RealView® Debugger

Version 4.1

Trace User Guide

ARM®

ARM DUI 0322E (ID052710)

RealView Debugger
Trace User Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
March 2006	A	Non-Confidential	Release v3.0 for RealView Development Suite v3.0
March 2007	B	Non-Confidential	Release v3.1 for RealView Development Suite v3.1
September 2008	C	Non-Confidential	Release v4.0 for RealView Development Suite v4.0
27 March 2009	D	Non-Confidential	Release v4.0 SP1 for RealView Development Suite v4.0
28 May 2010	E	Non-Confidential	Release 4.1 for RealView Development Suite v4.1

Product Status

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

Web Address

`http://www.arm.com`

Contents
RealView Debugger Trace User Guide

Preface

This preface introduces the *RealView® Debugger Trace User Guide*. It contains the following sections:

- *About this book* on page vii
- *Feedback* on page xi.

About this book

This book describes how to use the RealView Debugger tracing features. See the other books in the RealView Debugger documentation suite for more information about the debugger.

Intended audience

This book has been written for users of RealView Debugger tracing features. It is assumed that users are experienced programmers, and have some experience with tracing.

Although prior experience of using RealView Debugger is not assumed, it is recommended that users first familiarize themselves with performing common debugging operations before using the tracing features. Also, you must understand how real-time tracing is beneficial in helping to debug programs that are running at full clock speed.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction to Tracing***

Read this chapter for a general overview of the RealView Debugger tracing features.

Chapter 2 ***Getting Started with Tracing***

Read this chapter for an overview of how trace hardware components operate together to enable you to perform tracing with RealView Debugger, and describes the general procedure for performing tracing on your application after you have configured your system.

Chapter 3 ***Configuring the Conditions for Trace Capture***

Read this chapter for details of how to configure the conditions for trace capture.

Chapter 4 ***Configuring the ETM***

Read this chapter for details of how to configure the *Embedded Trace Macrocell*™ (ETM) for capturing trace on ETM-enabled hardware.

Chapter 5 ***Tracepoints in RealView Debugger***

Read this chapter for an introduction to tracepoints in RealView Debugger.

Chapter 6 ***Setting Unconditional Tracepoints***

Read this chapter for an overview of the tracepoints available in RealView Debugger.

Chapter 7 ***Setting Conditional Tracepoints***

Read this chapter for details of how to set conditional tracepoints.

Chapter 8 ***Managing Tracepoints***

Read this chapter for details of how to manage tracepoints in RealView Debugger.

Chapter 9 ***Analyzing Trace with the Analysis Window***

Read this chapter for details of how to analyze captured trace information in RealView Debugger.

Chapter 10 ***Tracing Tutorial***

Provides a tutorial on how to use many of the tracing features in RealView Debugger. It is recommended that you follow this tutorial before using the tracing features on your application.

Appendixes

Appendix A ***Setting up the Trace Hardware***

Read this appendix for details on how to set up the hardware for the trace configurations supported by RealView Debugger.

Appendix B ***Setting up the Trace Software***

Read this appendix for details of how to set up the software for the trace configurations supported by RealView Debugger.

Conventions

Conventions that this book can use are described in the following sections.

Typographical

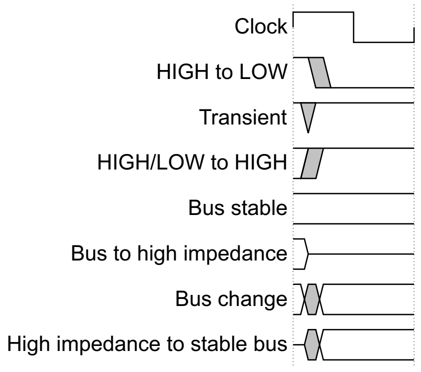
The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal 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 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: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`

Timing diagrams

The figure named *Key to event timing diagram conventions* on page ix explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



Key to event timing diagram conventions

Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in *Key to event timing diagram conventions*. If a timing diagram shows a single-bit signal in this way then its value does not affect the accompanying description.

Further reading

This section lists publications by ARM and by third parties.

See also:

- `http://infocenter.arm.com` for access to ARM documentation.
- `http://www.arm.com` for current errata, addenda, and Frequently Asked Questions.

ARM publications

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

- *RealView Debugger Essentials Guide* (ARM DUI 0181)
- *RealView Debugger User Guide* (ARM DUI 0153).
- *RealView Debugger Target Configuration Guide* (ARM DUI 0182)
- *RealView Debugger RTOS Guide* (ARM DUI 0323)
- *RealView Debugger Command Line Reference Guide* (ARM DUI 0175).

For details on using the compilation tools, see the books in the ARM Compiler toolchain documentation.

For details on using RealView Instruction Set Simulator, see the following documentation for more information:

- *RealView Instruction Set Simulator User Guide* (ARM DUI 0207).

For general information on software interfaces and standards supported by ARM tools, see `install_directory\Documentation\Specifications\...`.

See the datasheet or Technical Reference Manual for information relating to your hardware.

See the following documentation for information relating to the ARM debug interfaces suitable for use with RealView Debugger:

- *DSTREAM Setting Up the Hardware* (ARM DUI 0481)
- *DSTREAM System and Interface Design Reference* (ARM DUI 0499)

- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities* (ARM DUI 0498)
- *RealView ICE and RealView Trace Setting Up the Hardware* (ARM DUI 0515)
- *RealView ICE and RealView Trace System and Interface Design Reference* (ARM DUI 0517).

Other publications

For a comprehensive introduction to ARM architecture see:

Steve Furber, *ARM System-on-Chip Architecture, Second Edition*, 2000, Addison Wesley, ISBN 0-201-67519-6.

For a detailed introduction to regular expressions, as used in the RealView Debugger search and pattern matching tools, see:

Jeffrey E. F. Friedl, *Mastering Regular Expressions, Powerful Techniques for Perl and Other Tools*, 1997, O'Reilly & Associates, Inc. ISBN 1-56592-257-3.

For the definitive guide to the C programming language, on which the RealView Debugger macro and expression language is based, see:

Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language, second edition*, 1989, Prentice-Hall, ISBN 0-13-110362-8.

For more information about the JTAG standard, see:

IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std. 1149.1), available from the IEEE (`www.ieee.org`).

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any problems with this product, submit a Software Problem Report:

1. Select **Send a Problem Report...** from the RealView Debugger **Help** menu.
2. Complete all sections of the Software Problem Report.
3. To get a rapid and useful response, give:
   - a small standalone sample of code that reproduces the problem, if applicable
   - a clear explanation of what you expected to happen, and what actually happened
   - the commands you used, including any command-line options
   - sample output illustrating the problem.
4. E-mail the report to your supplier.

Feedback on this book

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

- the title
- the number, ARM DUI 0322E
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction to Tracing

This chapter introduces the requirements and RealView® Debugger features that enable you to capture and analyze trace information. It includes:

1.1 About Tracing in RealView Debugger

RealView® Debugger enables you to perform tracing on your application or system. You can capture in real-time a historical, non-intrusive trace of instructions and data accesses. Tracing is a powerful tool that enables you to investigate problems while the system runs at full speed. These problems can be intermittent, and are difficult to identify through traditional debugging methods that require starting and stopping the processor. Tracing is also useful when trying to identify potential bottlenecks or to improve performance-critical areas of your application.

RealView Debugger provides the following trace features:

- global trace configuration settings
- *Embedded Trace Macrocell*™ (ETM) configuration for ETM enabled hardware
- tracepoints to limit tracing to specific areas of your application
- various features to analyze the trace output
- the ability to obtain profiling information from the captured trace
- limited tracing capabilities with *RealView ARMulator® ISS* (RVISS).

The rest of this chapter gives a more detailed overview of the trace capabilities in RealView Debugger.

1.2 Requirements for tracing

RealView Debugger supports tracing with either trace hardware or a hardware simulator. The following sections describe the system requirements for both types of tracing:

- *Trace hardware*
- *Simulators* on page 1-4.

1.2.1 Trace hardware

To capture trace information using trace hardware, you must use one of the following solutions:

- a *Trace Port Analyzer* (TPA) solution, such as RealView Trace or RealView Trace 2, in conjunction with RealView ICE to capture trace from an *Embedded Trace Macrocell™* (ETM™)
- a *Joint Test Action Group* (JTAG) solution such as DSTREAM or RealView ICE to download trace data from an on-chip trace buffer, for example an *Embedded Trace Buffer™* (ETB™).

——— **Note** ———

RealView Debugger does not support:

- tracing from the external trace port of a SoC with DSTREAM
- tracing of *Digital Signal Processors* (DSPs).

———————————

Trace Port Analyzers

A TPA is a trace capture unit that is external to the ETM trace solution. RealView Debugger supports the following ARM TPAs in conjunction with a RealView ICE unit:

- RealView Trace
- RealView Trace 2 unit.

ETM trace solutions

If you are using an ETM trace solution that exports trace information to external trace capture hardware, that is a *Trace Port Analyzer* (TPA), then you must have the following components:

- an ETM-enabled ARM architecture-based processor
- a TPA, which can be:
  - RealView Trace
  - RealView Trace 2.

On-chip trace buffer solutions

RealView Debugger trace supports the following on-chip trace buffer solutions:

- ARM On-Chip Trace, which can be:
  - a non-CoreSight ETM with non-CoreSight ETB
  - a CoreSight ETM with CoreSight ETB
- Intel XScale.

You can use a DSTREAM or RealView ICE unit for on-chip tracing.

JTAG solutions

RealView Debugger trace supports RealView ICE version 1.1, and later JTAG solutions.

See also

- *ASIC that supports trace* on page 2-3
- Appendix A *Setting up the Trace Hardware*.

1.2.2 Simulators

If you do not have trace hardware, you can use RVISS for basic instruction and data tracing.

——— **Note** ———

Tracing is not supported on *Instruction Set System Model* (ISSM), *Real-Time System Model* (RTSM), and SoC Designer targets.

————————

Considerations when tracing on RVISS models

Be aware of the following when tracing on RVISS models:

- RVISS does not model the ETM. Trace capture is performed by directly capturing program execution information from the model.
- If you are using the RVISS Tracer feature to capture trace, you can capture trace only to a file. However:
  - — the captured trace is in text format, and can result in large file sizes
  - — you cannot load the file into the Analysis window.

1.3 Features for setting up the conditions for trace capture

The trace conditions you can set depend on the trace features supported by your debug target. Where supported, RealView Debugger enables you to:

- Set up the global trace conditions for generating trace information.
- Configure the trace hardware associated with the target processor to be traced.
- Use an automatic tracing mode to generate trace information.
- Set tracepoints to generate trace information in specific regions of your application. For example, you can:
  - start and stop tracing at specific addresses
  - specify a range of addresses where tracing of instructions and/or data occurs
  - specify a range of addresses where tracing of instructions and/or data does not occur
  - specify a trigger to indicate a point where trace information is of interest, and when hit to indicate that the captured trace is to be sent to the debugger.

The following sections describe the features for setting up the trace conditions:

- *Capturing data trace*
- *Capturing trace information for profiling*.

1.3.1 Capturing data trace

If you want to trace data, you must use either:

- an automatic tracing option that includes data capture
- tracepoints that include data capture.

See also

- *Setting up new trace conditions on a running processor* on page 2-20
- *Specifying the type of information to collect for data transfer instructions* on page 3-11
- *Setting a trace range* on page 6-7
- *Setting a trace start and end point* on page 6-5
- Chapter 3 *Configuring the Conditions for Trace Capture*
- Chapter 4 *Configuring the ETM*
- Chapter 5 *Tracepoints in RealView Debugger*.

1.3.2 Capturing trace information for profiling

Profiling enables you to capture information so that you can perform a statistical analysis of your trace information.

See also

- *Setting up new trace conditions on a running processor* on page 2-20
- *Profiling trace information* on page 2-22
- *Capturing trace without using tracepoints* on page 3-4
- *Setting a trace range* on page 6-7
- *Setting a trace start and end point* on page 6-5

- Chapter 3 *Configuring the Conditions for Trace Capture*
- Chapter 4 *Configuring the ETM*
- Chapter 5 *Tracepoints in RealView Debugger*.

1.4 Tracing with RealView Debugger commands

You can use the RealView Debugger *Command-Line Interface* (CLI) to set up the conditions for capturing trace information and to analyze the captured trace information. You can also include the commands in scripts.

Where appropriate, this document shows the CLI commands you can use to capture and analyze the trace information, and how the commands relate to the equivalent GUI features. Many of the GUI operations output the equivalent CLI commands to the Output view.

——— **Note** ———

There are some operations, such as viewing profiling information, that you cannot perform at the CLI.

———————————

See also:

- the following in the *RealView Debugger Command Line Reference Guide*:
  - — *Debugger commands listed by function* on page 2-3.

1.5 Available resources

The following sections describe the resources that are available for each tracing method:

- *Trace hardware components*
- *RealView ARMulator ISS*.

1.5.1 Trace hardware components

The resources available for tracing hardware depend on the trace hardware components available on your development platform:

- For non-CoreSight systems, trace can be captured:
  - directly from an ETM trace source using a TPA, such as RealView Trace
  - into an ETB trace sink from an ETM trace source.
- For systems with a CoreSight ETM, trace can be captured:
  - directly from the CoreSight ETM trace source using a TPA, such as RealView Trace
  - into a CoreSight ETB trace sink from the CoreSight ETM trace source.

You can set unconditional and conditional tracepoints depending on the resources available for the target you are tracing:

- trigger
- trace start point
- trace end point
- trace range (Include or Exclude)
- combination of trace ranges and trace start and end points
- ExternalOut points of an ETM.

See also

- Chapter 6 *Setting Unconditional Tracepoints*
- Chapter 7 *Setting Conditional Tracepoints*
- Appendix A *Setting up the Trace Hardware*
- Appendix B *Setting up the Trace Software*
- the following documents, which you can obtain from the ARM website:
  - *Embedded Trace Macrocell Architecture Specification*
  - *Technical Reference Manual* (TRM) for your ETM version.

1.5.2 RealView ARMulator ISS

If you are using RVISS, basic instruction and data tracing support is available. You can set only unconditional tracepoints on RVISS, which can be any of the following:

- trigger
- trace start point (instructions only)
- trace start point (instructions and data)
- trace end point.

Note

RVISS does not model the ETM.

See also

- Chapter 6 *Setting Unconditional Tracepoints*
- Chapter 7 *Setting Conditional Tracepoints*.

1.6 Using the examples

As an introduction to the RealView Debugger tracing and profiling features, see Chapter 10 *Tracing Tutorial*. These examples assume that you are familiar with RealView Debugger, and demonstrate the benefits of using trace and profiling to solve typical development problems.

Chapter 2
Getting Started with Tracing

This chapter gives an overview of how trace hardware components operate together to enable you to perform tracing with RealView® Debugger, and describes the general procedure for performing tracing on your application after you have configured your system. It contains the following sections:

2.1 Preparations for tracing

Before you can use the trace features in RealView Debugger, you must:

1. Make sure that you have the required system components to perform tracing.
2. Make sure that RealView Debugger is properly connected to your target:
    - If you are using a simulator instead of trace hardware, make sure that your connection is correctly configured.
    - If you are using trace hardware, you must first ensure that the components are configured properly and connected.

See also:

- *Requirements for tracing* on page 1-3
- Appendix A *Setting up the Trace Hardware*
- Appendix B *Setting up the Trace Software*.

2.2 About the trace hardware setup

The following sections describe the hardware elements that provide the trace capability:

- *ASIC that supports trace*
- *Trace Port Analyzer* on page 2-7
- *JTAG interface unit* on page 2-7.

2.2.1 ASIC that supports trace

You must have a target system that includes either an ARM® architecture-based processor or XScale™ processor.

ARM processor with an ETM

For ARM processors that contain EmbeddedICE® logic and *Embedded Trace Macrocell*™ (ETM™), you can capture trace directly from the ETM. Figure 2-1 shows an example of how RealView Debugger can be joined with ARM ETM-enabled hardware to enable tracing capabilities.

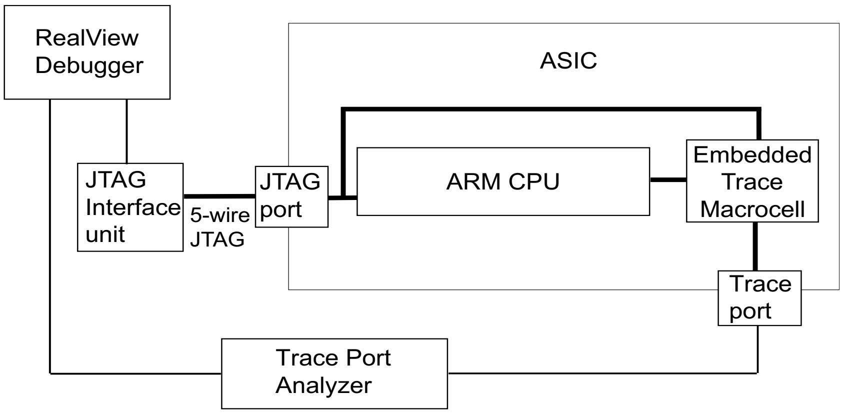


Figure 2-1 Trace hardware setup with ETM

The ETM monitors the ARM processor buses, and passes compressed information through the trace port to the *Trace Port Analyzer* (TPA). To detect sequences of events performed by the processor, the on-chip ETM contains a number of resources that are selected when the ASIC is designed. These resources comprise the trigger and filter logic you utilize within RealView Debugger.

——— **Note** ———

This is the default setup expected by RealView Debugger for development platforms that do not contain CoreSight™ components.

ARM processor with an ETM and ETB

For ARM processors that contain EmbeddedICE logic, an *Embedded Trace Buffer*™ (ETB™), and an ETM, you can view the trace captured by the ETB. Figure 2-2 shows an example of how RealView Debugger can be joined with ARM ETB-enabled hardware to enable tracing capabilities.

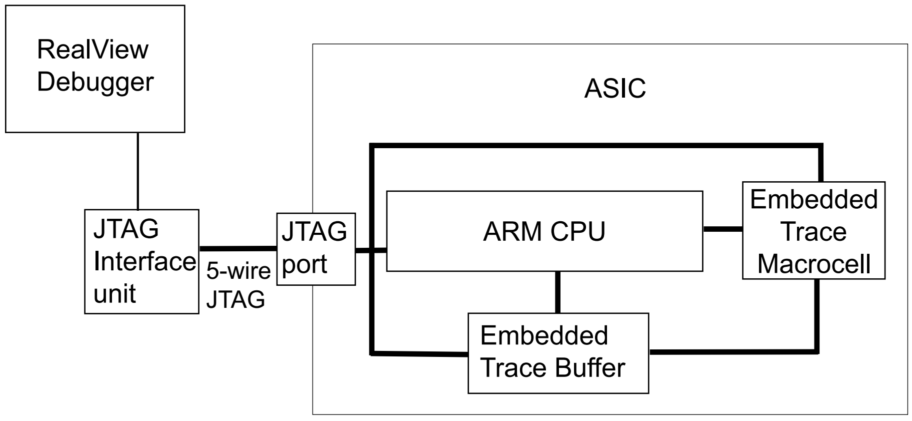


Figure 2-2 Trace hardware setup with ETB

The ETB stores data produced by the ETM. This is required for processors that run at higher speeds than can be supported by an external TPA. Therefore, trace information can be captured in real-time, and accessed by RealView Debugger at a reduced clock rate.

For details on how a processor with an ETB interacts with other trace hardware elements, see the documentation that accompanies the processor.

——— **Note** ———

Tracing an ASIC using an ETB does not require a TPA such as RealView Trace.

———————————

ARM processor that supports CoreSight ETM

For ARM processors that support the CoreSight architecture, trace capture is available from the CoreSight ETM (CSETM). Figure 2-3 shows an example of how RealView Debugger can be joined with ARM hardware that supports CoreSight to enable tracing capabilities using the following CoreSight components:

- CSETM as the trace source
- Trace Funnel (CSTFunnel), if the system has multiple trace sources
- *Trace Port Interface Unit* (TPIU) as the trace sink with an off-chip trace port.

