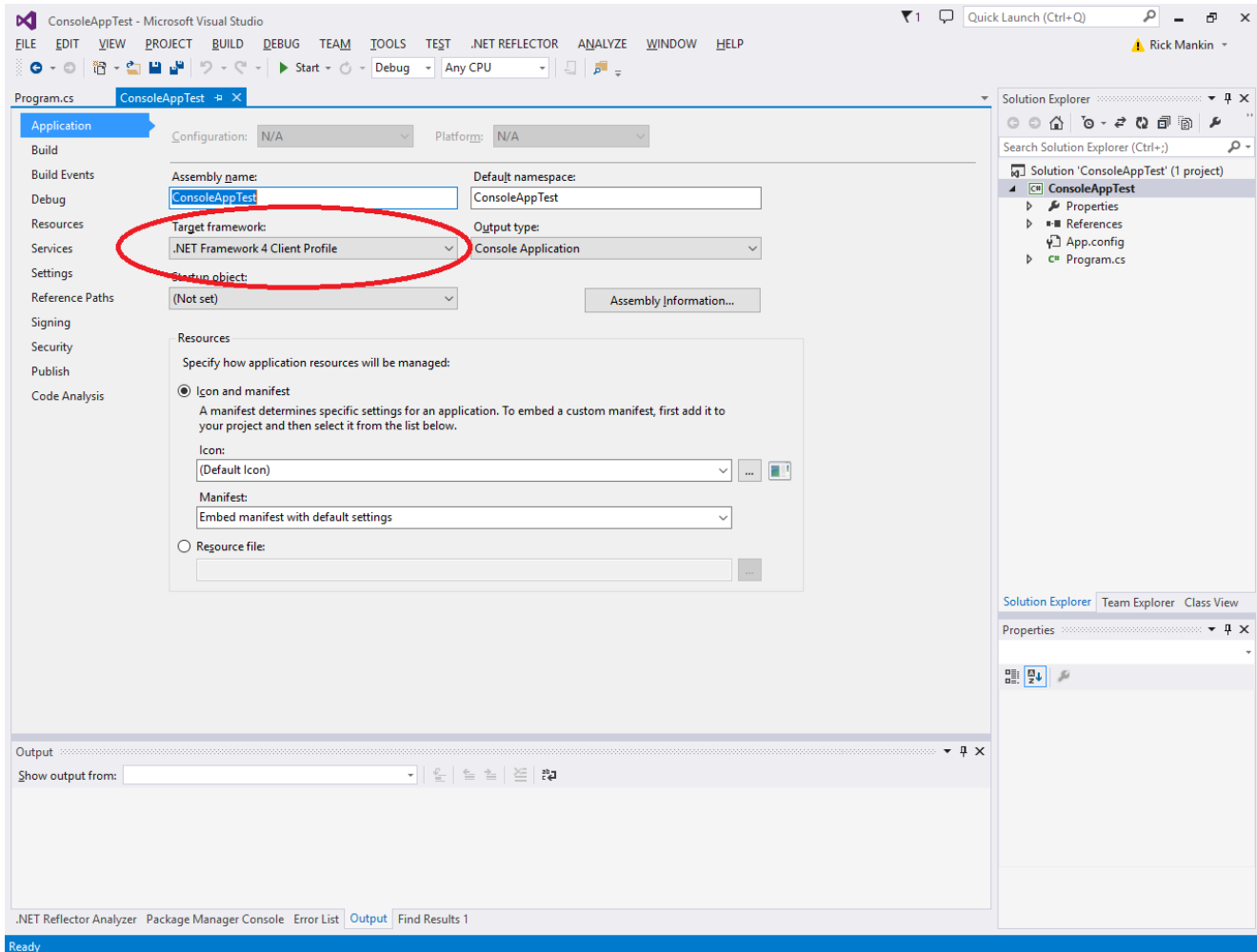


Figure A-1 Setting the target framework

3. Add references to the files CarbonAnalyzerAPI.d11 and CarbonAutoEnums.d11. These are located in either \$MAXSIM_HOME/lib/Linux/release or \$MAXSIM_HOME/Lib/Win32/Release.