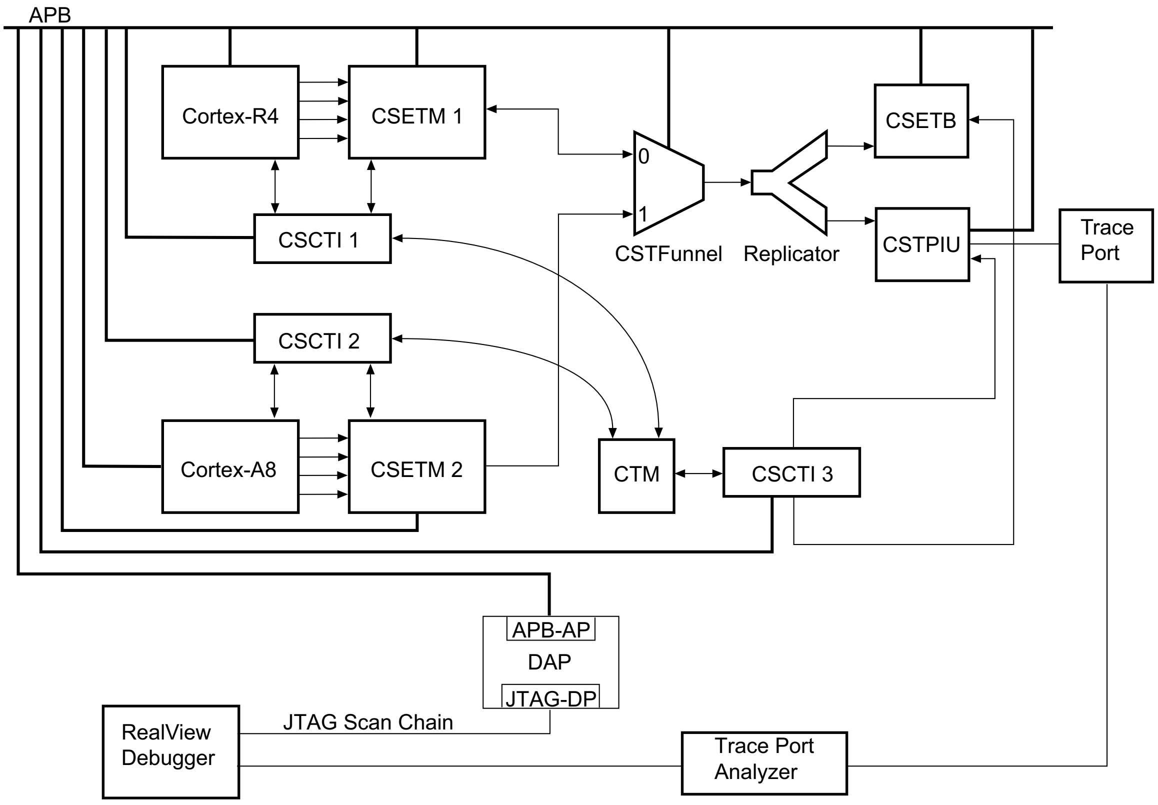


Figure 2-3 Trace hardware setup with CoreSight ETM

ARM processor with a CoreSight ETM and CoreSight ETB

For ARM processors that support the CoreSight architecture, trace capture is available through the CoreSight ETB (CSETB). Figure 2-4 shows an example of how RealView Debugger can be joined with ARM hardware that supports CoreSight to enable tracing capabilities using the following CoreSight components:

- CSETM as the trace source (only one source can be selected at a time in this release)
- CSETB as the trace sink
- Trace Funnel (CSTFunnel), if the system has multiple trace sources.

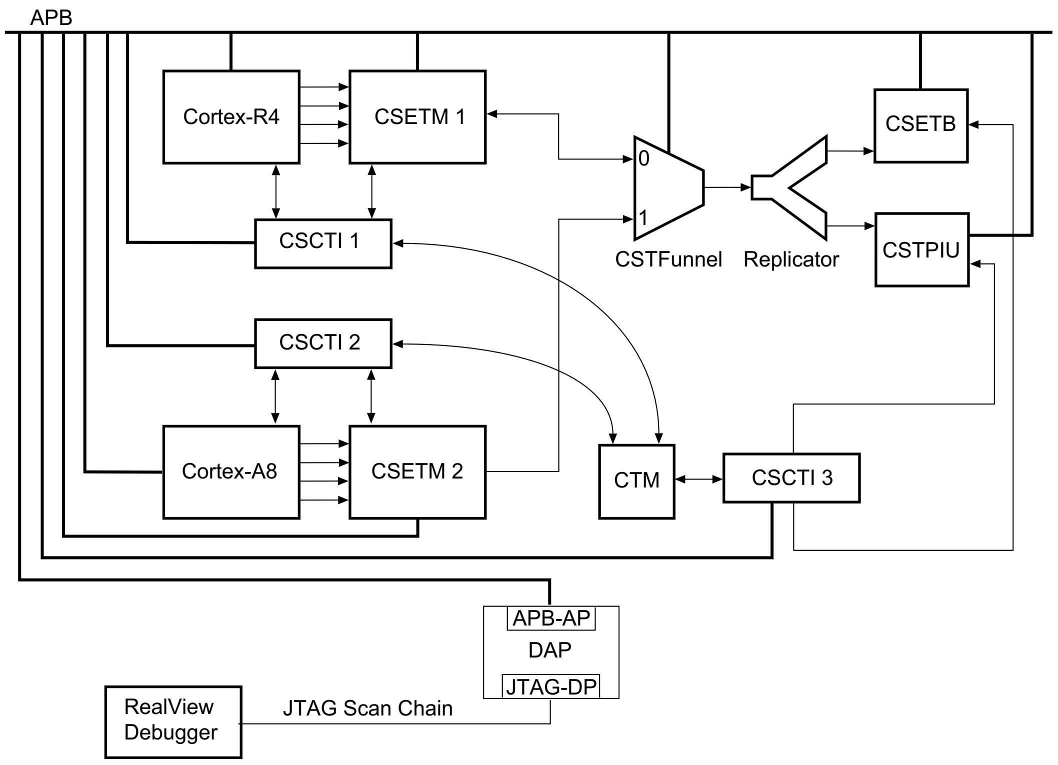


Figure 2-4 Trace hardware setup with CoreSight ETB

XScale

This is a processor based on the Intel XScale Microarchitecture processor. The XScale processor can only trace instructions, and does not trace data.

See also

- *Requirements for tracing* on page 1-3
- Chapter 4 *Configuring the ETM*
- *Configuring trace capture from an ETB with DSTREAM or RealView ICE* on page B-3
- *Configuring trace capture from a CoreSight ETM with RealView Trace* on page B-5
- *Configuring trace capture from a CoreSight ETB with DSTREAM and RealView ICE* on page B-7
- *Configuring trace capture from a CoreSight ETB with DSTREAM and RealView ICE* on page B-7.

2.2.2 Trace Port Analyzer

A *Trace Port Analyzer* (TPA) is an external device that stores the information from the trace port. The RealView Trace unit is an example of a TPA.

Note

An external TPA is not required for on-chip trace. You can use the RealView ICE unit for on-chip tracing.

2.2.3 JTAG interface unit

This component is a protocol converter that converts low-level commands from RealView Debugger into JTAG signals to the EmbeddedICE logic and the ETM.

2.3 About the Analysis window

The Analysis window enables you to capture trace from a development platform that supports an ETM and, optionally, an ETB. You can then use the Analysis window to:

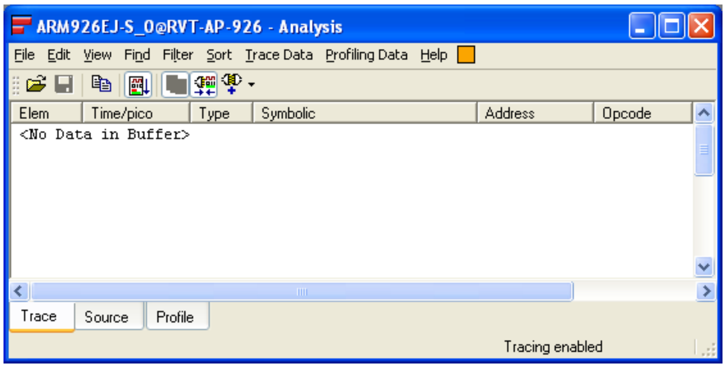
- view trace data, source, and profiling information using tabbed views
- filter the results of a trace capture
- search for a specific item of trace information
- manipulate the display of trace information.

The following sections give an overview of the Analysis window:

- *Displaying the Analysis window*
- *Equivalent CLI commands for various Analysis window features*.

2.3.1 Displaying the Analysis window

Select **Analysis Window** from the **View** menu of the Code window to display the Analysis window. Figure 2-5 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Elem Time/pico Type Symbolic Address Opcode
Trace Source Profile

<No Data in Buffer>

Tracing enabled

Figure 2-5 Analysis window

2.3.2 Equivalent CLI commands for various Analysis window features

Many of the features in the Analysis window can be performed using the following CLI commands:

- `ANALYZER`, to configure the trace logic analyzer
- `ETM_CONFIG`, to configure the ETM
- `TRACEBUFFER`, to save, load, search, and filter the trace data.

——— **Note** ———

For some features, such as profiling, there is no equivalent CLI command.

———————————

See also

- Chapter 9 *Analyzing Trace with the Analysis Window*
- *Mapping Analysis window options to CLI commands and qualifiers* on page 9-49
- the following in the *RealView Debugger Command Line Reference Guide*:
  — *Alphabetical command reference* on page 2-12 for details of the `ANALYZER`, `ETM_CONFIG`, and `TRACEBUFFER` commands.

2.4 Basic RealView Debugger tracing procedure

It is recommended that you familiarize yourself with the tracing procedure before trying to perform trace capture on your own application using RealView Debugger. The procedure depends on the target hardware.

The following sections describe this task:

- *Before you start*
- *Basic procedure for hardware that does not support CoreSight*
- *Basic procedure for hardware that supports CoreSight* on page 2-10.

2.4.1 Before you start

Before you can begin tracing with RealView Debugger:

1. Make sure that your target supports trace.
2. Set up your trace hardware.
3. Start RealView Debugger and ensure that your trace software is properly configured.

Note

You can perform non-stop tracing on a running processor.

See also

- *Requirements for tracing* on page 1-3
- Appendix A *Setting up the Trace Hardware*
- Appendix B *Setting up the Trace Software*.

2.4.2 Basic procedure for hardware that does not support CoreSight

A typical procedure for tracing your application is as follows:

1. Connect to the target.
2. Connect to the logic analyzer:
   a. Select **Analysis Window** from the **View** menu to display the Analysis window. Figure 2-5 on page 2-8 shows an example.
   b. Click **Toggle Analyzer** on the Analysis window toolbar.
3. Load the image for the application you want to trace.
4. Skip the next two steps if you want to use the default tracing configuration.
5. If you want to configure specific tracing features, change the general tracing configuration options as follows:
   - For ETM-enabled targets, use the Configure ETM dialog box.
   - For any target, use the options in the **Edit** menu of the Analysis window.

   Note

   The available menu options depend on your target.
6. If you want to limit trace capture to a specific area of your image, determine the area of interest in your source files or disassembly for which you want to perform tracing, and set tracepoints accordingly.

——— **Note** ———

If you do not set any tracepoints, automatic tracing occurs. That is, when you connect to an analyzer trace capture is enabled. You can disconnect from the analyzer to disable it or, for ETM-based targets, disable tracing without disconnecting from the analyzer.

———————————

7. Run your application to capture trace information. The results of the trace capture are returned to the Analysis window, where you can analyze the results.

——— **Note** ———

The availability of tracing resources and options in the Analysis window depend on your system configuration.

———————————

See also

- *Available resources* on page 1-8
- *About the Analysis window* on page 2-8
- *Enabling and disabling tracing* on page 2-19
- *Setting up new trace conditions on a running processor* on page 2-20
- *Capturing trace without using tracepoints* on page 3-4
- Chapter 3 *Configuring the Conditions for Trace Capture*
- Chapter 4 *Configuring the ETM*
- Chapter 5 *Tracepoints in RealView Debugger*
- Chapter 9 *Analyzing Trace with the Analysis Window*
- Chapter 10 *Tracing Tutorial*
- the following in the *RealView Debugger User Guide*:
  — Chapter 4 *Loading Images and binaries*.

2.4.3 Basic procedure for hardware that supports CoreSight

A typical procedure for tracing your application on hardware that support CoreSight is as follows:

1. Connect to the targets and associated CoreSight components on your development platform:
   a. In the Connect to Target window, locate the Debug Configuration for your development system.
   b. Expand the Debug Configuration name to display the associated targets and CoreSight components.
   c. Double-click on each target and the associated CoreSight components in turn to establish the connections.

   ——— **Note** ———

   You must keep the connections to the CoreSight components and target processors established during your debugging session.

   ———————————

2. Configure the CoreSight components as required:
   a. Select **Registers** from the **View** menu to display the Registers view.
   b. Set up the registers for the CoreSight component as required.
   c. Change the current connection to display the registers for the next CoreSight component.

   d. Repeat steps b and c to configure the remaining CoreSight components.

3. Make sure that the current connection is to the target processor that you are using to debug your application. That is, make sure the details for the target processor are visible in the RealView Debugger Code window.

   Note

   You can customize a Debug Configuration to define a pre-connect sequence. This sequence specifies the order in which targets are connected.

4. Connect to the logic analyzer:

   a. Select **Analysis Window** from the **View** menu to display the Analysis window. Figure 2-5 on page 2-8 shows an example.

   b. Click **Toggle Analyzer** on the Analysis window toolbar.

5. Load the image for the application you want to trace.

6. Skip the next two steps if you want to use the default tracing configuration.

7. If you want to configure specific tracing features, change the general tracing configuration options as follows:

   - For ETM-enabled targets, use the Configure ETM dialog box.
   - For any target, use the options in the **Edit** menu of the Analysis window.

     Note

     The available menu options depend on your target.

8. If you want to limit trace capture to a specific area of your image, determine the area of interest in your source files or disassembly for which you want to perform tracing, and set tracepoints accordingly.

   Note

   If you do not set any tracepoints, automatic tracing occurs. That is, when you connect to an analyzer trace capture is enabled. You can disconnect from the analyzer to disable it or, for ETM-based targets, disable tracing without disconnecting from the analyzer.

9. Run your application to capture trace information.

See also

- *Available resources* on page 1-8
- *About the Analysis window* on page 2-8
- *Enabling and disabling tracing* on page 2-19
- *Setting up new trace conditions on a running processor* on page 2-20
- *Capturing trace without using tracepoints* on page 3-4
- Chapter 3 *Configuring the Conditions for Trace Capture*
- Chapter 4 *Configuring the ETM*
- Chapter 5 *Tracepoints in RealView Debugger*
- Chapter 9 *Analyzing Trace with the Analysis Window*
- Chapter 10 *Tracing Tutorial*
- the following in the *RealView Debugger User Guide*:
  - — *Connecting to all targets for a specific Debug Configuration* on page 3-48
  - — *Changing the current target connection* on page 3-50

  - Chapter 4 *Loading Images and binaries*
- the following in the *RealView Debugger Target Configuration Guide*:
  - *Configuring a connection sequence for multiple targets* on page 3-36.

2.5 Configuring trace from a Cortex-R4 processor and a CoreSight ETM

To configure a CoreSight development platform containing a Cortex-R4 processor, a CoreSight ETM, and a TPIU:

1. Plug the JTAG cable into the trace connector and plug the trace connector into the Mictor port on the target hardware.
2. Display the RVConfig utility:
   a. In RealView Debugger, create a new `RealView ICE` Debug Configuration, or select an existing Debug Configuration.
   b. If you are modifying an existing Debug Configuration, right-click on the Debug Configuration name to display the context menu.
   c. Select **Configure** from the context menu.
      The RVConfig utility is displayed.

   If you create a new Debug Configuration, the RVConfig utility is displayed automatically.
3. In the RVConfig utility:
   a. Click **Auto Configure**.
   b. Click **Trace Associations** in the RVConfig utility to display the Trace Associations Editor.
   c. Add the following lines:
      ```
      ETM: TraceOutput1=TPIU
      TPIU: Port0=CSETM_0
      ```
      The first line specifies that trace output is going off-chip through the TPIU.
      The second line specifies that trace is coming from the CSETM into Port 0 of the TPIU.
   d. Click **OK** to close the Trace Associations Editor.
   e. Select **Save** from the **File** menu.
   f. Select **Close** from the **File** menu.
4. Connect to the Cortex-R4 target.
5. Connect to the CSETM target.
6. Connect to the CSTPIU target.
7. Set the CSTPIU_CURPORTSIZE register to the required value shown in Table 2-1.

Table 2-1 Possible values for the CSTPIU_CURPORTSIZE register

Value	Description
`0x8000`	for 16-bit wide trace
`0x0080`	for 8-bit wide trace
`0x0008`	for 4-bit wide trace.

8. Set the CSTPIU_ FFCR register to the required value shown in Table 2-2.

Table 2-2 Possible values for the CSTPIU_FFCR register

Value	Description
`0x00`	Bypass mode
`0x01`	Normal mode
`0x02`	Continuous mode

Note

For Normal and Continuous modes, you must also set ETM_CS_TRACE_ID to a value between 1 and 127.

9. If triggers are required, set bits 8 and 9 of the CSTPIU_FFCR register to one.

   Note

   If extracting Normal or Continuous trace, you must set the ETM_CS_TRACE_ID register to the ID of the processor from which trace is to be extracted. In a single processor system, set the ID to `0x01`. In a dual processor system, set the ID to `0x02` to get trace from processor 2.

10. In RealView Debugger, switch to the connection for the processor you are tracing.
11. Connect the Trace Analyzer.
12. Perform the trace capture.

Note

If you change any registers in the CSETM and CSTPIU targets, or any other CoreSight target, you must disconnect and subsequently reconnect the analyzer for these changes to be detected.

2.6 Configuring trace from a Cortex-R4 processor and a CoreSight ETB

To configure a CoreSight development platform containing a Cortex-R4 processor, an ETM, and an ETB:

1. Plug the JTAG cable directly into the target hardware. Do not use the trace connector.
2. Display the RVConfig utility:
   a. In RealView Debugger, create a new `RealView ICE` Debug Configuration, or select an existing Debug Configuration.
   b. If you are modifying an existing Debug Configuration, right-click on the Debug Configuration name to display the context menu.
   c. Select **Configure** from the context menu.
      The RVConfig utility is displayed.

   If you create a new Debug Configuration, the RVConfig utility is displayed automatically.
3. In the RVConfig utility:
   a. Click **Auto Configure**.
   b. Select the Cortex-R4 processor in the block diagram.
   c. Click **Properties...** to display the Device Properties dialog box.
   d. Select **Embedded Trace Buffer (ETB)**.
   e. Click **OK** to close the Device Properties dialog box.
   f. Select **Save** from the **File** menu.
   g. Select **Close** from the **File** menu.
4. Connect to the Cortex-R4 target.
5. Connect to the CSETM target.
6. Connect to the CSETB target.
7. Set the CSETB_ FFCR register to the required value shown in Table 2-3.

Table 2-3 Possible values for the CSTPIU_FFCR register

Value	Description
`0x00`	Bypass mode
`0x01`	Normal mode
`0x02`	Continuous mode

——— **Note** ———

For Normal and Continuous modes, you must also set ETM_CS_TRACE_ID to a value between 1 and 127.

8. In RealView Debugger, switch to the connection for the processor you are tracing.
9. Connect the Trace Analyzer.
10. Perform the trace capture.

——— **Note** ———

If you change any registers in the CoreSight ETM and TPIU targets, or any other CoreSight target, you must disconnect and subsequently reconnect the analyzer for these changes to be detected.

————————————

2.7 Connecting to an analyzer

Before you can capture trace information, you must connect RealView Debugger to the trace analyzer:

- for a hardware target, this can be an external TPA, an on-chip trace buffer, or JTAG solution
- for *RealView ARMulator® ISS* (RVISS) targets, trace is captured directly by RealView Debugger.

To connect to an analyzer:

1. Connect to the target.
2. Select **Analysis Window** from the **View** menu to display the Analysis window.
3. Select **Connect/Disconnect Analyzer** from the Analysis window **Edit** menu.

   The icon to the left of the option toggles to show that the analyzer is connected. By default, tracing is also enabled when you connect.

See also:
- *Trace hardware* on page 1-3
- *Simulators* on page 1-4
- *Disconnecting from the Analyzer* on page 2-18
- *Enabling and disabling tracing* on page 2-19
- *Configuring automatic tracing* on page 3-11.

2.8 Disconnecting from the Analyzer

To disconnect from the analyzer, select **Connect/Disconnect Analyzer** from the Analysis window **Edit** menu. The icon to the left of the option toggles to show that the analyzer is connected. Also, the Analysis window status bar changes to `Not connected`.

Disconnecting from the analyzer also disables tracing. For ETM-based targets, you can also disable and enable tracing and remain connected to the analyzer.

See also:

- *Enabling and disabling tracing* on page 2-19.

2.9 Enabling and disabling tracing

By default, tracing is enabled when you first connect to the analyzer. For ETM-based targets, you can disable and enable tracing and remain connected to the analyzer.

To globally disable or enable tracing, either:

- Select **Tracing Enabled** from the **Edit** menu.

  The icon to the left of the option toggles to show the tracing state, and the Analysis window status bar also changes.
- Click **Enable/Disable** to globally enable or disable tracing.

If you stop tracing during image execution, all captured information up to that point is returned to the Analysis window. You can select this option again to restart tracing at any time during image execution.

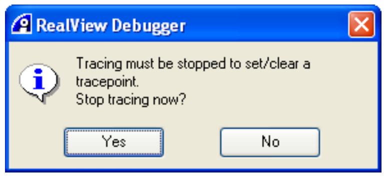
See also:

- *Disconnecting from the Analyzer* on page 2-18.

2.10 Setting up new trace conditions on a running processor

If you have a processor that is already running, you can set up new trace conditions on that processor without having to stop it. When you attempt to configure new trace conditions on a running processor, RealView Debugger displays one of the following prompts:

- When setting or clearing tracepoints, RealView Debugger displays the Prompt dialog box shown in Figure 2-6.


RealView Debugger

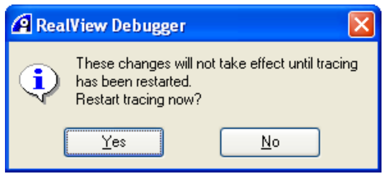
Tracing must be stopped to set/clear a tracepoint.
Stop tracing now?

Figure 2-6 Prompt displayed on a running target (tracepoints)

  If you click **Yes**, RealView Debugger disables tracing, sets or clears the tracepoint as required, and updates the trace buffer.

  To enable tracing again, select **Tracing Enabled** from the Analysis window **Edit** menu.

- When configuring the global trace conditions, RealView Debugger displays the Prompt dialog box shown in Figure 2-7.


RealView Debugger

These changes will not take effect until tracing has been restarted.
Restart tracing now?

Figure 2-7 Prompt displayed on a running target (global trace conditions)

  If you click **Yes**, RealView Debugger temporarily disables tracing, sets up the global trace condition that you requested, and enables tracing again.

See also:

- Chapter 3 *Configuring the Conditions for Trace Capture*.

2.11 Capturing trace output

To capture trace output:

1. Select **Analysis Window** from the **View** menu of the Code window to display the Analysis window.
2. Connect to the analyzer.
3. Configure your trace capture requirements.

   ——— **Note** ———

   RealView Debugger automatically enables the tracing of instructions when you connect to the analyzer, and traces all instructions executed during image execution. Alternatively, you can change the automatic tracing mode, or set tracepoints to limit the trace capture to a specific region of execution.

   ————————
4. If required, you can force the captured trace to be displayed by setting a trace trigger point, which does not stop processor execution.
5. Click **Run** on the Debug toolbar to run your application.

See also:
- *Displaying captured trace on a stop condition*
- *Connecting to an analyzer* on page 2-17
- *Trigger* on page 5-3
- Chapter 3 *Configuring the Conditions for Trace Capture*
- Chapter 9 *Analyzing Trace with the Analysis Window*
- Chapter 10 *Tracing Tutorial*
- the following in the *RealView Debugger User Guide*:
  - Chapter 8 *Executing Images*
  - Chapter 12 *Controlling the Behavior of Breakpoints*.

2.11.1 Displaying captured trace on a stop condition

When the processor execution stops, any captured trace is displayed in the Analysis window. The processor stops if you:
- set a breakpoint that stops the processor
- manually stop execution
- force the processor to run until a specific point is reached.

Captured trace is also displayed if the processor stops because the application terminated either by a fault condition or by exiting normally.

2.12 Profiling trace information

In addition to analyzing the trace information directly, you can perform a statistical analysis of your trace information. This enables you to display details such as the amount of time your application spends executing certain functions as a percentage of the overall execution time. You can also display the information graphically as a histogram.

The following sections describe how to perform profiling on different type of target:

- *Profiling with ETM-enabled hardware*
- *Profiling with RVISS* on page 2-23.

2.12.1 Profiling with ETM-enabled hardware

For ETM-enabled hardware, you can obtain profiling information using any of the following features:

Timestamping

Analyzing timestamp values enables you to see, for example, when pauses have occurred in processor execution, and how long it takes between successive invocations of a particular section of code.

With timestamping enabled, your TPA adds a timestamp value to each line of traced information. Trace information is time stamped at a resolution of 10ns on RealView Trace.

——— **Note** ———

Timestamping is not available when tracing with an ETB.

———

Cycle-accurate tracing

When profiling the execution of critical code sequences, it is often useful if you can observe the exact number of cycles that a particular code sequence takes to execute.

Cycle-accurate tracing causes the TPA to capture trace on all cycles, even if there is no trace to output on that cycle. Therefore, you can determine the number of cycles taken by a region of code by counting the number of cycles of traced capture.

——— **Note** ———

For ETMs earlier than ETMv3, cycle-accurate tracing is disabled when a trace discontinuity occurs (for example, if the processor enters debug state).

———

Instruction count-based

If you have an ETMv3 processor, then by default profiling is based on instruction count-based information. However, timestamping and cycle-accurate tracing overrides this.

——— **Note** ———

For ETMv3, timestamping and cycle-accurate tracing are mutually exclusive.

———

See also

- *About the Analysis window* on page 2-8
- Chapter 4 *Configuring the ETM*.

2.12.2 Profiling with RVISS

If you are using a simulator, such as RVISS, then RealView Debugger uses the cycle count from the trace information for profiling.

RealView Debugger displays profiling information in the Profile tab of the Analysis window.

See also

- *About the Analysis window* on page 2-8.

Chapter 3
Configuring the Conditions for Trace Capture

This chapter describes how to configure the conditions for trace capture. It contains the following sections:

3.1 Setting up the global conditions for trace capture

You can set global conditions for trace capture, which depend on your debug target. You can access these configuration settings from the **Edit** menu of the Analysis window.

For targets other than those indicated, you can:

- Change the size of the trace buffer.
- Choose how trace information is to be captured around a trigger tracepoint (except for on chip trace buffers).
- What trace information to capture when no tracepoints are set.

In addition, you can also configure the following:

- For targets with *Embedded Trace Macrocell*™ (ETM™), you can:
  - Choose to trace data only (ETMv3 only). You must set tracepoints with this option.
  - Choose to stop the processor when a trigger occurs.
  - Specify what information to trace for data transfer instructions.
  - Configure the ETM.
- For *RealView® ARMulator® ISS* (RVISS), you can:
  - choose to capture and return control-flow information only (branches)
  - specify what happens when the trace buffer is full.

Other options might be available if your target supports them.

See also:

- *Capturing trace without using tracepoints* on page 3-4
- *Configuring common trace options* on page 3-8
- Chapter 4 *Configuring the ETM*.

3.2 Setting up the conditions for profiling captured trace

For RVISS simulated targets profiling information is automatically generated from the cycle count during trace capture.

For ETM-based targets, select either of the following settings:

- **Enable Timestamping**
- **Cycle accurate tracing**.

If both options are selected, then profiling information is based on cycle counts.

——— **Note** ———

For ETMv3, instruction counts are automatically used for profiling.

————————

See also:

- *Profiling trace information* on page 2-22
- *Enable Timestamping* on page 4-10
- *Cycle accurate tracing* on page 4-11.

3.3 Capturing trace without using tracepoints

If you do not set any tracepoints, then RealView Debugger automatically traces all instructions executed. What is captured is determined by the Automatic Tracing Mode, which depends on your target:

- Instructions only (the default)
- Data only (ETMv3 only)
- Instructions and data.

——— **Note** ———

If you turn off Automatic Tracing Mode, you must use tracepoints to capture trace information.

————————

See also:

- *Configuring automatic tracing* on page 3-11.

3.4 Capturing trace with tracepoints

If you want to limit tracing to specific areas of your image, then RealView Debugger enables you to set a variety of tracepoints. Therefore, you can control the amount and content of application information that is traced. You can set a tracepoint on any of the following:

- a line or range of source code
- a line or range of disassembly code
- a function name
- a function entry point
- a memory address or range of memory addresses.

After you have configured the details for trace capture and executed your application, the captured information is returned to the Analysis window. From there, you can perform additional filtering by narrowing the results of the capture.

See also:

- *Filtering information in captured trace* on page 9-22
- Chapter 5 *Tracepoints in RealView Debugger*.

3.5 Setting tracepoints

To set unconditional or conditional tracepoints:

1. Ensure that you are connected to your target, and that the trace analyzer is connected.
2. Ensure that your image is loaded into RealView Debugger.
3. Determine the area of interest within your application for which you want to collect trace information. Depending on the area of interest, you must give focus to your application within one of the following RealView Debugger Code window views:

   Source view

   Select the required source file tab if you want to set tracepoints for specific lines of code in your source files.

   Disassembly view

   Select the **Disassembly** tab if you want to set tracepoints at specific addresses or address ranges in your image.

4. Choose from the following types of trace-capture setting types available, depending on the complexity of the criteria you want to specify for the capture:

   Unconditional tracepoints

   For setting trigger points, trace start and end points, or trace ranges for memory and data accesses.

   Conditional tracepoints

   For setting AND or OR conditions, counter conditions, and complex comparisons. These conditions can involve any supported combination of trigger points and ranges.

5. Select **Break/Tracepoints** from the Code window **View** menu to display the Break/Tracepoints view.

   The Break/Tracepoints view enables you to:

   - view tracepoint details, including the CLI command that was issued either by you or RealView Debugger to set the tracepoint
   - edit tracepoints
   - locate tracepoints in the source or disassembly view.

6. Run your image to begin trace capture with the details you have set.

——— **Note** ———

The availability of tracepoint options depends on the resources you have available and the type of connection you are using. If you have already used a number of resources, some of the options described in this section might not all be available to you.

———————

See also:

- *Capturing trace output* on page 2-21
- Chapter 6 *Setting Unconditional Tracepoints*
- Chapter 7 *Setting Conditional Tracepoints*
- Chapter 8 *Managing Tracepoints*
- the following in the *RealView Debugger User Guide*:
  — Chapter 8 *Executing Images*.

3.6 Setting tracepoints on exception vectors

If you want to trace an exception vector, such as Data Abort, you must disable the vector catch for that exception before running the image. This prevents the exception being caught, and the exception can be recorded in the trace buffer. If the processor is tracing and an exception occurs, the analysis window displays the text in Table 3-1 as appropriate.

Table 3-1 Text recorded in trace buffer for exceptions

Exception	Text recorded
Reset	`RESET`
Undefined instruction	`UNDEF EXCEPTION`
Supervisor Call (SVC) interrupt	`Branch to <SVC>`
Prefetch Abort (instruction memory read abort)	`PREFETCH ABORT`
Data Abort (data memory read or write abort)	`DATA ABORT`
IRQ (normal interrupt)	`INTERRUPTED`
FIR (fast interrupt)	`FAST INTERRUPT`
Secure Monitor Call (SMC)[a]	`SECURE MONITOR CALL`

a. This is supported only on ETMv3.2, or later.

You can configure vector catching using one of the following methods:

- Configure the connection settings in the `Vectors` group of the `Advanced_Information` block for the connection. This method persists between debugging sessions.
- Select **Processor Exceptions...** from the Code window **Debug** menu to configure the processor exceptions. The changes are lost when you disconnect.

See also:

- the following in the *RealView Debugger User Guide*:
  — *Specifying processor exceptions (global breakpoints)* on page 11-65
- the following in the *RealView Debugger Target Configuration Guide*:
  — *Debug Configuration Advanced_Information settings reference* on page A-10.

3.7 Configuring common trace options

This section describes how to set trace configuration options that are common to all trace captures you perform. It includes:

- *Configuring analyzer settings for ETM-enabled targets*
- *Setting the size of the trace buffer (RVISS targets only)* on page 3-9
- *Storing only control-flow changes* on page 3-9
- *Configuring the behavior when the trace buffer is full* on page 3-9
- *Configuring how trace information is collected at a trigger* on page 3-10
- *Specifying the type of information to collect for data transfer instructions* on page 3-11
- *Configuring automatic tracing* on page 3-11
- *Setting and Editing Event Triggers* on page 3-12
- *Clearing All Event Triggers* on page 3-12
- *Configuring the address and signal controls of your target processor* on page 3-12.

The availability of some of these options, and any dialogs that might be displayed when you select them, depend on your *Trace Port Analyzer* (TPA), your target processor, and the state of some configuration options.

The check marks next to some options in the **Edit** menu indicate the default settings, which vary depending on the hardware you are using.

——— **Note** ———

Make sure that RealView Debugger has finished updating the trace buffer before you set a trace configuration option.

———————————

See also:

- *Connecting to an analyzer* on page 2-17
- *Capturing trace with tracepoints* on page 3-5
- *Data only trace (Do not trace instructions) (ETMv3.1 and later)* on page 4-11
- *Trigger* on page 5-3
- Chapter 4 *Configuring the ETM*
- Chapter 5 *Tracepoints in RealView Debugger*.

3.7.1 Configuring analyzer settings for ETM-enabled targets

To configure analyzer settings:

1. Display the Analysis window.
2. Connect to an analyzer.
3. Select **Configure Analyzer Properties...** from the **Edit** menu to display the Configure ETM dialog box.

3.7.2 Setting the size of the trace buffer (RVISS targets only)

If you are capturing trace on a RVISS target, and the trace appears to be incomplete, then you might have to increase the size of the trace buffer.

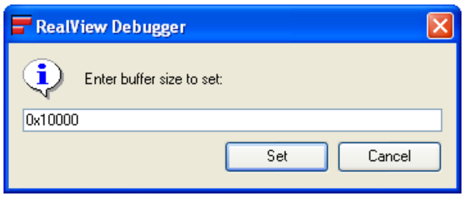
——— **Note** ———

It is recommended that you do not change the size of the trace buffer when debugging hardware targets.

————————

To set the size of the trace buffer for a RVISS target:

1. Select **Set Trace Buffer Size...** from the **Edit** menu to display the Prompt dialog box. Figure 3-1 shows an example:



Enter buffer size to set:

0x10000

Figure 3-1 Setting the size of the trace buffer

2. Enter the required maximum buffer size, in decimal or hexadecimal.

3. Click **Set**.

3.7.3 Storing only control-flow changes

To capture and return control-flow information (branches), select **Store Control-flow Changes Only** from the **Edit** menu.

This ensures that only control-flow branches are stored in the trace buffer. It reduces the amount of information stored in the buffer during a trace capture session. This is especially useful when you are interested in capturing profiling information (displayed in the **Profile** tab) to analyze branching information only.

——— **Note** ———

This option is not available for ETM targets.

————————

3.7.4 Configuring the behavior when the trace buffer is full

To configure the behavior of RealView Debugger when the trace buffer is full:

1. Select **Buffer Full Mode** from the **Edit** menu.

2. Select the required option from the submenu:

   Stop Processor on Buffer Full

   This option causes the target processor to stop executing when the trace buffer becomes full.

   Stop Collecting on Buffer Full

   This option stops the collection of trace information when the trace buffer becomes full. The target processor is not stopped.

   If you want the processor to stop whenever the trace buffer becomes full, you must select the option **Stop Processor on Buffer Full**.

Continue Collecting on Buffer Full

This option forces the collection of trace information to continue while the processor is running, even after the trace buffer becomes full. In this case, older information is cleared from the buffer as new information enters it.

——— **Note** ———

This option is overridden if you have specified a trigger and the trigger is reached.

———————————

——— **Note** ———

The ETM uses a circular buffer in the hardware. Therefore, **Continue Collecting on Buffer Full** is the only mode that is available.

———————————

3.7.5 Configuring how trace information is collected at a trigger

You can configure whether trace information is collected before, around or after a trigger. The position of the trigger might be approximate, depending on the characteristics of your TPA. Also, the trigger itself might or might not be present in the resulting trace.

A trigger has no impact on the output from the ETM, other than the signal that it has occurred. It is interpreted by the TPA to determine what additional information to capture and when to dump its contents back to the debugger. You might want to use a trigger:

- for non-stop tracing, where the captured trace is dumped to the debugger when the trigger is hit
- to collect the trace near a point of interest, or when a specific condition exists.

——— **Note** ———

For ETM-based targets, triggers are accurate to within an instruction or so, even for ETMv3.

———————————

To configure how trace is collected at a trigger:

1. Select **Trigger Mode** from the **Edit** menu.
2. Select the required option from the submenu:

   Collect Trace Before Trigger

   This option is the default. It forces the capture of all trace information prior to the trigger position.

   Collect Trace Around Trigger

   This option causes the capture of all trace information around the trigger position.

   If you do not specify trace start and end points, then half of the trace information prior to reaching the trigger, and half of the trace information after reaching the trigger, is captured.

   If you have placed the trigger between trace start and end points, then the amount of trace information collected before and after the trigger depends on the position of the trigger relative to the trace start and end points.

   Collect Trace After Trigger

   This option causes the capture of all trace information after the trigger position.
3. If you want to stop the target processor when the trigger is hit, select **Edit → Trigger Mode → Stop Processor on Trigger**.

Note

This option is available only for ETM-based targets.

3.7.6 Specifying the type of information to collect for data transfer instructions

To specify the type of information to collect for data transfer instructions:

1. Select **Data Tracing Mode** from the **Edit** menu.
2. Select the required option from the submenu:

Address Only	Use this option when you want the data transfer address, but no value(s) transferred. This option is the default.
Data Only	Use this option when you want the value(s) transferred, but not the data transfer address.
Data and Address	Use this option when you require both the data transfer address and the value(s) transferred.

Note

These options are available only for ETM-based targets.

Be aware of the following:

- Any data tracing requires more information to be stored, and so increases the bandwidth.
- These options do not enable data tracing, but only select what type of information is stored when data trace is enabled. You can instruct RealView Debugger to capture addresses, data, or both in the following ways:
  - Select the appropriate Automatic Tracing Mode.
  - Set the required tracepoint type for individual tracepoints as you set them. This overrides the Automatic Tracing Mode option you have selected.

3.7.7 Configuring automatic tracing

To collect trace information without having to configure any tracepoints:

1. Select **Automatic Tracing Mode** from the **Edit** menu.
2. Select the required option from the submenu:

Off (Use tracepoints)

This option disables Automatic Tracing Mode. When this option is selected, you must use tracepoints to capture trace information.

Instructions Only

This option is the default. RealView Debugger automatically traces on instructions.

Data Only

This option is available only for ETMv3 targets, and is grayed out for all other targets. It instructs RealView Debugger to automatically trace on data only.

Instructions and Data

This option instructs RealView Debugger to automatically trace on both instructions and data.

When an automatic tracing mode is set, you only have to execute your application image to generate trace information. When the processor stops, either by itself or by manual intervention, the trace information currently in the buffer is returned to the Analysis window.

——— **Note** ———

When you set one or more tracepoints, automatic tracing is overridden and trace information is captured according to the tracepoints.