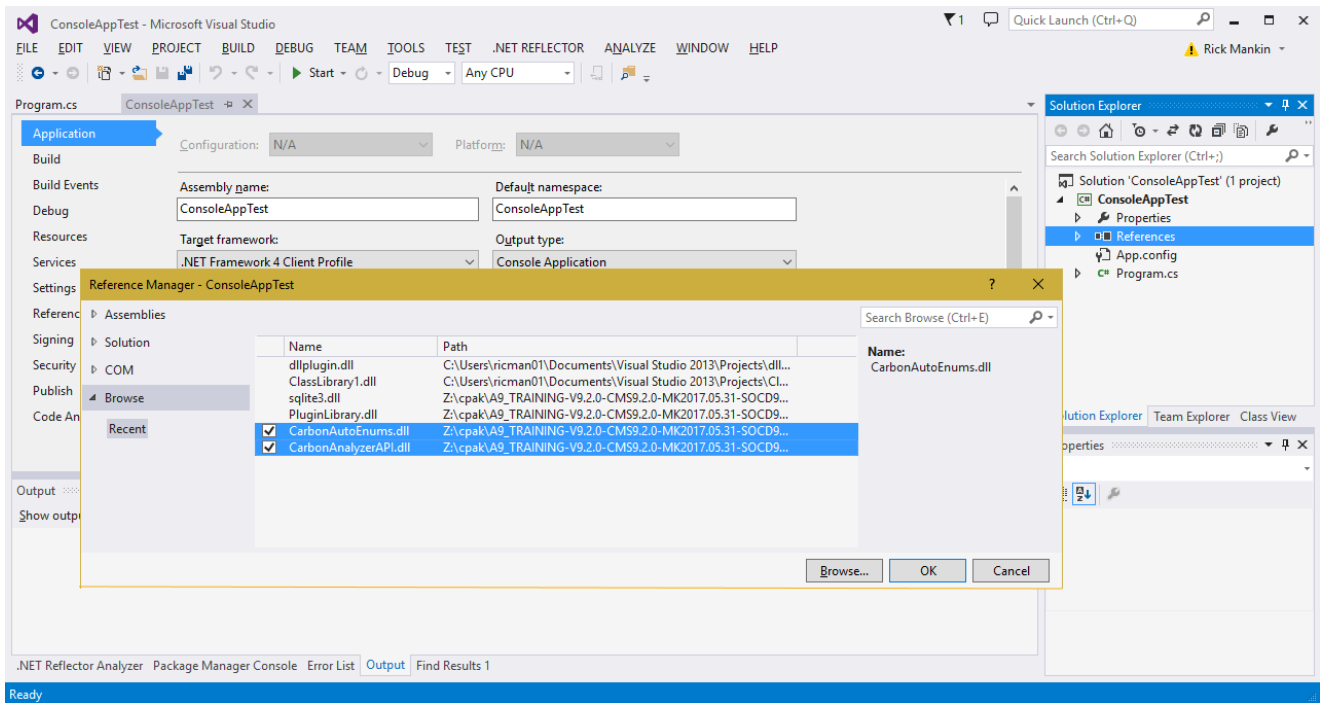


Figure A-2 Adding references in Visual Studio

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
```
6. Ensure your plugin passes in an instance of the System Analyzer database using `Carbon.Analyzer.Plugins.AnalyzerArgument[]`.
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.