———————————

3.7.8 Setting and Editing Event Triggers

The **Set/Edit Event Triggers...** option on the **Edit** menu is not supported in this release.

3.7.9 Clearing All Event Triggers

The **Clear All Event Triggers** option on the **Edit** menu is not supported in this release.

3.7.10 Configuring the address and signal controls of your target processor

The **Physical to Logical Address Mapping** option on the **Edit** menu is not supported in this release.

Chapter 4
Configuring the ETM

This chapter describes how to configure the *Embedded Trace Macrocell*™ (ETM™) for capturing trace on ETM-enabled hardware. It includes:

——— **Note** ———

The methods described in this chapter also apply to configuring the CoreSight™ ETM, if supported by your development platform. However, you must also configure other CoreSight components as described in *Configuring trace capture from a CoreSight ETM with RealView Trace* on page B-5.

4.1 About configuring the ETM

The facilities provided by the ETM depend on design choices made by the manufacturer. Also, some features used by RealView® Debugger depend on the support provided by the *Trace Port Analyzer* (TPA) you are using. RealView Debugger automatically detects which facilities are implemented by the ETM and the TPA.

RealView Debugger enables or disables ETM configuration settings depending on whether they are available on your target. Where a value is fixed because of a target-specific restriction, the value is displayed in the Configure ETM dialog box, but the setting to amend it is disabled. However, there are facilities that might not be detected, and there are some optional features that cannot be autodetected. For example, if the `FIFOFULL` logic is present but not connected to the processor, the ETM does report that `FIFOFULL` logic is present, but the `FIFOFULL` logic does not operate.

You use the Configure ETM dialog box to configure the ETM behavior. The settings you configure in this dialog box apply to any tracing you perform, whatever trace capture criteria you set and whatever the number of trace captures you perform

See also:

- Chapter 3 *Configuring the Conditions for Trace Capture*.

4.1.1 ETM with Lock Access implemented causes APB lock up

If an ETM is powered-down and Locked (that is, the default state out of reset if the Lock Access mechanism is enabled), then the write to the ETM Control register fails as expected. However, when RealView Debugger attempts to read and display all of the ETM registers, this causes the APB to lock up.

To prevent the APB from locking up, set the `RVD_CS_ETM_LOCK` environment variable to the hexadecimal value that represents the password required to unlock the ETM hardware. The value is written to the `CS_LOCKACCESS` register on connect before the ETM is powered up.

See also

- Chapter 3 *Configuring the Conditions for Trace Capture*.

4.2 Displaying the Configure ETM dialog box

To display the Configure ETM dialog box, select either:

- **Configure Analyzer Properties...** from the Analysis window **Edit** menu.
- **Tools** → **Analyzer/Trace Control** → **Configure Analyzer Properties...** from a Code window.

——— **Note** ———

These menu options are not available for simulators, such as *RealView ARMulator® ISS* (RVISS).

————————

After you have configured the ETM, click:

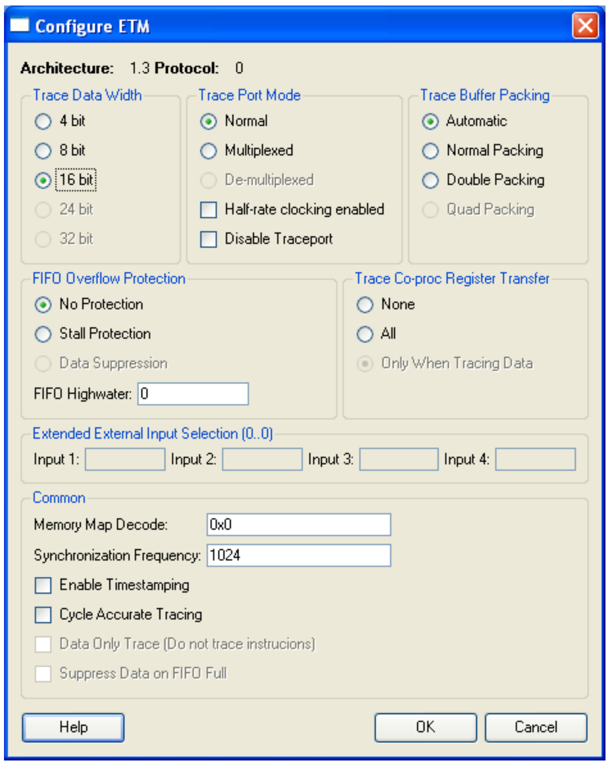
- **OK** to save the configuration
- **Cancel** to discard any changes.

See also:

- *Configure ETM dialog box for ETMv1*
- *Configure ETM dialog box for ETMv3* on page 4-4.

4.2.1 Configure ETM dialog box for ETMv1

The appearance of the Configure ETM dialog box varies depending on the ETM architecture you are using. For ETMv1 architectures, the Configure ETM dialog box appears as shown in Figure 4-1. The ETM architecture and protocol versions are displayed at the top of the Configure ETM dialog box.



Configure ETM

Architecture: 1.3 Protocol: 0

Trace Data Width: 4 bit, 8 bit, [✓ 16 bit], 24 bit, 32 bit

Trace Port Mode: [✓ Normal], Multiplexed, De-multiplexed; ☐ Half-rate clocking enabled; ☐ Disable Traceport

Trace Buffer Packing: [✓ Automatic], Normal Packing, Double Packing, Quad Packing

FIFO Overflow Protection: [✓ No Protection], Stall Protection, Data Suppression; FIFO Highwater: 0

Trace Co-proc Register Transfer: None, All, [✓ Only When Tracing Data]

Extended External Input Selection (0..0): Input 1: Input 2: Input 3: Input 4:

Common:

Memory Map Decode: 0x0

Synchronization Frequency: 1024

☐ Enable Timestamping

☐ Cycle Accurate Tracing

☐ Data Only Trace (Do not trace instrucions)

☐ Suppress Data on FIFO Full

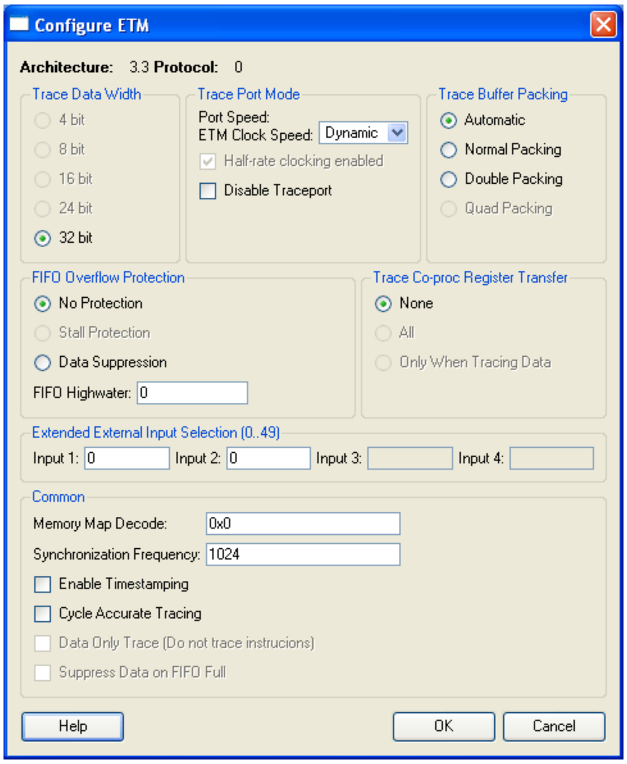
Figure 4-1 The Configure ETM dialog box for ETMv1

See also

- *Configuring the ETM parameters* on page 4-5.

4.2.2 Configure ETM dialog box for ETMv3

For ETMv3 architectures, the Configure ETM appears as shown in Figure 4-2.


Configure ETM

Architecture: 3.3 **Protocol:** 0

Trace Data Width: 4 bit, 8 bit, 16 bit, 24 bit, [✓ 32 bit]

Trace Port Mode:
Port Speed:
ETM Clock Speed: Dynamic
[✓] Half-rate clocking enabled
[ ] Disable Traceport

Trace Buffer Packing: [✓ Automatic], Normal Packing, Double Packing, Quad Packing

FIFO Overflow Protection: [✓ No Protection], Stall Protection, Data Suppression
FIFO Highwater: 0

Trace Co-proc Register Transfer: [✓ None], All, Only When Tracing Data

Extended External Input Selection (0..49)
Input 1: 0 Input 2: 0 Input 3: Input 4:

Common
Memory Map Decode: 0x0
Synchronization Frequency: 1024
[ ] Enable Timestamping
[ ] Cycle Accurate Tracing
[ ] Data Only Trace (Do not trace instrucions)
[ ] Suppress Data on FIFO Full

Figure 4-2 The Configure ETM dialog box for ETMv3

Note

The settings available depend on the ETM.

See also

- *Configuring the ETM parameters* on page 4-5.

4.3 Configuring the ETM parameters

The following sections describe how to configure the ETM parameters:

- *Trace data width*
- *Trace port mode (ETMv1)* on page 4-6
- *Trace port mode (ETMv3)* on page 4-7
- *Trace buffer packing* on page 4-7
- *FIFO overflow protection* on page 4-8
- *Trace coproc register transfer* on page 4-9
- *Extended external input selection (ETMv3.1 and later)* on page 4-9
- *Memory map decode* on page 4-10
- *Synchronization frequency (ETMv3)* on page 4-10
- *Enable Timestamping* on page 4-10
- *Cycle accurate tracing* on page 4-11
- *Data only trace (Do not trace instructions) (ETMv3.1 and later)* on page 4-11
- *Suppress data on FIFO full* on page 4-12.

Note

Some of these settings might be grayed out, depending on your system design or current configuration, for example, packing modes determine what port widths are available.

See also:

- *Specifying the type of information to collect for data transfer instructions* on page 3-11
- *Common parameters for setting conditional tracepoints* on page 7-3
- *Changing the columns displayed in the Trace view* on page 9-4
- *Changing the default format of time information* on page 9-13
- *Profiling trace information* on page 2-22
- *Viewing captured profiling information* on page 9-30
- the following in the *Command Line Reference Guide*:
  - *ETM_CONFIG* on page 2-143
- *ETM control register* section in the *ARM Embedded Trace Macrocell Specification*.

4.3.1 Trace data width

This control enables you to set the width of the ETM trace data port, which determines the number of trace port pins that are used to broadcast trace information. This can be useful, for example, when trace port pins are multiplexed onto *General Purpose Input/Output* (GPIO) pins, and the hardware is configured to use these pins in their GPIO role. Only those widths supported by your TPA and target are enabled for selection.

The width of the ETM data port is configured by the processor manufacturer to one of the following sizes:

4-bit	A 4-bit ETM data port using 9 output signals on the target being traced.
8-bit	An 8-bit ETM data port using 13 output signals on the target being traced.
16-bit	A 16-bit ETM data port using 21 output signals on the target being traced.
24-bit	A 24-bit ETMv3 data port. This size is not currently available for any trace hardware.

32-bit	A 32-bit ETMv3 data port. This size is currently available only for ETB11™ trace hardware, when using on-chip tracing with RealView ICE. Also, if you are using ETB11 trace hardware, then this is the only size available.

——— **Note** ———

A narrow port width reduces the bandwidth available to the target for sending trace data off-chip. This usually produces more FIFO overflows, which results in less data being generated. Therefore, the profiling quality is reduced.

———

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.2 Trace port mode (ETMv1)

This control specifies the way the trace port operates. For ETMv1 architectures, select one of the following values:

Normal	The normal mode. Trace data from the ETM is written to the output pins at the processor frequency.
Multiplexed	Use this to reduce the number of output pins used by the trace port. Two output signals are output on the same pin by clocking the signals at double the normal rate.
De-Multiplexed	Use this to reduce the signal switching frequency of the trace port signals. One output signal is output on two pins, so the pins are clocked at half the normal rate.

——— **Note** ———

- Only **Normal** mode operation is possible when you are using ETM hardware implementing ETM Architecture version 1.1 or earlier.
- If you use multiplexed or demultiplexed clocks, you might have to alter the configuration of your TPA.

———

You can also use this section of the dialog box to enable or disable half-rate clocking, or suppress output from the trace port:

Half-rate clocking enabled

Select **Half-rate clocking enabled** if you want to set the ETM half-rate clocking signal. The effect of this signal is dependent on the implementation of your system.

——— **Note** ———

- This setting is not available when you are using ETM hardware implementing ETMv1.0. Hardware implementing ETMv1.1 might support the setting, but RealView Debugger cannot detect whether it does.
- If you enable half-rate clocking, you might have to alter the configuration of your TPA.
- This capability is not supported by all TPAs. See the documentation that accompanies your hardware to see if it is available.

———

Disable traceport

Select **Disable traceport** if you want to suppress the output from the trace port. This is useful if your hardware has two or more ETMs sharing a single trace port.

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.3 Trace port mode (ETMv3)

This control specifies the way the trace port operates. For ETMv3 architectures, select the **Port speed:ETM clock speed** ratio from the drop-down list. This enables the trace port to run at a different speed to that of the processor. Therefore, you can capture trace from cores that run faster than the trace port capture speed. The available options are determined by the ASIC vendor:

- **Dynamic** (for use with on-chip trace buffers). This is the only option available if you are using ETB11™.
- **1:1** (the port speed is the same as the ETM clock speed).
- **1:2** (the port speed is half the ETM clock speed).
- **1:3** (the port speed is one third of the ETM clock speed).
- **1:4** (the port speed is one quarter of the ETM clock speed).
- **2:1** (the port speed is double the ETM clock speed).
- **Implementation defined** (defined by the ASIC manufacturer).

——— **Note** ———

The **Port speed:ETM clock speed** ratio defines the data rate of the trace port, not the trace clock speed. The trace clock speed is always half the trace port rate, because half-rate clocking is mandated by the ETMv3 architecture.

———————————

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.4 Trace buffer packing

This setting is specific to the *Trace Port Analyzer* (TPA). It enables you to select the mode in which trace data is packed into the trace buffer. Double and Quad packing modes enable more efficient use of the TPA memory. This increases the trace depth, but results in coarser timestamping. Half-rate clocking and packing modes are available at **TRACECLK** frequencies above 100MHz.

The trace buffer packing modes available are:

Automatic This mode enables the TPA to select the best packing mode for the current port width. Select this option if timestamping is not enabled.

Normal Packing

In this mode, the TPA records one timestamp for each trace sample. This gives the best possible resolution of timestamps, at the expense of trace depth.

Double Packing

In this mode, the TPA records one timestamp for every two consecutive trace samples. This reduces the resolution of the timestamps, but increases the trace depth.

Quad Packing

In this mode, the TPA records one timestamp for every four consecutive trace samples. This gives the greatest trace depth, but reduces the resolution of timestamps even more.

——— **Note** ———

Some combinations of packing mode and port width might not be valid for your system. For example, **Quad Packing** might only be possible if you are using a narrow port width.

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.5 FIFO overflow protection

The ETM contains a FIFO buffer that holds the traced data for transmission through the trace port. When this FIFO buffer becomes full, trace information can be lost if new information arrives before it can drain, unless you have programmed the ETM to stall (temporarily stop) the processor. The options in this section of the dialog box enable you to protect against FIFO overflows.

Select one of the following options:

No protection

Select this option if you do not want to use FIFO overflow protection.

Stall processor

Select this option if you want the system to stall the processor until the FIFO buffer is empty. You must also set the **FIFO highwater** by typing a value into the text box. When the number of bytes left in the FIFO buffer is reduced to the number of bytes you set in **FIFO highwater**, the processor stalls as soon as possible. It restarts when the FIFO buffer is empty.

——— **Note** ———

This capability is not supported by all systems. See the documentation that accompanies your system to find out whether `FIFOFULL` is implemented.

Data suppression

Select this option to instruct RealView Debugger to stop tracing data when the FIFO is close to the overflow limit. This helps to prevent a FIFO overflow, because the bandwidth required for instruction tracing is much lower than that required for data tracing.

The following warning message is displayed when data is being suppressed, and again when data is unsuppressed:

```
Warning: Data suppression protected ETM FIFO from overflow
```

——— **Note** ———

This option is only available if you are using ETMv3, and is grayed out for all other ETM architectures.

———————————

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.6 Trace coproc register transfer

This area of the dialog box enables you to specify when *Coprocessor Register Transfers* (CPRTs) are traced. CPRTs are the instructions *Move Coprocessor from ARM Register* (MCR) and *Move ARM Register from Coprocessor* (MRC).

Select one of the following options:

None Do not trace CPRTs.

All Trace all CPRTs.

Only when tracing data

Only trace CPRTs when data is being traced. To specify whether or not to trace data, select **Data tracing mode** from the **Edit** menu.

——— **Note** ———

This setting is only available if you are using ETMv3. It is grayed out for all other ETM architectures.

———————————

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.7 Extended external input selection (ETMv3.1 and later)

Enables you to enter a value for each part of the Extended External Input Selector register, inputs 1 to 4. Each part can have a value in the range 0 to 255. However, the number of inputs, the range of values supported, and the default value of each input depends on the ETM you are using. For example, the ARM implementation of the ARM1136JF-S™ processor in the CM1136 core module has two extended external inputs with values in the range zero to 20 and default values of zero. Third party ARM1136JF-S processor implementations might have a different number of external inputs.

If you enter a value outside the range supported by the ETM, an Error dialog box is displayed. The dialog box informs you of the range of values supported and the input that has the incorrect value.

Use these inputs in conjunction with setting external condition tracepoints.

——— **Note** ———

These inputs are supported only by ETMv3.1, and later.

———————————

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.8 Memory map decode

This is an implementation-dependent value that varies depending on the memory map decode logic present in your system. This value is written to a control register, intended to configure the memory map decode hardware. For more details, see your system ASIC documentation.

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.9 Synchronization frequency (ETMv3)

For ETMv3.0, and later, a synchronization frequency register is used to define the time between synchronization points in the trace data. That is, the points where the trace tools start decompressing the trace output. This allows smaller buffers, such as *Embedded Trace Buffer*™ (ETB™), to be more resistant to data corruption by decreasing the time between synchronization points. However, increasing the synchronization frequency gives more synchronization points, and more information must be stored in the trace buffer.

The synchronization frequency can be a value in the range 100 to 4095, with the default being 1024:

- for ETMv3.0, the value is in cycles
- for ETMv3.1 and later, the value is in bytes.

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.10 Enable Timestamping

This setting enables the timestamp recording logic in your TPA, such as RealView Trace. Timestamps are displayed in the `Time/unit` and `+Time` columns of the Analysis window. To change the unit in which timestamps are displayed, select **Scale Time Units...** from the Analysis window **View** menu.

If you want to view profiling information, you must enable either timestamping or cycle-accurate tracing. If you have an ETMv3 processor, then instruction count-based profiling information is shown, unless you enable one of these options. However, if discontinuities in the trace are not important for the profile you want to generate, then a more precise profile can be generated if you specify cycle-accurate tracing without timestamping. The ETMv3 trace protocol derived core cycles are used as the timing data for profiling.

Considerations for timestamping

Be aware of the following for timestamping:

- For ETMv3, timestamping and cycle-accurate tracing are mutually exclusive.
- Timestamping is not supported when using an on-chip ETB.

- Transmitting the timestamps uses additional bandwidth on the TPA to host connection. Storing the timestamps in the TPA reduces the maximum length of trace that you can capture. Therefore, only enable this feature when you have to.

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.11 Cycle accurate tracing

This setting determines whether the ETM operates in cycle-accurate mode:

- When **Cycle accurate tracing** is selected, the ETM records the number of cycles executed while tracing is enabled. This includes cycles during which no trace information is normally returned, such as memory wait states.

  The `Elem` column of the Analysis window shows the cycle number of the cycle in which each instruction was executed. The count does not include cycles executed during a trace discontinuity. You must use the `Time/`*`unit`* column, which displays timestamp values, to measure across discontinuities in the trace output.
- If **Cycle accurate tracing** is not selected, the ETM does not record cycle counts.

Considerations for cycle-accurate tracing

Be aware of the following for cycle-accurate tracing:

- For ETMv3, timestamping and cycle-accurate tracing are mutually exclusive.
- For ETMv1, cycle-accurate tracing does not record timing information between trace discontinuities.
- Fewer instructions are captured when doing cycle-accurate tracing relative to non cycle-accurate tracing. This is because the ETM must output more information with less compression.

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.3.12 Data only trace (Do not trace instructions) (ETMv3.1 and later)

This setting forces the tracing of data transfers only. Otherwise, both data and instructions are traced. In addition, cycle-accurate tracing is disabled, because no cycle values are recorded.

You must also enable data tracing if you select this option. Otherwise, you do not receive any trace, or you might only see discontinuities or status messages in the trace.

——— **Note** ———

This setting is only available if you are using ETMv3.1 or later, and is grayed out for all other ETM architectures.

————————

4.3.13 Suppress data on FIFO full

This setting suppresses the output from the trace port after a FIFO overflow occurs. Some versions of the ETM produce incorrect trace data following FIFO overflow. This only occurs on cached processors with slow memory systems, and happens when a cache miss occurs at the same time that the FIFO on the ETM overflows. If you select this setting, the decompressor suppresses the data that the ETM might have traced incorrectly. However, some correctly traced data might also be suppressed.

——— **Note** ———

This setting is disabled if your ETM does not generate incorrect trace data under these circumstances.

————————

See also

- *Configuring the ETM parameters* on page 4-5
- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13.

4.4 Equivalent ETM_CONFIG CLI command qualifiers

Table 4-1 shows the Configure ETM dialog box settings and the equivalent qualifiers to use with the `ETM_CONFIG` command to perform the same function in the CLI.

Table 4-1 ETM Configuration settings to command qualifier mapping

Configure ETM dialog box setting	Command qualifier
Trace data widths:	
• 4 bit	`port_width:0`
• 8 bit	`port_width:1`
• 16 bit	`port_width:2`
• 24 bit (not supported with RealView Trace)	`port_width:3`
• 32 bit (supported by ETB11 only, but not with RealView Trace)	`port_width:4`
Trace port modes (non-v3 architecture):	
• Normal	`nomultiplex`
• Multiplexed	`multiplex`
• Demultiplexed	`demultiplex`
Trace port modes (ETMv3 only), of the form	
`Port_speed``:`*`ETM_clock_speed`*:	`portratio:0`
• 1:1	`portratio:1`
• 1:2	`portratio:2`
• 1:3	`portratio:3`
• 1:4	`portratio:4`
• 2:1	`portratio:5`
• Use dynamic ratio modes for on-chip trace.	`portratio:6`
• Use the implementation-defined mode, if implemented by the ASIC designer.	
Half-rate clocking enabled	`half_rate`
Disable traceport	`disableport`
Trace buffer packing:	
• Automatic	`packauto`
• Normal packing	`packnormal`
• Double packing	`packdouble`
• Quad packing	`packquad`
FIFO overflow protection:	
• Stall processor	`stall_full`
• Data suppression	`suppressdata`
• FIFO highwater	`FIFO_hw:`*`n`*
Trace coproc register transfer:	
• Stall processor	`coprocessor`
• Only when tracing data	`filtercoprocessor`

Table 4-1 ETM Configuration settings to command qualifier mapping (continued)

Configure ETM dialog box setting	Command qualifier
Extended external input selection (ETMv3.1 and later): • Input 1 • Input 2 • Input 3 • Input 4 The number of inputs available depends on your ETM configuration.	 `extin1:`*n* `extin2:`*n* `extin3:`*n* `extin4:`*n*
Memory map decode	`mmap_decode:`*n*
Synchronization frequency	`syncfrequency:`*n*
Enable Timestamping	`time_stamps`
Cycle-accurate tracing	`cycle_accurate`
Data only trace (Do not trace instructions)	`dataonly`
Suppress data on FIFO full	`datasuppression`

See also:

- the following in the *Command Line Reference Guide*:
  - *ETM_CONFIG* on page 2-143.

Chapter 5
Tracepoints in RealView Debugger

This chapter gives an overview of the tracepoints available in RealView® Debugger. It includes:

- *About tracepoints* on page 5-2
- *Tracepoint types* on page 5-3
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7.

5.1 About tracepoints

Tracepoints enable you to set conditions for generating trace information. Tracepoints can be:

Unconditional	These include individual trigger points, trace start and end points, and trace ranges for instruction and data accesses.
Conditional	These include AND or OR conditions, tests on the number of executions, and complex comparisons.

You can set tracepoints in the Code window, either through dialog boxes, or by issuing CLI commands directly or in command scripts.

See also:

- Chapter 6 *Setting Unconditional Tracepoints*
- Chapter 7 *Setting Conditional Tracepoints*.

5.2 Tracepoint types

The following sections describe the types of tracepoint available in RealView Debugger:

- *Trigger*
- *Trace Start Point and Trace End Point* on page 5-4
- *Trace Range* on page 5-5
- *ExternalOut Points* on page 5-6.

5.2.1 Trigger

A trigger enables you to focus trace collection around a specific region of interest. Although you can set multiple trigger conditions, only the first trigger condition that is hit appears in the trace buffer.

A trigger has no impact on the output from the targets with *Embedded Trace Macrocell*™ (ETM™), other than the signal that it has occurred. The ETM outputs a trigger event, either over the trace port to your *Trace Port Analyzer* (TPA) (see Figure 2-1 on page 2-3) or to an on-chip target with *Embedded Trace Buffer*™ (ETB™) (see Figure 2-2 on page 2-4).

For *RealView ARMulator® ISS* (RVISS), the appropriate instruction packet is marked as the trigger.

Specifying the area of interest near a trigger

A trigger is interpreted by the TPA to determine what additional information to capture and when to dump its contents back to the debugger:

Trace before This indicates that trace information only before the trigger is of interest, which is the default. This is used to find out what has caused a certain event, for example, to see what sequence of code was executed before entering an error handler routine. A small amount of trace data is often included after the trigger condition.

Trace around

This indicates that trace information before and after the trigger is of interest. You can alter the amount of trace information before and after the trigger by using a trace start point to enable tracing before the trigger, and a trace end point to disable tracing after the trigger.

Trace after This indicates that trace information only from the trigger point onwards is of interest. It is used to find out what happens after a particular event, for example what happens after entering an interrupt service routine. A small amount of trace data is often included before the trigger condition.

Identifying a trigger

A trigger is identified by an arrow ➡ in the left margin, next to the line of code you select.

You can set triggers over a range of addresses, where any instruction in that range activates the trigger. However, only the first instruction that is hit within the range activates the trigger. In this case, the range is indicated in the same way as any other trace range.

Considerations when stopping execution at a trigger

You can also choose to stop the processor when the trigger event occurs. However, the trace information that is collected when tracing around or after the trigger depends on your TPA. For RealView Trace, trace information is captured before the trigger, but none after the trigger.

——— **Note** ———

On simulators, such as RVISS, you cannot stop the processor when a trigger occurs. However, you can still choose to capture trace before, around, or after the trigger.

———————————

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Setting a trace range for selected source lines or disassembly* on page 6-13
- *Status messages in the captured trace* on page 9-3.

5.2.2 Trace Start Point and Trace End Point

Set a Trace Start Point and a Trace End Point to indicate where tracing is to be enabled and disabled. The advantage of enabling and disabling the capture of trace information at specific parts of your image is that it effectively increases the amount of useful information that can be captured for a given size of trace buffer. Therefore, you can enable selective tracing over a longer time period.

Only instructions are traced between trace start and end points. If you want to trace data, then you must also set a trace range that includes data, and that also overlaps the region bounded by the trace start and end points.

You can use trace start and end points in conjunction with trace ranges.

Trace start and end point support

The support for trace start and end points depends on your version of ETM:

- In ETMv1.1 or earlier, RealView Debugger uses the ETM state machine, if available, to support a limited number of start and end points, either:
  - up to four start points and up to two stop points
  - up to two start points and up to four stop points.

  The limitation option used by RealView Debugger depends on the type of tracepoints you want to set. For example, if you are using ETMv1.1 or earlier, and set three trace start points, then the first case is assumed. That is, you can set up to four start points, but are limited to two stop points.

- In ETMv1.2 or later, you can turn instruction tracing on or off whenever certain instructions are executed or when specified data addresses are accessed. This means that you can trace functions, subroutines, or individual variables held in memory.

  You can enable or disable tracing when any single address comparator matches. The effect is only precise if the address comparator is instruction execution or data address based. You can use instruction fetch comparisons, but the effect is not precise.

How trace is captured using trace start and end points

Trace information is returned from the specified start point until the specified end point, including any areas that are branched to. However:

- if you do not set a trace end point, trace information is returned from the trace start point until execution stops
- if you do not set a trace start point, then behavior depends on your system:
  - if you are using an ETM-based system, no trace information is returned, and a warning is displayed in the **Cmd** tab of the Output view
  - if you are using a simulator, tracing begins immediately and trace information is returned up until the trace end point is traced.

Identifying trace start and end points

The trace start and end points are identified respectively by the arrows ⇩ and ⇧ in the left margin. You must set each point individually at the required lines of code.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Setting a trace range for selected source lines or disassembly* on page 6-13
- *Status messages in the captured trace* on page 9-3.

5.2.3 Trace Range

A Trace Range indicates an area of your image where a specific type of trace information:

- is captured (Include range)
- is not captured (Exclude range).

You can set the start and end positions of the range individually at the required lines of code, or select a range of lines and set the Trace Range with a single operation.

If you do not set an end position for the range, the default address `0xFFFFFFFF` is used (effectively the end of memory).

You can use trace ranges in conjunction with trace start and end points.

——— **Note** ———

Trace ranges are available only for ETM-based hardware.

———————————

Identifying a trace range

The start and end positions of a Trace Range are identified respectively by the arrows ⇩ and ⇧ in the left margin of the code view.

Include trace range

An Include trace range indicates an area of your image where you want to capture a specific type of trace information. Information is captured only for the specified area, and not for any areas that are branched to. These branches are represented as `Trace Pause` status lines in the Analysis window. You can set the following types of Include trace range:

- instructions only

- instructions and data.

Exclude trace range

An Exclude trace range indicates an area of your image where you do not want to capture trace information. You can set the following types of Exclude trace range:

- instructions and data (cannot be used in conjunction with Include trace ranges)
- data only (can be used in conjunction with Include trace ranges).

Note

The ETM hardware enables you to set both Include and Exclude trace ranges only in specific circumstances. If you attempt to use an Exclude trace range incorrectly, RealView Debugger displays an error in the Output view.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Setting a trace range for selected source lines or disassembly* on page 6-13
- *Status messages in the captured trace* on page 9-3.

5.2.4 ExternalOut Points

Each ExternalOut point controls a single-bit output signal from the ETM. Up to four signals are available, `ExternalOut1`, `ExternalOut2`, `ExternalOut3`, and `ExternalOut4`. These can be used to enable or disable trace capture, or to enable or disable a device or peripheral. The ASIC manufacturer determines the availability and use of these output signals. See your ASIC documentation for details.

You can set a single ExternalOut point at a specific address, or a number of ExternalOut point over a range of addresses.

Behavior of ExternalOut points

The ExternalOut points do not cause trace to be captured, but enable external signals to be generated by the compute logic of the ETM. The result of external outputs depends on the ASIC designer.

Identifying an ExternalOut point

An ExternalOut point that is set on a single line of code is identified by an arrow ➡ in the left margin, next to the line of code you select.

ExternalOut points that are set over a range of addresses are identified in the same way as any other trace range.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Setting a trace range for selected source lines or disassembly* on page 6-13
- *Status messages in the captured trace* on page 9-3.

5.3 Setting trace ranges in conjunction with trace start and end points

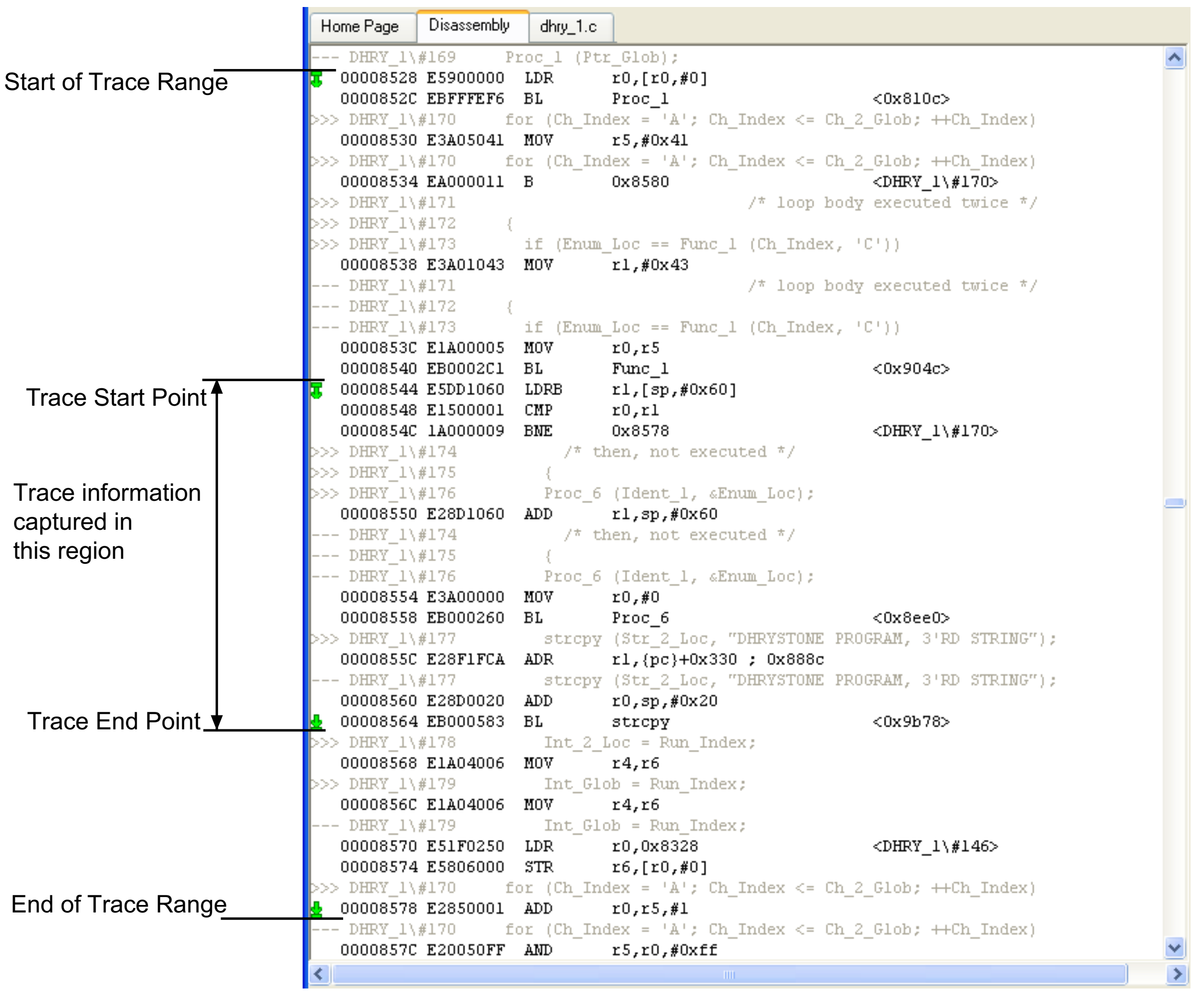
On hardware targets, you can set a trace range in conjunction with trace start and end points. If you do, then the position of the trace start and end points determines the region where trace is captured:

- If the trace range is within the region defined by the trace start and end points then trace information is captured for the whole range.
- If the trace start and end points are within the region defined by the trace range then trace information is captured only between the trace start and end points.

  Figure 5-1 on page 5-8 shows an example. In this example, trace information is captured only between addresses `0x00008544` and `0x00008564`.
- If the trace start point is after the start of the range, no trace information is captured between the start of the range and the trace start point.
- If the trace end point is before the end of the range, no trace information is captured between the trace end point and the end of the range.

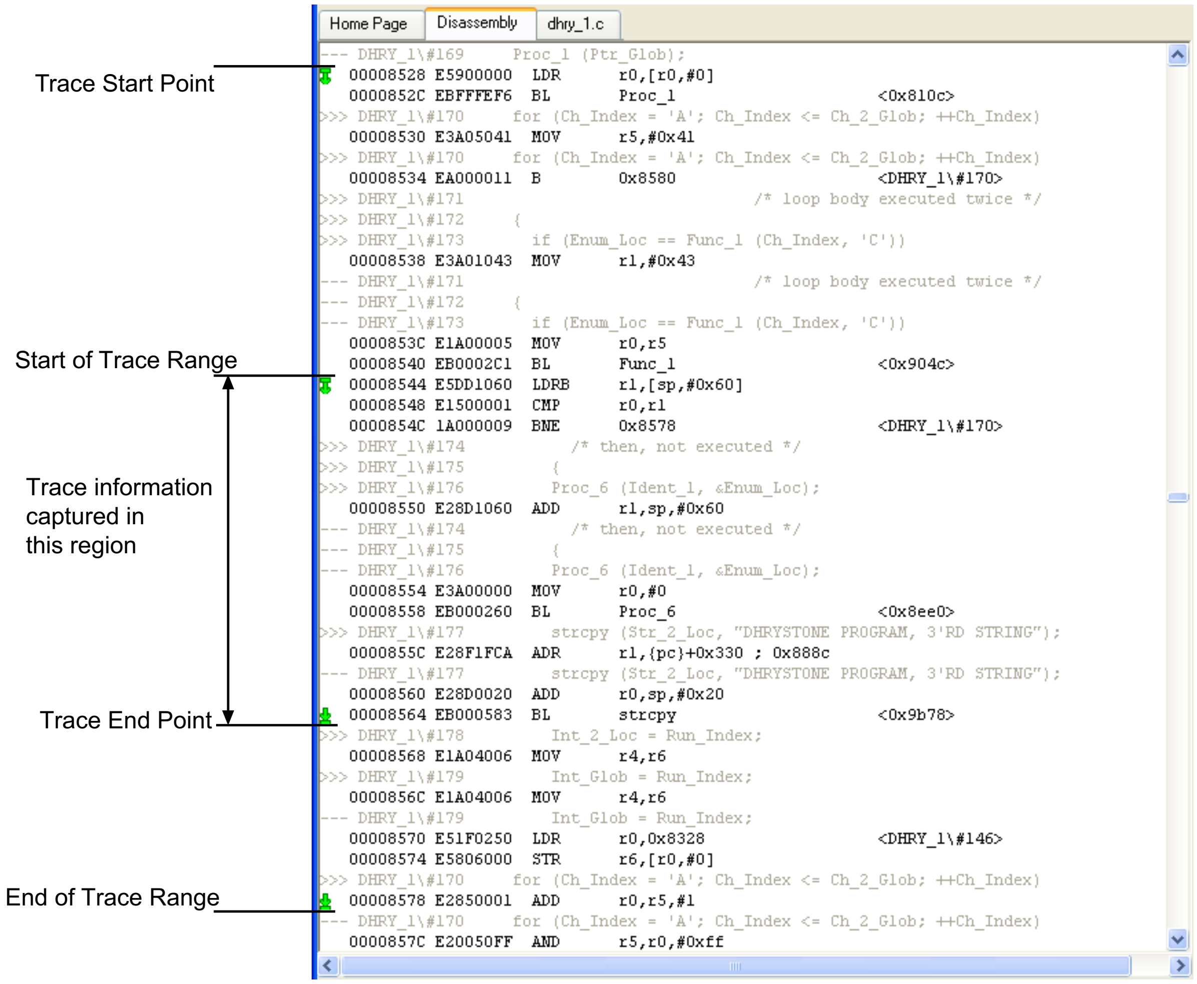
  Figure 5-2 on page 5-9 shows an example. In this example, trace information is captured only between addresses `0x00008544` and `0x00008564`.