Example A-1 Sample plugin

The following is an example of a simple Plugin program:

```
public string Description
{
    get { return "Plugin1 Description Here"; }
}
public string Name
{
    get { return "Plugin1"; }
}
public Carbon.Analyzer.Plugins.AcceptableSourceTypes Accepts
{
    get { return Carbon.Analyzer.Plugins.AcceptableSourceTypes.Transaction; }
}
public string Analyze(Carbon.Analyzer.Plugins. AnalyzerArgument[] arguments)
{
    foreach (Carbon.Analyzer.Plugins.AnalyzerArgument arg in 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;
                if (tx.Command == CarbonEnum.Transaction.Command.READ)
                {
                    latency = b.BeatComplete - tx.AddressStart;
                    if (latency > maxReadLatency)
                        maxReadLatency = latency;
                    numReads++;
                }
                if (tx.Command == CarbonEnum.Transaction.Command.WRITE)
                {
                    latency = tx.ResponseComplete - tx.AddressStart;
                    if (latency > maxWriteLatency)
                        numWrites++;
                }
            }
            if (latency > maxLatency)
                maxLatency = latency;
            break;
        }
    }
    // 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 "";
}

```

Related concepts

[A.3 Enabling plugins to run in System Analyzer on page Appx-A-39.](#)

Standalone instructions

For IDE-independent plugin development, you can use either an Arm-supplied wrapper script or `dmcs`, which is a C# compiler and linker that runs on Linux for compilation.

Instructions for compiling your plugin 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.

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.

Related concepts

[A.4 Enabling plugins to run as a console application on page Appx-A-40.](#)

A.3 Enabling plugins to run in System Analyzer

To make a plugin accessible to System Analyzer, either copy its .exe or .dll to the plugins folder, or set the CARBON_ANALYZER_PLUGINS environment variable so System Analyzer is able to recognise any .exe or .dll stored in the plugin directory.

- 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 either the wrapper script approach or environment variable approach.

This section contains the following subsections:

- [A.4.1 Wrapper script approach on page Appx-A-40.](#)
- [A.4.2 Environment variable approach on page Appx-A-40.](#)

A.4.1 Wrapper script approach

The Arm-provided wrapper script `systemmetrics.bat` is available in `$MAXSIM_HOME/bin` (Linux) or `%MAXSIM_HOME%\bin` (Windows). This script is for the Arm-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`.

Related concepts

[Standalone instructions on page Appx-A-37.](#)

A.5 Driving the program from a test driver

This section describes how to drive your plugin program from a Visual Studio test driver.

This section contains the following subsection:

- [A.5.1 Visual Studio instructions on page Appx-A-41.](#)

A.5.1 Visual Studio instructions

Driving a plugin program from a test driver using Visual Studio.

Procedure

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:

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

        Carbon.Analyzer.Plugins.AnalyzerArgument arg = new
Carbon.Analyzer.Plugins.AnalyzerArgument();
        arg.DatabaseFileName = @"A9.profile";
        arg.SourceType = Carbon.Analyzer.Plugins.AcceptableSoruceTypes.Transaction;
        string html = p1.Analyze(new Carbon.Analyzer.Plugins.AnalyzerArgument[] { arg });
        Console.WriteLine(html);
        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`.

Postrequisites

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

A.6 Plugin method reference

You can reference your plugins using the Name and Description method, the Accepts method, and the Analyze method.

This section contains the following subsections:

- [A.6.1 Name and description methods on page Appx-A-42.](#)
- [A.6.2 Accepts method on page Appx-A-42.](#)
- [A.6.3 Analyze Method on page Appx-A-44.](#)

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 the following figure, the Name method assigns name System Metrics 1.2.3 to the plugin.

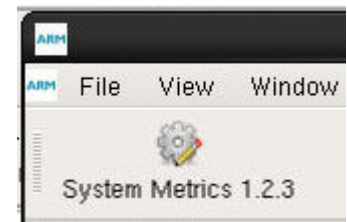


Figure A-3 Plugin button in System Analyzer

A.6.2 Accepts method

The Accepts method indicates the kind of data that this plugin supports.