To summarize, trace start and end points determine where tracing is enabled, but the trace ranges determine what information is captured.



```
--- DHRY_1\#169      Proc_1 (Ptr_Glob);
   00008528 E5900000  LDR      r0,[r0,#0]
   0000852C EBFFFEF6  BL       Proc_1                      <0x810c>
>>> DHRY_1\#170      for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
   00008530 E3A05041  MOV      r5,#0x41
>>> DHRY_1\#170      for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
   00008534 EA000011  B        0x8580                      <DHRY_1\#170>
>>> DHRY_1\#171                                 /* loop body executed twice */
>>> DHRY_1\#172      {
>>> DHRY_1\#173        if (Enum_Loc == Func_1 (Ch_Index, 'C'))
   00008538 E3A01043  MOV      r1,#0x43
--- DHRY_1\#171                                 /* loop body executed twice */
--- DHRY_1\#172      {
--- DHRY_1\#173        if (Enum_Loc == Func_1 (Ch_Index, 'C'))
   0000853C E1A00005  MOV      r0,r5
   00008540 EB0002C1  BL       Func_1                      <0x904c>
   00008544 E5DD1060  LDRB     r1,[sp,#0x60]
   00008548 E1500001  CMP      r0,r1
   0000854C 1A000009  BNE      0x8578                      <DHRY_1\#170>
>>> DHRY_1\#174            /* then, not executed */
>>> DHRY_1\#175          {
>>> DHRY_1\#176          Proc_6 (Ident_1, &Enum_Loc);
   00008550 E28D1060  ADD      r1,sp,#0x60
--- DHRY_1\#174            /* then, not executed */
--- DHRY_1\#175          {
--- DHRY_1\#176          Proc_6 (Ident_1, &Enum_Loc);
   00008554 E3A00000  MOV      r0,#0
   00008558 EB000260  BL       Proc_6                      <0x8ee0>
>>> DHRY_1\#177          strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 3'RD STRING");
   0000855C E28F1FCA  ADR      r1,{pc}+0x330 ; 0x888c
--- DHRY_1\#177          strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 3'RD STRING");
   00008560 E28D0020  ADD      r0,sp,#0x20
   00008564 EB000583  BL       strcpy                      <0x9b78>
>>> DHRY_1\#178          Int_2_Loc = Run_Index;
   00008568 E1A04006  MOV      r4,r6
>>> DHRY_1\#179          Int_Glob = Run_Index;
   0000856C E1A04006  MOV      r4,r6
--- DHRY_1\#179          Int_Glob = Run_Index;
   00008570 E51F0250  LDR      r0,0x8328                   <DHRY_1\#146>
   00008574 E5806000  STR      r6,[r0,#0]
>>> DHRY_1\#170      for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
   00008578 E2850001  ADD      r0,r5,#1
--- DHRY_1\#170      for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
   0000857C E20050FF  AND      r5,r0,#0xff
```

Figure 5-1 Trace start and end points within a trace range



```
--- DHRY_1\#169      Proc_1 (Ptr_Glob);
    00008528 E5900000  LDR      r0,[r0,#0]
    0000852C EBFFFEF6  BL       Proc_1                     <0x810c>
>>> DHRY_1\#170      for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
    00008530 E3A05041  MOV      r5,#0x41
>>> DHRY_1\#170      for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
    00008534 EA000011  B        0x8580                     <DHRY_1\#170>
>>> DHRY_1\#171                              /* loop body executed twice */
>>> DHRY_1\#172      {
>>> DHRY_1\#173        if (Enum_Loc == Func_1 (Ch_Index, 'C'))
    00008538 E3A01043  MOV      r1,#0x43
--- DHRY_1\#171                              /* loop body executed twice */
--- DHRY_1\#172      {
--- DHRY_1\#173        if (Enum_Loc == Func_1 (Ch_Index, 'C'))
    0000853C E1A00005  MOV      r0,r5
    00008540 EB0002C1  BL       Func_1                     <0x904c>
    00008544 E5DD1060  LDRB     r1,[sp,#0x60]
    00008548 E1500001  CMP      r0,r1
    0000854C 1A000009  BNE      0x8578                     <DHRY_1\#170>
>>> DHRY_1\#174          /* then, not executed */
>>> DHRY_1\#175          {
>>> DHRY_1\#176          Proc_6 (Ident_1, &Enum_Loc);
    00008550 E28D1060  ADD      r1,sp,#0x60
--- DHRY_1\#174          /* then, not executed */
--- DHRY_1\#175          {
--- DHRY_1\#176          Proc_6 (Ident_1, &Enum_Loc);
    00008554 E3A00000  MOV      r0,#0
    00008558 EB000260  BL       Proc_6                     <0x8ee0>
>>> DHRY_1\#177          strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 3'RD STRING");
    0000855C E28F1FCA  ADR      r1,{pc}+0x330 ; 0x888c
--- DHRY_1\#177          strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 3'RD STRING");
    00008560 E28D0020  ADD      r0,sp,#0x20
    00008564 EB000583  BL       strcpy                     <0x9b78>
>>> DHRY_1\#178          Int_2_Loc = Run_Index;
    00008568 E1A04006  MOV      r4,r6
>>> DHRY_1\#179          Int_Glob = Run_Index;
    0000856C E1A04006  MOV      r4,r6
--- DHRY_1\#179          Int_Glob = Run_Index;
    00008570 E51F0250  LDR      r0,0x8328                  <DHRY_1\#146>
    00008574 E5806000  STR      r6,[r0,#0]
>>> DHRY_1\#170      for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
    00008578 E2850001  ADD      r0,r5,#1
--- DHRY_1\#170      for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
    0000857C E20050FF  AND      r5,r0,#0xff
```

Figure 5-2 Trace end point within a trace range

Chapter 6
Setting Unconditional Tracepoints

This chapter describes how to set unconditional tracepoints. It includes:

6.1 About setting unconditional tracepoints

You can set unconditional tracepoints using one of the following methods:

- Use the New Tracepoint dialog box to quickly set the following unconditional tracepoints:
  - triggers
  - trace start and end points
  - trace ranges
  - external output events, if the target has an *Embedded Trace Macrocell*™ (ETM™) that supports them.

  These can be set individually or in conjunction with other tracepoints, to ensure capture of trace information for the precise area of interest.
- Set a trace range on selected lines of source or disassembly in the Code window.
- Manually with the Create Tracepoint dialog box.
- Use the any of the trace commands, `TRACE`, `TRACEDATAACCESS`, `TRACEDATAREAD`, `TRACEDATAWRITE`, `TRACEEXTCOND`, `TRACEINSTREXEC`, or `TRACEINSTRFETCH`.

Note

The amount of trace information returned depends on the buffer size that is currently set.

When you set a tracepoint, such as a trigger, a corresponding **Clear** option becomes available in the List Selection dialog box. You can select the **Clear** option to remove the tracepoint you have set, and the arrow in the left margin of your code is removed. The option **Clear Range** removes both the start and end points of the range you have set.

Note

You can set multiple tracepoints on an individual source line. Therefore, if you clear one tracepoint and another exists at the same location, the arrow icon that indicates the tracepoint is still present.

See also:

- *Configuring common trace options* on page 3-8
- *Setting a trigger point* on page 6-4
- *Setting a trace start and end point* on page 6-5
- *Setting a trace range* on page 6-7
- *Setting an ExternalOut Point* on page 6-10
- *Setting a trace range for selected source lines or disassembly* on page 6-13
- *Setting unconditional tracepoints with the Create Tracepoint dialog box* on page 6-14
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the `TRACE`, `TRACEDATAACCESS`, `TRACEDATAREAD`, `TRACEDATAWRITE`, `TRACEEXTCOND`, `TRACEINSTREXEC`, and `TRACEINSTRFETCH` commands.

6.2 Capturing instructions and data

If you want to capture instructions and data, use one of the following methods:

- For processors with an ETM:
  - set an unconditional trace range with the New Tracepoint dialog box, using the **Start of Trace Range (Instruction and Data)** and **End of Trace Range (Instruction and Data)** options
  - use the Create Tracepoint dialog box to set tracepoints of type **Trace Instr and Data**.
- For *RealView® ARMulator® ISS* (RVISS) connections only:
  - set unconditional trace start and end points with the New Tracepoint dialog box using the **Trace Start Point (Instruction and Data)** and **Trace End Point** options
  - set trace start and end points with the Create Tracepoint dialog box to set tracepoints of types **Trace Start Point (Instr and Data)** and **Trace End Point**.

See also:

- *Setting a trace range* on page 6-7
- *Setting a trace start and end point* on page 6-5
- *Setting unconditional tracepoints with the Create Tracepoint dialog box* on page 6-14.

6.3 Setting a trigger point

A trigger point enables you to display the trace that has been captured up to that point.

To set a trigger point:

1. In the source or disassembly view, right-click on the gray margin to the left of the location to display context menu.
2. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box. Figure 6-1 shows an example:

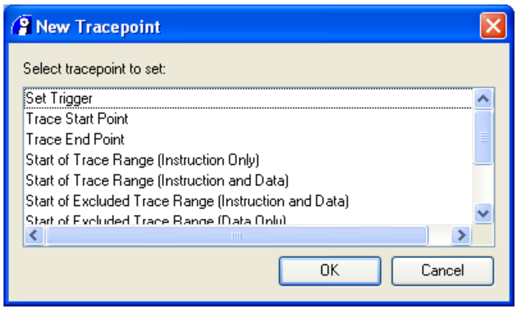

New Tracepoint
Select tracepoint to set:
Set Trigger
Trace Start Point
Trace End Point
Start of Trace Range (Instruction Only)
Start of Trace Range (Instruction and Data)
Start of Excluded Trace Range (Instruction and Data)
Start of Excluded Trace Range (Data Only)
OK
Cancel

Figure 6-1 New Tracepoint dialog box

——— **Note** ———

The options that appear in this dialog box depend on the available resources and on the target that you are using. In some cases, you must clear an existing tracepoint or range to free up the resources you might require for a new tracepoint or range.

———

3. Select **Set Trigger** in the New Tracepoint dialog box.
4. Click **OK**. An arrow ➡ is placed in the left margin next to the line of code you have selected.

Use this in conjunction with the **Trigger Mode** options on the Analysis window to determine whether trace information is collected before, around or after the trigger.

See also:

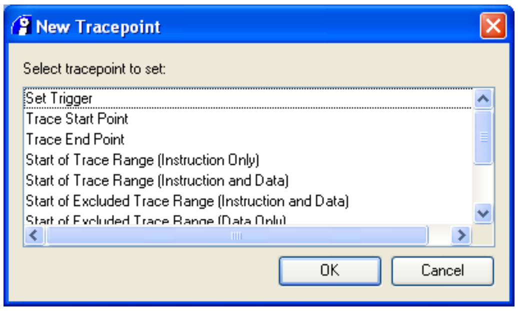
- *Configuring how trace information is collected at a trigger* on page 3-10
- *Trigger* on page 5-3.

6.4 Setting a trace start and end point

Trace start and end points enable you to start and stop trace capture at specific locations. Any functions that are called between the start and end points are also traced.

To set a trace start or end point:

1. In a source or disassembly view, right-click on the gray margin to the left of the required location to display the context menu.
2. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box. Figure 6-2 shows an example:



Set Trigger
Trace Start Point
Trace End Point
Start of Trace Range (Instruction Only)
Start of Trace Range (Instruction and Data)
Start of Excluded Trace Range (Instruction and Data)
Start of Excluded Trace Range (Data Only)

Figure 6-2 New Tracepoint dialog box

Note

The options that appear in this dialog box depend on the available resources and on the target that you are using. In some cases, you must clear an existing tracepoint or range to free up the resources you might require for a new tracepoint or range.

3. Select the required tracepoint type from the New Tracepoint dialog box:

Trace Start Point

Sets a trace start point at the selected location in the source or disassembly view.

Note

If you do not set a **Trace End Point** in combination with a **Trace Start Point**, all-inclusive trace information is returned from the start point onward.

This option is not available for RVISS targets.

Trace Start Point (Instruction Only)

Sets a trace start point at the selected location in the source or disassembly view. Tracing begins at the start point, and instructions only are traced.

Note

If you do not set a **Trace End Point** in combination with a **Trace Start Point**, all-inclusive trace information is returned from the start point onward.

This option is only available for RVISS targets.

Trace Start Point (Instruction and Data)

Sets a trace start point at the selected location in the source or disassembly view. Tracing begins at the start point, and both instructions and data are traced.

Note

If you do not set a **Trace End Point** in combination with a **Trace Start Point**, all-inclusive trace information is returned from the start point onward.

This option is only available for RVISS targets.

Trace End Point

Sets a trace end point at the selected location in the source or disassembly view. That is, the point where trace collection stops.

Note

Setting a trace end point but no trace start point results in different behavior depending on your system:

- If you are using an ETM-based system, no trace information is returned, and a warning is displayed in the **Cmd** tab of the Output view.
- If you are using a simulator, tracing begins immediately and trace information is returned up until the trace end point is traced.

---

4. Click **OK**. The selected tracepoint is set as follows:
   - for a trace start point, an arrow is placed in the left margin next to the line of code you have selected
   - for a trace end point, an arrow is placed in the left margin next to the line of code you have selected.

Note

For hardware targets, you can set trace ranges in conjunction with trace start and end points.

See also:

- *Trace Start Point and Trace End Point* on page 5-4
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7.

6.5 Setting a trace range

A trace range enables you to capture trace within a specific region of your code. However, any functions that are called within the specified region are not traced.

When you set a trace range using the options described in this section, the end address is automatically set to `0xFFFFFFFF`. To set a specific end address, you must use the corresponding end range point option.

Note

Trace ranges are available only for ETM-based hardware.

You can set trace ranges in conjunction with trace start and end points.

Note

If you want to set an exclude trace range within a standard trace range, you must first set a specific end address for the standard trace range.

To set a trace range:

1. In the source or disassembly view, right-click on the gray margin to the left of the location where you want the trace range to end. A context menu is displayed.
2. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box. Figure 6-3 shows an example:

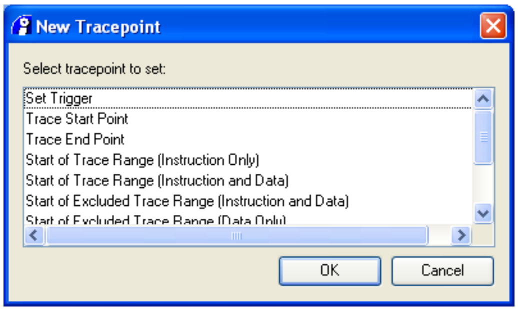

New Tracepoint
Select tracepoint to set:
Set Trigger
Trace Start Point
Trace End Point
Start of Trace Range (Instruction Only)
Start of Trace Range (Instruction and Data)
Start of Excluded Trace Range (Instruction and Data)
Start of Excluded Trace Range (Data Only)
OK
Cancel

Figure 6-3 New Tracepoint dialog box

Note

The options that appear in this dialog box depend on the available resources and on the target that you are using. In some cases, you must clear an existing tracepoint or range to free up the resources you might require for a new tracepoint or range.

3. Select the required start-range point from the New Tracepoint dialog box:

Start of Trace Range (Instruction Only)
Sets the start point for a range of addresses for which trace of program instructions only are captured.

Start of Trace Range (Instruction and Data)
Sets the start point for a range of addresses for which trace of program instructions and data accesses are captured.

Start of Excluded Trace Range (Instruction and Data)

Sets the start point for a range of addresses for which trace of program instructions and data accesses are not captured. This option is the inverse of the option **Start of Trace Range (Instruction and Data)**, where the excluded range you set ensures that program instructions and data accesses are captured for all areas of your application except those within the range you specify.

——— **Note** ———

If the excluded range you specify contains a branch to another area of your application, that branched area is included in the trace capture if it has itself been marked for capture, or if no other points are set.

————————

Start of Excluded Trace Range (Data Only)

Sets the start point for a range of addresses for which trace of data accesses only are not captured. That is, program instructions for the range you specify are captured, and program instructions and data accesses for all areas outside that range are also captured.

——— **Note** ———

If the excluded range you specify contains a branch to another area of your application, the program instructions and data accesses of that branched area are also included in the trace capture if they have themselves been marked for capture, or if no other points are set.

————————

——— **Note** ———

Because the tracepoint type you select from this dialog box is set only at the chosen line of code or address, you must set the start-range and end-range points individually.

————————

4. Click **OK**. An arrow for the start-range point is placed in the left margin next to the line of code you have selected.

5. In the source or disassembly view, right-click on the gray margin to the left of the location where you want the trace range to end. The context menu is displayed.

6. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box. Figure 6-3 on page 6-7 shows an example.

   An end-range point corresponding to the selected start-range point is available:

   - **End of Trace Range (Instruction Only)**
   - **End of Trace Range (Instruction and Data)**
   - **End of Excluded Trace Range (Instruction and Data)**
   - **End of Excluded Trace Range (Data Only)**

   ——— **Note** ———

   No start-range point options are available until after you have set an associated end-range point.

   ————————

7. Select the available end-range point.

8. Click **OK**. An arrow for the end-range point is placed in the left margin next to the line of code you have selected.

——— **Note** ———

For hardware targets, you can set trace ranges in conjunction with trace start and end points.

———————

See also:

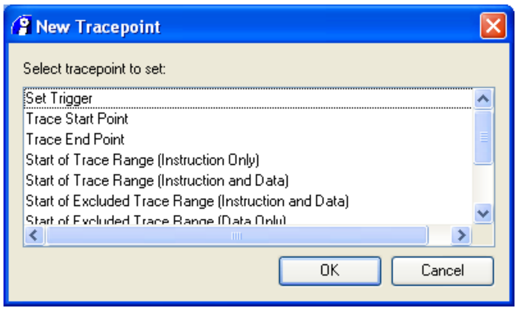
- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Setting a trace range for selected source lines or disassembly* on page 6-13.

6.6 Setting an ExternalOut Point

The ExternalOut points do not cause trace to be captured, but enable external signals to be generated by the compute logic of the ETM. The result of external outputs depends on the ASIC designer.

To set the ExternalOut points:

1. In the source or disassembly view, right-click on the gray margin to the left of the location to display context menu.
2. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box. Figure 6-4 shows an example:



Select tracepoint to set:

Set Trigger
Trace Start Point
Trace End Point
Start of Trace Range (Instruction Only)
Start of Trace Range (Instruction and Data)
Start of Excluded Trace Range (Instruction and Data)
Start of Excluded Trace Range (Data Only)

Figure 6-4 New Tracepoint dialog box

——— **Note** ———

The options that appear in this dialog box depend on the available resources and on the target that you are using. In some cases, you must clear an existing tracepoint or range to free up the resources you might require for a new tracepoint or range.

———————————

3. Select the required option from the New Tracepoint dialog box:
   - **Set ExternalOut1 Point**
   - **Set ExternalOut2 Point**
   - **Set ExternalOut3 Point**
   - **Set ExternalOut4 Point**.
4. Click **OK**. An arrow ➡ is placed in the left margin next to the line of code you have selected.

These options are available only for ETM-based hardware, and the number of ExternalOut points available is determined by the ASIC designer.

See also:

- *ExternalOut Points* on page 5-6.

6.7 Equivalent CLI commands used by the New Tracepoint dialog box

For ETM-based traces, when you set unconditional tracepoints with the New Tracepoint dialog box, RealView® Debugger generates equivalent CLI commands. The basic CLI command that is used by RealView Debugger to set the tracepoint types from this dialog box is:

```
TRACEINSTREXEC,hw_out="Tracepoint Type=type" address | address_range
```

The command is modified depending on the type of tracepoint you set. The command qualifiers that RealView Debugger uses when you set unconditional tracepoints with the New Tracepoint dialog box are:

Trigger `hw_out="Tracepoint Type=Trigger"`

Trace Start Point

```
hw_out="Tracepoint Type=Start Tracing"
```

Trace Start Point (Instruction Only)

```
hw_out:"Tracepoint Type=Trace Start Point (Instr)"
```

Trace Start Point (Instruction and Data)

```
hw_out:"Tracepoint Type=Trace Start Point (Instr+Data)"
```

Trace End Point

This depends on the target:

- for ETM-based targets:
  `hw_out="Tracepoint Type=Stop Tracing"`
- for RVISS targets:
  `hw_out:"Tracepoint Type=Trace End Point"`

Start of Trace Range (Instruction Only)

```
hw_out="Tracepoint Type=Trace Instr"
```

Start of Trace Range (Instruction and Data)

```
hw_out="Tracepoint Type=Trace Instr and Data"
```

Start of Excluded Trace Range (Instruction and Data)

```
hw_out="Tracepoint Type=Trace Instr",hw_not=addr
```

Start of Excluded Trace Range (Data Only)

```
hw_out="Tracepoint Type=Trace Instr and Data",hw_not=addr
```

ExternalOut Points

```
hw_out="Tracepoint Type=ExternalOut1"
hw_out="Tracepoint Type=ExternalOut2"
hw_out="Tracepoint Type=ExternalOut3"
hw_out="Tracepoint Type=ExternalOut4"
```

For more details on the `TRACEINSTREXEC` command

See also:

- *Setting a trigger point* on page 6-4
- *Setting a trace start and end point* on page 6-5
- *Setting a trace range* on page 6-7
- *Setting an ExternalOut Point* on page 6-10

- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the `TRACEINSTREXEC` command.

6.8 Setting a trace range for selected source lines or disassembly

You can set a trace range for selected lines of source or disassembly. To do this:

1. Highlight a range of source lines in the selected source code tab, or lines of disassembly in the **Disassembly** tab.

   ——— **Note** ———

   Take care when highlighting lines of code. Selecting multiple functions might result in unexpected behavior, depending on where in memory these functions are placed. That is, functions might not follow each other in memory, so you might end up tracing a lot more besides the functions you have selected.

   ————————

2. Right-click in the gray bar to the left of the source or disassembly to display the context menu.
3. Select **Set Trace Range** from the context menu.

   ——— **Note** ———

   This option is available only for ETM-based targets.

   ————————

RealView Debugger ensures the trace information is captured only for the range you have selected. Only program instructions in the range are captured. Any instructions executed outside the range are not captured. These discontinuities are represented as `Trace Pause` status lines in the Analysis window.

See also:

- *CLI command used for setting an instruction-only trace range*.

6.8.1 CLI command used for setting an instruction-only trace range

The equivalent CLI command used is:

- in the **Disassembly** tab:

  `TRACE,range` *`start_address..end_address`*

- in a source code tab:

  `TRACE,range \`*`MODULE`*`\#`*`start_line`*`..\`*`MODULE`*`\#`*`end_line`*

If you also want to trace data, then use the following command:

`TRACE,range,data` *`address_range`*.

You can also set trace ranges using other trace commands (such as `TRACEINSTREXEC`).

See also

- *Status messages in the captured trace* on page 9-3
- the following in the *RealView Debugger Command Line Reference Guide*:
  — *Alphabetical command reference* on page 2-12 for details of the `TRACEINSTREXEC` command.

6.9 Setting unconditional tracepoints with the Create Tracepoint dialog box

You can use the Create Tracepoint dialog box to set unconditional tracepoints manually. To do this, you only have to set the following parameters:

- tracepoint type
- tracepoint comparison type
- **when** (address or address range).

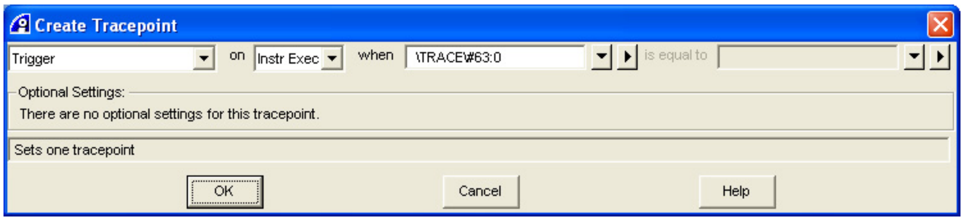
The parameters available depend on your target. This section describes the settings to use to set the equivalent tracepoints that are available on the New Tracepoint dialog box.

See also:

- *RVISS targets*
- *ETM-based targets* on page 6-15.

6.9.1 RVISS targets

If you are tracing a RVISS target, the Create Tracepoint dialog box shown in Figure 6-5 is used.



There are no optional settings for this tracepoint.

Sets one tracepoint

Figure 6-5 Create Tracepoint dialog box (RVISS targets)

Table 6-1 shows the values to use in the Create Tracepoint dialog box to set the tracepoint equivalents described in:

- *Setting a trigger point* on page 6-4
- *Setting a trace start and end point* on page 6-5.

Table 6-1 Parameters for manually setting unconditional tracepoints on RVISS targets

Type of tracepoint	Tracepoint Type setting	Tracepoint Comparison Type setting	when setting
Set Trigger	**Trigger**	**Instr Exec**	***address***
Trace Start Point (Instruction Only)	**Trace Start Point (Instr)**	**Instr Exec**	***address***
Trace Start Point (Instruction and Data)	**Trace Start Point (Instr+Data)**	**Instr Exec**	***address***
Trace End Point	**Trace End Point**	**Instr Exec**	***address***

See also

- *Setting a trigger point* on page 6-4
- *Setting a trace start and end point* on page 6-5
- *Setting a trace range* on page 6-7
- *Setting an ExternalOut Point* on page 6-10.

6.9.2 ETM-based targets

If you are tracing an ETM-based target, the Create Tracepoint dialog box shown in Figure 6-6 is used.

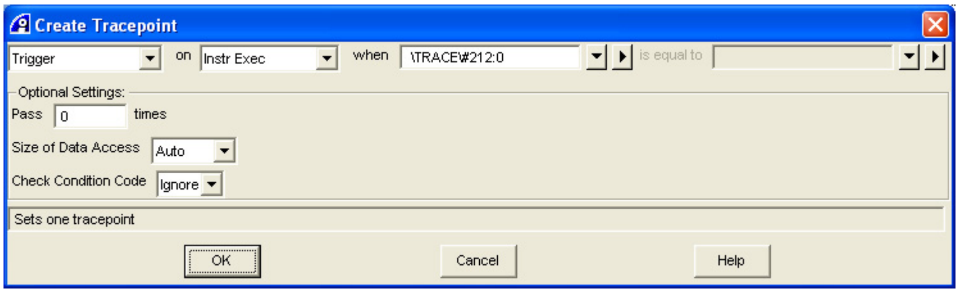


Figure 6-6 Create Tracepoint dialog box (ETM-based targets)

Table 6-2 shows the values to use in the Create Tracepoint dialog box to set the tracepoint equivalents described in:

- *Setting a trigger point* on page 6-4
- *Setting a trace start and end point* on page 6-5
- *Setting a trace range* on page 6-7
- *Setting an ExternalOut Point* on page 6-10.

——— **Note** ———

To set the equivalent end range address for trace ranges, specify the address range as `start_address..end_address`. The end range address must be greater than the start range address. Use an end range address of `0xFFFFFFFF` if you want to trace to the end of the image.

Table 6-2 Parameters for manually setting unconditional tracepoints on ETM-based targets

Type of tracepoint	Tracepoint Type setting	Tracepoint Comparison Type setting	when setting
Trigger	**Trigger**	**Instr Exec**	***address*** or ***address_range***
Trace Start Point	**Start Tracing**	**Instr Exec**	***address***
Trace End Point	**Stop Tracing**	**Instr Exec**	***address***
Trace Range	**Trace Instr**	**Instr Exec**	***address_range***
Start of Trace Range (Instruction Only)	**Trace Instr**	**Instr Exec**	***address_range***
Start of Trace Range (Instruction and Data)	**Trace Instr and Data**	**Instr Exec**	***address_range***
Start of Excluded Trace Range (Instruction and Data)	**Trace Instr**	**Instr Exec**	**NOT*address_range***
Start of Excluded Trace Range (Data Only)	**Trace Instr and Data**	**Instr Exec**	**NOT*address_range***
Trace ExternalOut1 Point, Trace ExternalOut2 Point, Trace ExternalOut3 Point, or Trace ExternalOut4 Point	**ExternalOut1**, **ExternalOut2**, **ExternalOut3**, or **ExternalOut4**	**Instr Exec**	***address*** or ***address_range***

See also

- *Setting a trigger point* on page 6-4
- *Setting a trace start and end point* on page 6-5
- *Setting a trace range* on page 6-7
- *Setting an ExternalOut Point* on page 6-10.

6.10 Setting tracepoints from your Favorites List

When you first start to use RealView Debugger on Windows, your personal Favorites List is empty. You can create tracepoints and add them to this list or you can add tracepoints that you have been using in the current debugging session.

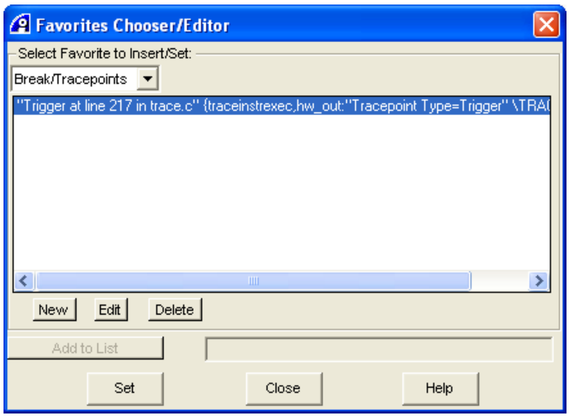
RealView Debugger keeps a record of all tracepoints that you set during your debugging session as part of your history file. By default, at the end of your debugging session, these processor-specific lists are saved in the file `exphist.sav` in your RealView Debugger home directory. This file also keeps a record of your favorites, for example Break/Tracepoints.

Note

You must connect to a target before you can use or modify your Favorites List.

To set a tracepoint from your favorites list:

1. Select **Debug** → **Breakpoints** → **Set Break/Tracepoint from List** → **Break/Tracepoint Favorites...** from the Code window main menu to display the Favorites Chooser/Editor dialog box. Figure 6-7 shows an example. Any previous tracepoints that have been added to the list are shown.


Favorites Chooser/Editor

Select Favorite to Insert/Set: Break/Tracepoints

"Trigger at line 217 in trace.c" {traceinstrexec,hw_out:"Tracepoint Type=Trigger" \TRA

Figure 6-7 Favorites/Chooser Editor dialog box

2. Select the tracepoint that you want to set from your favorites list.
3. Click **Set** to set the tracepoint and close the dialog box.

Chapter 7
Setting Conditional Tracepoints

This chapter describes how to set conditional tracepoints. It includes:

7.1 About setting conditional tracepoints

Conditional tracepoints enable you to test for various conditions that must be met before trace capture can begin. Conditional tracepoints are available only for targets with *Embedded Trace Macrocell*™ (ETM™). You cannot set conditional tracepoints on *RealView ARMulator® ISS* (RVISS) targets.

A number of dialog boxes are provided to help you set conditional tracepoints:

- The dialog boxes used in the procedures described in this chapter have a set of common parameters. Parameters that are specific to a dialog box are described in the procedures that use the dialog box.
- When you set a tracepoint using these dialog boxes, RealView Debugger also generates the equivalent trace commands with qualifiers. The parameters that determine which trace command and qualifier is generated is also shown in the following sections.

See also:

- *Common parameters for setting conditional tracepoints* on page 7-3
- Chapter 4 *Configuring the ETM*
- Chapter 5 *Tracepoints in RealView Debugger*
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands*.

7.2 Common parameters for setting conditional tracepoints

In each of the dialog boxes that are available for setting tracepoints, you can:

- set the tracepoint type from a drop-down list
- set the tracepoint comparison type from a drop-down list
- enter the required addresses, address ranges, or data address comparison
- test for a data value
- test for an external condition.

Each parameter description also shows which part of the generated trace command is added by that parameter.

See also:

- *Tracepoint types*
- *Tracepoint comparison types* on page 7-4
- *Entering addresses, address ranges, and data address comparisons* on page 7-6
- *Entering data value comparisons* on page 7-7
- *External Conditions* on page 7-8.

7.2.1 Tracepoint types

The following tracepoint types are available:

Trigger Sets an explicit trigger point on the selected address.

You can also use the **Collect trace**... options to specify how the trace is collected when the trigger is hit.

Start Tracing

Starts tracing at the selected address.

Note

This type is available only on the Create Tracepoint dialog box, and only for ETM-based targets.

Stop Tracing

Stops tracing at the selected address.

Note

This type is available only on the Create Tracepoint dialog box.

Trace Instr Traces instructions only.

Trace Instr and Data

Traces both instructions and data.

ExternalOut1-4

Controls single-bit output signals from the ETM. Up to four signals are available. The ASIC manufacturer determines the availability and usage of these output signals. See your ASIC documentation for details.

The tracepoint type adds the `hw_out:"Tracepoint Type=`*`type`*`"` qualifier to the CLI command that is generated by RealView Debugger when it sets a tracepoint.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Extended external input selection (ETMv3.1 and later)* on page 4-9

- *Trace Range* on page 5-5
- *External Conditions* on page 7-8
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the trace commands.

7.2.2 Tracepoint comparison types

The tracepoint comparison type specifies the action that activates the tracepoint. The following tracepoint comparison types are available:

Instr Exec The address of each instruction that is presented to the execution unit is compared against the address you specify (even though the instruction might not be executed if its condition code evaluates to False).

This generates a `TRACEINSTREXEC` command.

Instr Fetch The address of each instruction fetched is compared against the address you specify.

This generates a `TRACEINSTRFETCH` command.

Data Read The meaning of this type depends on whether or not you specify a data value:

- If you do not specify data value, then the address of the data read from is compared against the address you specify.
- If you specify a data value, then the data value read from the specified address is compared against the value you specified.

This generates a `TRACEDATAREAD` command.

Data Write The meaning of this type depends on whether or not you specify a data value:

- If you do not specify data value, then address of the data written to is compared against the address you specify.
- If you specify a data value, then the data written to the specified address is compared against the value you specified.

This generates a `TRACEDATAWRITE` command.

Data Access The meaning of this type depends on whether or not you specify a data value:

- If you do not specify data value, then address of the data accessed, in either a read or write direction, is compared against the address you specify.
- If you specify a data value, then the data value accessed at the specified address, in either a read or write direction, is compared against the value you specified.

This generates a `TRACEDATAACCESS` command.

External Condition

The tracepoint is activated on one of the following events:

- external inputs 1 to 4
- extended external inputs 1 to 4 (for ETMv3.1, and later)
- EmbeddedICE® watchpoints 1 and 2
- access to ASIC memory maps 1 to 16.

The ASIC manufacturer determines the availability and usage of these input signals. See your ASIC documentation for details.

This generates a `TRACEEXTCOND` command.

Note

Not all tracepoint comparison types are available for all tracepoint units. Also, availability is determined by what options you have already set.

Parameters available for each comparison type

The tracepoint comparison type determines which parameters you can set, shown in Table 7-1. The **Ignore Security Level** parameter is available only on targets that support TrustZone® technology.

Table 7-1 Parameters available for each comparison type

Comparison type	Available parameters
Instr Exec	**when** (location specifier) **Pass times** **Size of Data Access** **Check Condition Code** **Ignore Security Level**
Instr Fetch	**when** (location specifier) **Pass times** **Size of Data Access** **Ignore Security Level**
Data Read	**when** (address or address range) **is equal to** (optional value) **Pass times** **Size of Data Access** **Ignore Security Level**
Data Write	**when** (address or address range) **is equal to** (optional value) **Pass times** **Size of Data Access** **Ignore Security Level**
Data Access	**when** (address or address range) **is equal to** (optional value) **Pass times** **Size of Data Access** **Ignore Security Level**
External Condition	**when** (External Condition) **Pass times**

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Extended external input selection (ETMv3.1 and later)* on page 4-9
- *Trace Range* on page 5-5
- *External Conditions* on page 7-8
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the trace commands.

7.2.3 Entering addresses, address ranges, and data address comparisons

You can enter addresses, address ranges, and data address comparisons in the following ways:

- Click the drop-down arrow ▾ to display the address menu. You can choose from:
  - — a Function List dialog box, which provides a list of the functions in your image
  - — a Variable List dialog box, which provides a list of the variables in your image
  - — a Module/File List dialog box, which provides a list of the modules in your image
  - — a Register List dialog box, which provides a list of the memory mapped registers defined in any BCD files assigned to the connection for the target
  - — a Favorites/Chooser Editor dialog box, which provides a list of any addresses you have added to your personal Favorites List
  - — a list of previously-used expressions.

  The options shown here depend on your debug target and connection.

- Type the required address, address range, or data address comparison into the text box. Click the right arrow ▸ to get help with the syntax of the entry if required. The following options are available:

  Address Range If you select this option when the text box is empty, RealView Debugger inserts `start..end` into the text box. Replace `start` with the start address and `end` with the end address.

  If you select this option when the text box contains a value, RealView Debugger takes this value as the start address, and inserts `..` after the value. Enter the end address.

  Address Range by Length

  If you select this option when the text box is empty, RealView Debugger inserts `start..+len` into the text box. Replace `start` with the start address and `len` with the required offset value in bytes.

  If you select this option when the text box contains a value, RealView Debugger takes this value as the start address, and inserts `..+` after the value. Enter the required offset value in bytes.

  NOT Address Compare

  When you select this option, RealView Debugger inserts `$NOT$` into the text box. If the text box already contains an address or address range, `$NOT$` is inserted before it. Otherwise, enter the address or address range after `$NOT$`, for example:

  ```
  $NOT$0x8000..0x8100
  ```

  You can use this option to set up excluded trace ranges (see Table 6-2 on page 6-15).

  This adds the `hw_not:addr` command qualifier.

  Autocomplete Range

  This option generates an auto-range from an expression that you enter, which can be any of:

  - A function name, where the generated address range is from the start-to-end of the function.
  - A structure, where the generated address range is from the start-to-end of the structure.

- An array symbol, where the generated address range is from the start of the variable to the end, where the end is the *start*+*sizeof*(*var*). For example, if the start address is `0x8000`, and the array size is 16 bytes, the end address is considered to be `0x8010` (that is, `0x8000`+16).

RealView Debugger filters the information down to only rows represented by the generated auto-range.

Enter a symbol and then click this option to compute the end-of-range address based on the symbol size. For example, if you enter a function then the autocompleted range is from the start of the function to the end. Similarly, enter a global variable to see the end-of-range address autocompleted as the variable storage address plus variable size.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Extended external input selection (ETMv3.1 and later)* on page 4-9
- *Trace Range* on page 5-5
- *External Conditions* on page 7-8
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the trace commands.

7.2.4 Entering data value comparisons

You can enter values to compare against. This adds the `hw_dvalue:` command qualifier.

Note

The options available depend on your target.

Enter a value in the following ways:

- Click the drop-down arrow ▾ to display the data value menu. You can choose from:
  - a Function List dialog box, which provides a list of the functions in your image
  - a Variable List dialog box, which provides a list of the variables in your image
  - a Module/File List dialog box, which provides a list of the modules in your image
  - a Register List dialog box, which provides a list of the memory mapped registers defined in any BCD files assigned to the connection for the target
  - a Favorites/Chooser Editor dialog box, which provides a list of any addresses you have added to your personal Favorites List
  - a list of previously-used expressions.

  The options shown here depend on your debug target and connection.

- Type the required value into the text box. Click the right arrow ▸ to get help with the syntax of the entry if required. The following options are available:

  Value Mask The value mask enables you to specify individual bits to test when comparing values. Testing is performed on the following basis:
  - a binary zero in the filter indicates that the bit is not tested
  - a binary one in the filter indicates that the corresponding bit of the transfer is compared with the corresponding bit of the Data value.

When you select this option, RealView Debugger inserts `$MASK$=0xFFFFFFFF` into the text box. Enter the value you want to compare against, and edit the mask to the required value.

This adds the `hw_dmask:`*`mask`* command qualifier.

NOT Value Compare

When you select this option, RealView Debugger inserts `$NOT$` into the text box. If the text box already contains a value, `$NOT$` is inserted before it. Otherwise, enter the value after `$NOT$`, for example:

`$NOT$0x1000`

This adds the `hw_not:data` command qualifier.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Extended external input selection (ETMv3.1 and later)* on page 4-9
- *Trace Range* on page 5-5
- *External Conditions*
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the trace commands.