The following options are available for the Accepts method: All, Transaction, Connection, Database, Profile, Software and Transaction.

For example, to create a plugin that accepts transaction data, use `AcceptableSourceTypes` in the Accepts method of the plugin code to specify Transactions. For example:

```
public Carbon.Analyzer.Plugins.AcceptableSourceTypes Accepts
{
    get { return Carbon.Analyzer.Plugins.AcceptableSourceTypes.Transaction; }
}
```

When the plugin is implemented in the System Analyzer GUI, its dialog appears as shown in the following figure. From this view, you can pick connections of interest related to analyzing Transactions.

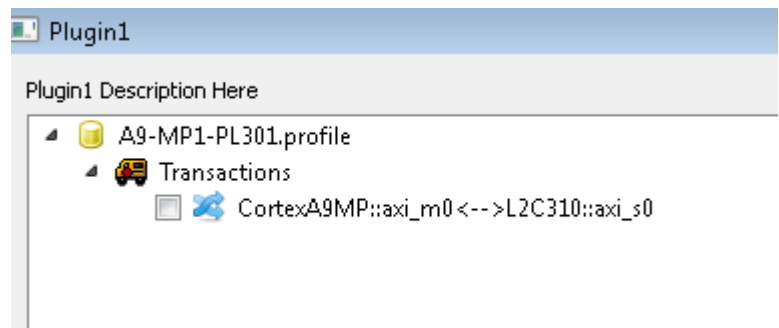


Figure A-4 Plugin dialog

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. Monitor and Profiler probes must be enabled in SoC Designer on the system connections of

interest. Refer to the *SoC Designer User Guide* (100996) for information on configuring Monitor and Profiler probes.

- **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. Use the Profiling Manager in SoC Designer to enable profiling on hardware streams of interest, as shown in the following figure. Refer to the *SoC Designer User Guide* (100996) for information about the Profiling Manager.

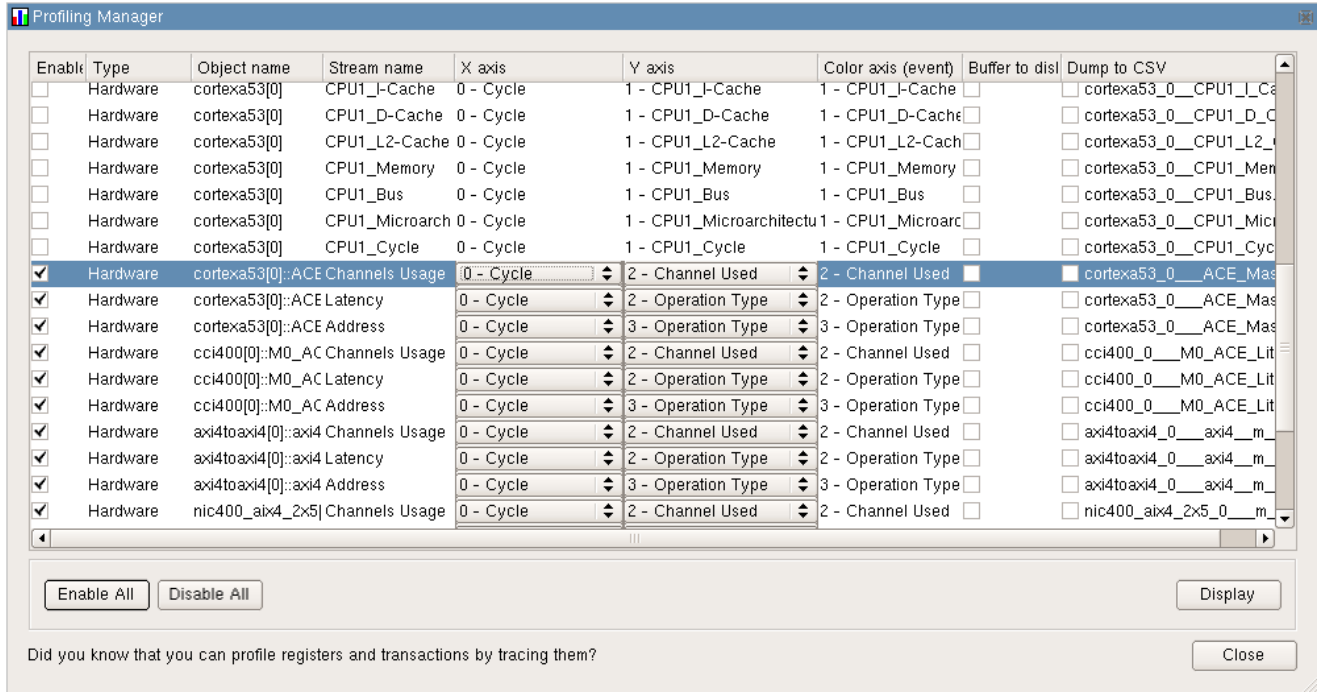


Figure A-5 Profiled hardware streams in Profiling Manager

- **Software** — Plugin presents choices related to software that was captured from SoC Designer. Use the Profiling Manager in SoC Designer to enable profiling on software of interest. Refer to the *SoC Designer User Guide* (100996) for information about the Profiling Manager. The following figure shows a software stream (as well as hardware streams) enabled for profiling in the Profiling Manager.

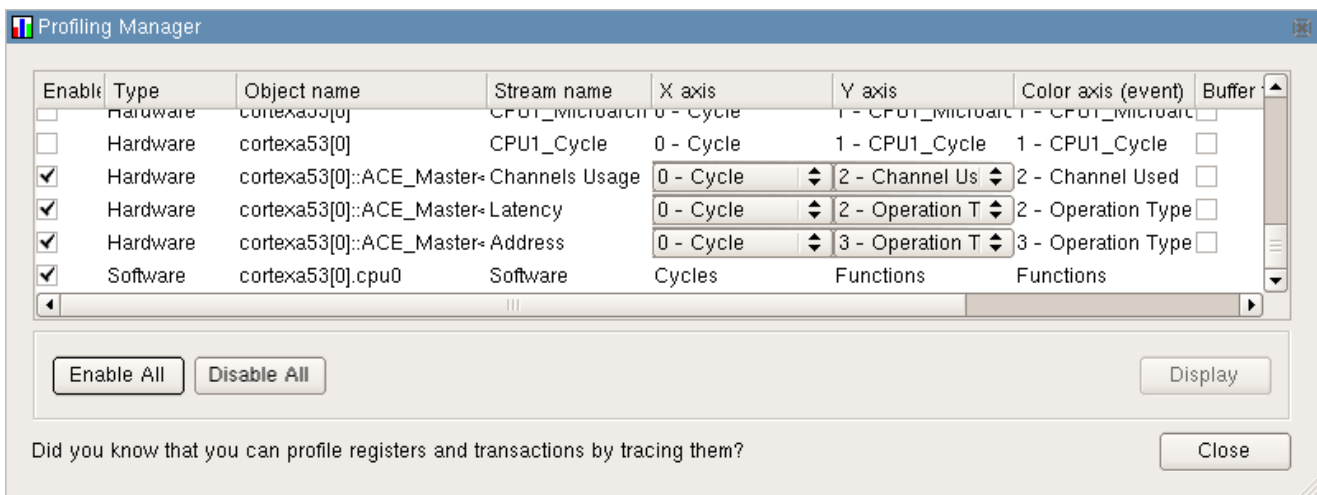
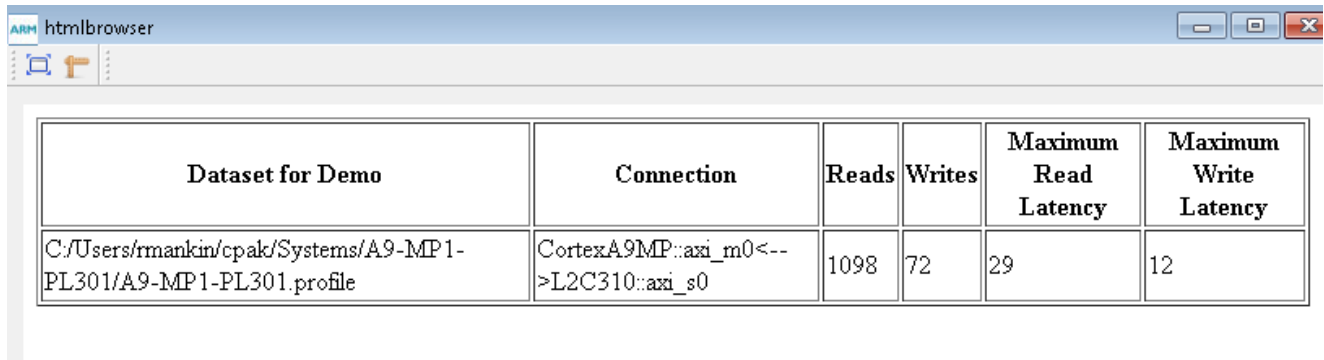


Figure A-6 Profiled software in Profiling Manager

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 the following figure iterates over the transactions and produces the maximum latency information, producing HTML output similar to that shown in the following figure:



The screenshot shows a web browser window titled 'htmlbrowser'. Inside the browser, there is a table with the following data:

Dataset for Demo	Connection	Reads	Writes	Maximum Read Latency	Maximum Write Latency
C:/Users/rmankin/cpak/Systems/A9-MP1-PL301/A9-MP1-PL301.profile	CortexA9MP::axi_m0<-->L2C310::axi_s0	1098	72	29	12

Figure A-7 HTML output

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 supported 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.

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

This section contains the following subsection:

- [A.7.1 Modifying ValidateMetrics to use with custom profiles on page Appx-A-45](#).

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.

Related concepts

[A.7.1 Modifying ValidateMetrics to use with custom profiles on page Appx-A-45](#).

Appendix B

Metrics and formulas

This appendix provides a list of AXI-related calculations and formulas that can be used to perform analysis, and a list of CHI formulas and variable definitions that are used with CHI metrics.

It contains the following sections:

- *B.1 AXI4 metrics and formulas table* on page Appx-B-47.
- *B.2 CHI metrics and formulas table* on page Appx-B-50.

B.1 AXI4 metrics and formulas table

List of AXI-related calculations and formulas that can be used to perform analysis. This is a partial list.

Table B-1 AXI4 formulas and variable definitions

Calculation		Formula
channel utilization <i>Ignore AW, AR, and B channels; these are not interesting.</i>	Write channel	$(\sum (Dn_End - D1_Start + 1)) / \text{cycles}$ where: <ul style="list-style-type: none"> Dn_End — Last Data Beat, end cycle. D1_Start — First Data Beat, start cycle.
	Read channel	$(\sum (Dx_End - Dx_Start + 1)) / \text{cycles}$ where: <ul style="list-style-type: none"> Dx_End — First Data beat, start cycle. <i>x</i> holds the beat number. Dx_Start — Last Data beat, start cycle. <i>x</i> holds the beat number.
bus utilization <i>Do not double count when transactions overlap.</i>	Write	$(\text{number of cycles a transaction is active (AW_Start without Rsp_End)}) / \text{cycles}$ where: <ul style="list-style-type: none"> AW_Start — Write Address, start cycle. Rsp_End — Write Response end (B Channel). This (response) is the final step for a write transaction.
	Read	$(\text{number of cycles a transaction is active (AR_Start without Dn_End)}) / \text{cycles}$ where: <ul style="list-style-type: none"> Ax_Start — <i>x</i> 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	$(\sum (\text{Size} * \text{Beats})) / \text{cycles}$
channel efficiency	Read/Write channel	$(\sum (\text{Size} * \text{Beats})) / (\text{cycles} * \text{channel_utilization} * \text{BUS_WIDTH})$

Table B-1 AXI4 formulas and variable definitions (continued)

Calculation		Formula
latency calculations	Write-trans (AW)	$(\sum (AW_End - AW_Start + 1)) / \#total_transactions$ where: <ul style="list-style-type: none"> • AW_End — Write Address, end cycle • AW_Start — Write Address, start cycle
	Write-initial	$(\sum (D1_End - D1_Start + 1)) / \#total_transactions$ where: <ul style="list-style-type: none"> • D1_Start — First Data Beat, start cycle • D1_End — First Data Beat, end cycle
	Write-subsequent	$(\sum (Dx_End - Dx_Start + 1]_End))) / (\#total_beats - total_transactions)$ where: <ul style="list-style-type: none"> • Dx_End — First Data beat, start cycle. <i>x</i> holds the beat number. • Dx_Start — Last Data beat, start cycle. <i>x</i> holds the beat number. <p>\sum is for $x = 2-n$; in other words, first beats are ignored.</p>
	Write-burst	$(\sum (Dn_End - D1_Start + 1)) / \#total_transactions$ where: <ul style="list-style-type: none"> • Dn_End — Last Data Beat, end cycle. • D1_Start — First Data Beat, start cycle
	writetransactions	$(\sum (Rsp_End - AW_Start)) / total_transactions$ where: <ul style="list-style-type: none"> • Rsp_End — Write Response end (B Channel). This (response) is the final step for a write transaction. • AW_Start — Write Address, start cycle

Table B-1 AXI4 formulas and variable definitions (continued)

Calculation		Formula
latency calculations	Read-trans (AR)	$(\sum (AR_End - AR_Start + 1)) / \text{total_transactions}$ where: <ul style="list-style-type: none"> AR_End — Address Read channel, end cycle AR_Start — Address Read channel, start cycle
	Read-initial	$(\sum (D1_Start - AR_End)) / \text{total_transactions}$ where: <ul style="list-style-type: none"> D1_Start — First Data Beat, start cycle. AR_End — Address Read channel, end cycle.
	Read-subsequent	$\text{read-subsequent} ((\sum (Dx_Start - D[x-1]_Start))) / (\text{total_beats} - \text{total_transactions})$ where: <ul style="list-style-type: none"> Dx _Start — Last Data beat, start cycle. x holds the beat number. <p><i>\sum is for $x = 2-n$; in other words, first beats are ignored.</i></p>
	Read-burst	$(\sum (Dn_End - AR_End)) / \text{total_transactions}$ where: <ul style="list-style-type: none"> Dn_End — Last Data Beat, end cycle. AR_End — Address Read channel, end cycle.
	Read-transactions	$(\sum (Dn_End - AR_Start)) / \text{total_transactions}$ where: <ul style="list-style-type: none"> Dn_End — Last Data Beat, end cycle. AR_Start — Address Read channel, start cycle.
channel back pressure	Ax	For a given cycle: number of Ax_Starts without Ax_Ends where: <ul style="list-style-type: none"> 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.
	rd/wr	For a given cycle: number of beats not transferred for open transactions.
	B	For a given cycle: number of RSP_Starts without RSP_Ends where: <ul style="list-style-type: none"> Rsp_Start — Write Response start (B Channel). Rsp_End — Write Response end (B Channel). This (response) is the final step for a write transaction.
channel interleave counts	Ax	Number of open transactions.

B.2 CHI metrics and formulas table

List of CHI formulas and variable definitions used with CHI metrics. Note that *cycles* indicates the total cycles in simulation or the total cycles in sample period.

Table B-2 CHI formulas and variable definitions

Calculation		Formula
channel utilization	all channels	number of flits on channel / cycles
bus utilization	write	$\left(\sum (\text{number of cycles a transaction REQ write is active (end flit cycle - begin flit cycle + 1)}) \right) / \text{cycles}$ <p>or</p> $(\text{count of cycles where open write transactions} > 0) / \text{cycles}$ <p>Do not double count when transactions overlap.</p>
	read	$(\text{count of cycles where open read transactions} > 0) / \text{cycles}$ <p>or</p> $\left(\sum (\text{number of cycles a transaction REQ read is active (end flit cycle - begin flit cycle + 1)}) \right) / \text{cycles}$ <p>or</p> $(\text{count of cycles where open read transactions} > 0) / \text{cycles}$ <p>Do not double count when transactions overlap.</p>
	SNP	$\left(\sum (\text{number of cycles a transaction SNP is active (end flit cycle - begin flit cycle + 1)}) \right) / \text{cycles}$ <p>Do not double count when transactions overlap.</p>
bus throughput	write	$(\sum \text{TXREQ Size}) / \text{cycles}$ <p>For the TXREQ opcodes that begin with "Write"</p>
	read	$(\sum \text{TXREQ Size}) / \text{cycles}$ <p>For the TXREQ opcodes that begin with "Read"</p>
	SNP	$((\text{Number of TXDAT Snoop response flits}) * \text{bus_width (bytes)}) / \text{cycles}$ <p>TXDAT snoop resp opcodes include SnpRespData and SnpRespDataPtl</p>
channel efficiency	TXDAT	$(\sum \text{TXREQ Size}) / (\text{cycles} * \text{bus_utilization} * (\text{BUS_WIDTH}/8))$ <p>For the TXREQ opcodes that begin with "Write"</p>
	RXDAT	$(\sum \text{TXREQ Size}) / (\text{cycles} * \text{bus_utilization} * (\text{BUS_WIDTH}/8))$ <p>For the TXREQ opcodes that begin with "Read"</p>
	SNP	$((\text{Number of TXDAT Snoop response flits}) * (\text{BUS_WIDTH}/8)) / (\text{cycles} * \text{bus_utilization} * (\text{BUS_WIDTH}/8))$

Table B-2 CHI formulas and variable definitions (continued)

Calculation		Formula
latency	write	<ul style="list-style-type: none"> • write-trans (AW) — N/A • write-initial — $\sum ((\text{first DAT flit cycle}) - (\text{first flit cycle}) + 1)) / \text{total_transactions}$ • write-subsequent — $\sum ((\text{subsequent DAT flit cycle}) - (\text{previous DAT flit cycle}) + 1)) / \text{total_transactions}$ • write-burst — $\sum ((\text{last DAT flit cycle}) - (\text{first DAT flit cycle}) + 1)) / \text{total_transactions}$ • write-transactions — $\sum ((\text{last flit cycle}) - (\text{first flit cycle}) + 1)) / \text{total_transactions}$
	read	<ul style="list-style-type: none"> • read-trans (AR) — N/A • read-initial — $\sum ((\text{first DAT flit cycle}) - (\text{first flit cycle}) + 1)) / \text{total_transactions}$ • read-subsequent — $\sum ((\text{subsequent DAT flit cycle}) - (\text{previous DAT flit cycle}) + 1)) / \text{total_transactions}$ • read-burst — $\sum ((\text{last DAT flit cycle}) - (\text{first DAT flit cycle}) + 1)) / \text{total_transactions}$ • read-transactions — $\sum ((\text{last flit cycle}) - (\text{first flit cycle}) + 1)) / \text{total_transactions}$
channel interleave counts	REQ	Average number of open transactions per cycle
	SNP	Average number of open transactions per cycle

Appendix C

Revisions

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

It contains the following section:

- [C.1 Revisions on page Appx-C-53](#).

C.1 Revisions

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

List of changes that affect the various issues of this document.

Table C-1 Issue A

Change	Location	Affects
First Restamped release.	-	-