7.2.5 External Conditions

External conditions are specific to your ASIC implementation, and correspond to the signals on your ASIC. See your ASIC documentation for details. The conditions are available only for the tracepoint comparison type **External Condition**. You must select the condition from the drop-down menu that is displayed when you click the drop-down arrow ▾ of the **when** parameter.

The following external conditions are available:

ExternalIn1-4

Up to four external inputs are available, depending on your ETM configuration. The ASIC manufacturer determines the availability and use of these output signals. External inputs can be any combination of logic that evaluates to True or False.

Extended ExternalIn1-4

For ETMv3.1, and later, up to four extended external inputs are available, depending on your ETM configuration. The ASIC manufacturer determines the availability and use of these output signals. Extended external inputs can be any combination of logic that evaluates to True or False.

Watchpoint1-2

These resources are the watchpoint units in the EmbeddedICE macrocell. They are used when you set hardware watchpoints or hardware breakpoints. This functionality is only available when the signals from the EmbeddedICE are connected to the ETM. The ASIC manufacturer determines the availability and use of these output signals.

ASIC MemMap 1-16

These resources are only available if you have ASIC memory map decode hardware installed in your ASIC. Each implementation of ASIC memory map decode hardware has been defined by the ASIC manufacturer to return True when instructions are being fetched from a particular address range.

These options add the `hw_in:"External Condition=`*`condition`*`"` command qualifier to the CLI command generated by RealView Debugger.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Extended external input selection (ETMv3.1 and later)* on page 4-9
- *Trace Range* on page 5-5
- *External Conditions* on page 7-8
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the trace commands.

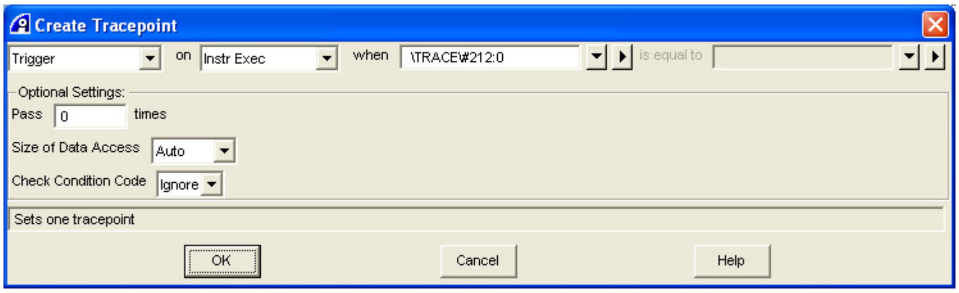
7.3 Setting a tracepoint to trace a specific memory access

For ETM-based targets, you can limit the trace capture to specific types of memory accesses:

- read-only access
- write-only access
- any access.

To limit the trace capture to specific types of memory access:

1. In the source or disassembly view, right-click on the gray margin to the left of the required location to display context menu.
2. Select **Create Tracepoint...** from the context menu to display the Create Tracepoint dialog box. Figure 7-1 shows an example:



Create Tracepoint

Trigger on Instr Exec when \TRACEW#212:0 is equal to

Optional Settings:

Pass 0 times

Size of Data Access: Auto

Check Condition Code: Ignore

Sets one tracepoint

Figure 7-1 Create Tracepoint dialog box

3. Select the required tracepoint type from the first drop-down list.
4. Select the **Data Read**, **Data Write**, or **Data Access** tracepoint comparison type from the second drop-down list. The tracepoint comparison type determines which of the remaining parameters you can set.
5. Specify the location at which the specified data access is to be traced. This can be an a single address or a range of addresses.
6. Leave the **is equal to** field blank so that all accesses at the address are traced.
7. Click **OK** to set the tracepoint and close the Create Tracepoint dialog box.

See also:

- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4
- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6.

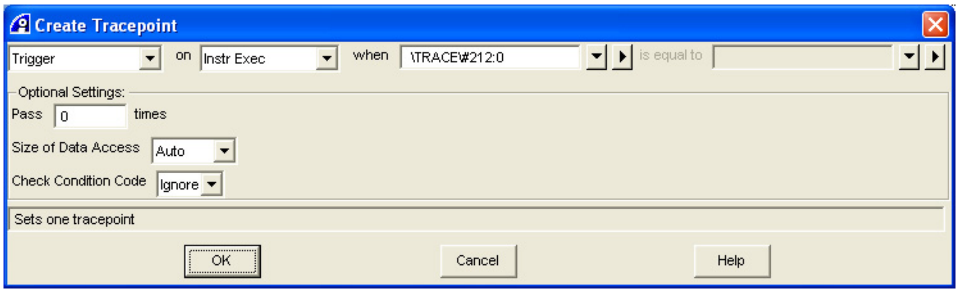
7.4 Setting a tracepoint to trace a specific data value

For ETM-based targets, you can specify that trace is captured only when a particular data value is accessed:

- data value reads only
- data value writes only
- data value reads and writes.

To limit the trace capture to specific types of memory access:

1. In the source or disassembly view, right-click on the gray margin to the left of the required location to display context menu.
2. Select **Create Tracepoint...** from the context menu to display the Create Tracepoint dialog box. Figure 7-2 shows an example:



Create Tracepoint

Trigger on Instr Exec when \TRACEW#212:0 is equal to

Optional Settings:

Pass 0 times

Size of Data Access Auto

Check Condition Code Ignore

Sets one tracepoint

Figure 7-2 Create Tracepoint dialog box

3. Select the required tracepoint type from the first drop-down list.
4. Select the **Data Read**, **Data Write**, or **Data Access** tracepoint comparison type from the second drop-down list. The tracepoint comparison type determines which of the remaining parameters you can set.
5. Specify the location at which the specified data value is to be traced. This can be an a single address or a range of addresses.
6. Enter the required data value to be traced in the **is equal to** field.
7. Click **OK** to set the tracepoint and close the Create Tracepoint dialog box.

See also

- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4
- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6.

7.5 Setting tracepoints to test for external conditions

For ETM-based targets, you can test for external conditions.

To test for external conditions:

1. In the source or disassembly view, right-click on the gray margin to the left of the required location to display context menu.
2. Select **Create Tracepoint...** from the context menu to display the Create Tracepoint dialog box. Figure 7-3 shows an example:

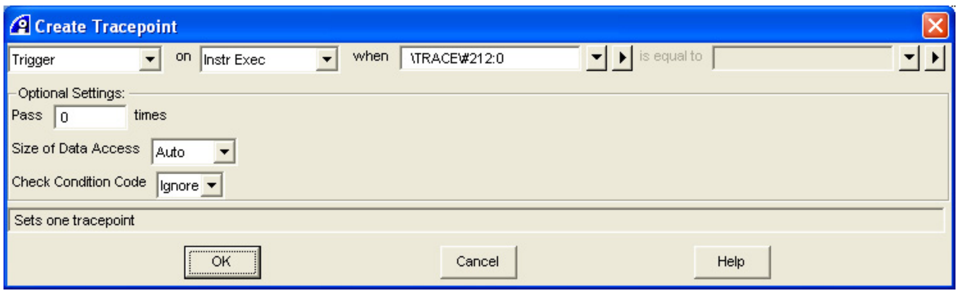

Create Tracepoint
Trigger on Instr Exec when \TRACE#212:0 is equal to
Optional Settings:
Pass 0 times
Size of Data Access Auto
Check Condition Code Ignore
Sets one tracepoint

Figure 7-3 Create Tracepoint dialog box

3. Select the required tracepoint type from the first drop-down list.
4. Select **External Condition** for the tracepoint comparison type from the second drop-down list. The tracepoint comparison type determines which of the remaining parameters you can set.
5. Click the drop-down arrow ▾ of the **when** field to display the external conditions menu.
6. Select the required condition to test.
7. If required, enter a **Pass** count.
8. Click **OK** to set the tracepoint and close the Create Tracepoint dialog box.

See also:

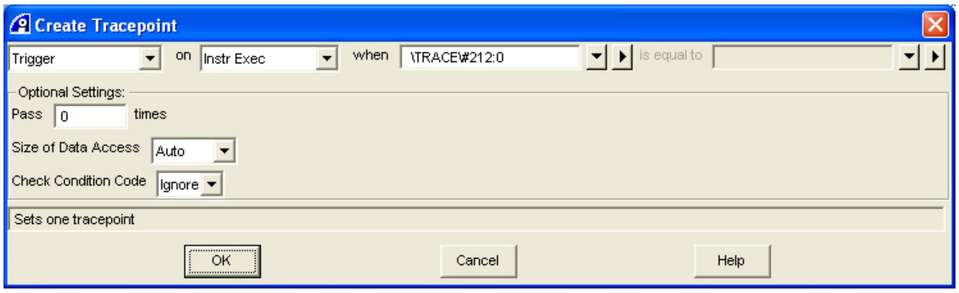
- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4
- *Parameters available for each comparison type* on page 7-5
- *External Conditions* on page 7-8
- *Setting a tracepoint that activates after a number of passes* on page 7-13.

7.6 Setting a tracepoint that activates after a number of passes

For ETM-based targets, you can delay the activation of a tracepoint until it has been passed over a specific number of times.

To delay the activation of a tracepoint:

1. In the source or disassembly view, right-click on the gray margin to the left of the required location to display context menu.
2. Select **Create Tracepoint...** from the context menu to display the Create Tracepoint dialog box. Figure 7-4 shows an example:



Create Tracepoint

Trigger on Instr Exec when \TRACE\#212:0 is equal to

Optional Settings:

Pass 0 times

Size of Data Access Auto

Check Condition Code Ignore

Sets one tracepoint

Figure 7-4 Create Tracepoint dialog box

3. Select the required tracepoint type from the first drop-down list.
4. Select the required tracepoint comparison type from the second drop-down list. The tracepoint comparison type determines which of the remaining parameters you can set.
5. Specify the address or address range to be traced as appropriate:
   - a line number in the source code, with or without a module name prefix, for example `\TRACE\#212:0`
   - an address or address range
   - a function name, for example, `Func_1`
   - a function entry point, for example, `Func_1\@entry`.
6. Enter the required number of passes into the **Pass** text box. If you do not set this item, it defaults to zero (`0`), and the tracepoint is activated on every pass.

   ——— Note ———

   If you specify a range in the `when` location field, then the ETM counters are decremented on every cycle executed in that range. ETM counters can be in the range 0 to 65535.

   ———————————

7. Set any other parameters as required.
8. Click **OK** to set the tracepoint and close the Create Tracepoint dialog box.

See also:

- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4
- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6
- the following in the *RealView Debugger Command Line Reference Guide*:
  — *Using variables in the debugger* on page 1-28 for details of the `@entry` symbol.

7.7 Setting tracepoints to trace accesses of a specific size

For ETM-based targets, you can capture trace for specific access sizes:

- for data accesses, the size of the data transfer
- for instruction accesses, the size of the instruction accessed.

To capture data and instruction accesses of a specific size:

1. In the source or disassembly view, right-click on the gray margin to the left of the required location to display context menu.
2. Select **Create Tracepoint...** from the context menu to display the Create Tracepoint dialog box. Figure 7-5 shows an example:

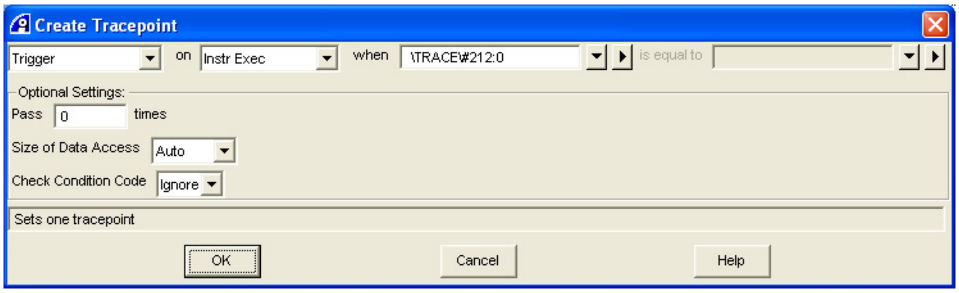


Figure 7-5 Create Tracepoint dialog box

3. Select the required tracepoint type from the first drop-down list.
4. Select the required tracepoint comparison type from the second drop-down list. The tracepoint comparison type determines which of the remaining parameters you can set.

   Note

   Do not select **External Condition**, because the **Size of Data Access** field is not supported for this type.

5. Specify the address or address range to be traced as appropriate:
   - a line number in the source code, with or without a module name prefix, for example `\TRACE\#212:0`
   - an address or address range
   - a function name, for example, `Func_1`
   - a function entry point, for example, `Func_1\@entry`.
6. Choose the required **Size of Data Access**. The following options are available:

   Auto To check byte addresses depending on the implementation:
   - halfword for Thumb® code
   - word for ARM® code.

   This is the default.

   For ETMv3, you must select **Halfword** for Thumb data accesses. Otherwise, the address comparators might activate unexpectedly.

   Byte To check byte addresses (Thumb code).

   Halfword

   To check both byte addresses in the halfword (Thumb code). This must be used, for example, if you are interested in byte accesses to either byte of a halfword.

Word To check all four byte addresses in the same word (ARM code). This must be used, for example, if you are interested in byte accesses to any byte of a word.

——— **Note** ———

This parameter adds the `hw_in:"Size of Data Access=`*`size`*`"` command qualifier to the CLI command that is generated by RealView Debugger when it sets a tracepoint.

———

7. Set any other parameters as required.
8. Click **OK** to set the tracepoint and close the Create Tracepoint dialog box.

See also:

- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4
- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Using variables in the debugger* on page 1-28 for details of the `@entry` symbol
  - *Alphabetical command reference* on page 2-12.

7.8 Setting tracepoints to trace instruction execution status

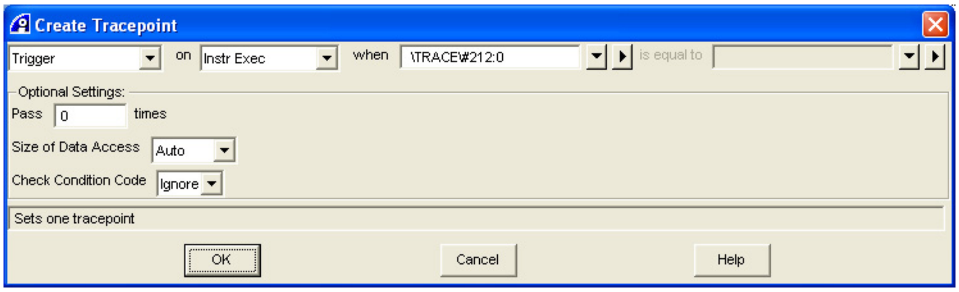
For ETM-based targets, you can capture trace for a specific execution status of an address. The address of each instruction that reaches the Execute stage of the pipeline is compared against the address you specify (the instruction has not been executed if its condition code evaluates to False).

Note

This condition is not available if your target hardware supports ETM architecture version 1.0 (ETMv1.0) or ETMv1.1.

To capture instructions depending on the execution status:

1. In the source or disassembly view, right-click on the gray margin to the left of the required location to display context menu.
2. Select **Create Tracepoint...** from the context menu to display the Create Tracepoint dialog box. Figure 7-6 shows an example:


Create Tracepoint

Trigger on Instr Exec when \TRACE\#212:0 is equal to

Optional Settings:

Pass 0 times

Size of Data Access: Auto

Check Condition Code: Ignore

Sets one tracepoint

Figure 7-6 Create Tracepoint dialog box

3. Select the required tracepoint type from the first drop-down list.
4. Select **Instr Exec** for the tracepoint comparison type from the second drop-down list. The tracepoint comparison type determines which of the remaining parameters you can set.

   Note

   Do not select **External Condition**, because the **Check Condition Code** field is not supported for this type.

   ---

5. Specify the address of the instruction to be traced. You can also specify a range of addresses to trace multiple instructions. You can enter:
   - a line number in the source code, with or without a module name prefix, for example `\TRACE\#212:0`
   - an address or address range
   - a function name, for example, `Func_1`
   - a function entry point, for example, `Func_1\@entry`.
6. Choose the required **Check Condition Code**. The following options are available:

Ignore	To match any execution status. This is the default.
Pass	To match only instructions that executed.
Fail	To match only instructions that did not execute.

Note

This parameter adds the `hw_out:"Check Condition Code=`*`code`*`"` command qualifier to the CLI command that is generated by RealView Debugger when it sets a tracepoint.

7. Set any other parameters as required.
8. Click **OK** to set the tracepoint and close the Create Tracepoint dialog box.

See also:

- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4
- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Using variables in the debugger* on page 1-28 for details of the `@entry` symbol
  - *Alphabetical command reference* on page 2-12.

7.9 Setting tracepoints to trace accesses on a specific security level

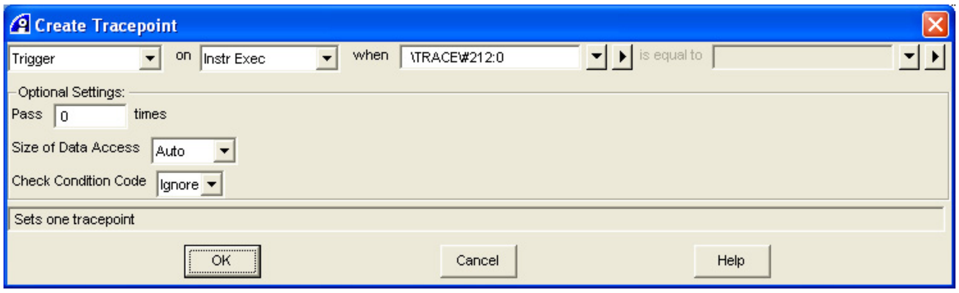
For ETM-based targets that support TrustZone technology, then you can limit the trace capture to addresses in:

- Secure memory space only
- Normal memory space only
- any memory space.

To limit the trace capture to addresses in a specific memory space for targets that support TrustZone technology:

1. In the source or disassembly view, right-click on the gray margin to the left of the required location to display context menu.

2. Select **Create Tracepoint...** from the context menu to display the Create Tracepoint dialog box. Figure 7-7 shows an example:



Create Tracepoint

Trigger on Instr Exec when \TRACE\#212:0 is equal to

Optional Settings:
Pass 0 times
Size of Data Access Auto
Check Condition Code Ignore

Sets one tracepoint

Figure 7-7 Create Tracepoint dialog box

3. Select the required tracepoint type from the first drop-down list.

4. Select the required tracepoint comparison type from the second drop-down list. The tracepoint comparison type determines which of the remaining parameters you can set.

5. Specify the location to be traced. The address prefix determines the specific memory space:

   - to capture trace in the Secure memory space, include the `S:` address prefix, for example `S:0x8000` or `S:\TRACE\#212:0`
   - to capture trace in the Normal memory space, include the `N:` address prefix, for example `N:Func_1`
   - to capture trace in any memory space, do not include an address prefix, for example `0x8000`.

   This can be an a single address or a range of addresses:

   - a line number in the source code, with or without a module name prefix, for example `\TRACE\#212:0`
   - an address or address range
   - a function name, for example, `Func_1`
   - a function entry point, for example, `Func_1\@entry`.

6. Choose the required setting for **Ignore Security Level**. The following options are available:

Yes	Capture trace when the processor is in any memory space. This is the default.
No	Capture trace only when the processor is in either Normal or Secure memory space.

Note

This parameter adds the `hw_in:"Ignore Security Level=`*`level`*`"` command qualifier to the CLI command that is generated by RealView Debugger when it sets a tracepoint.

---

7. Set any other parameters as required.
8. Click **OK** to set the tracepoint and close the Create Tracepoint dialog box.

See also:

- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4
- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Using variables in the debugger* on page 1-28 for details of the `@entry` symbol
  - *Alphabetical command reference* on page 2-12.

7.10 Setting a Trace on X after Y [and/or Z] tracepoint chain

You can define a tracepoint chain that is only active when one or more specified conditions have been met. For example, you can set a tracepoint chain that is only active when the first specified function has been executed but the second has not.

To set a conditional Trace on X after Y [and/or Z] tracepoint:

1. Select **Debug** → **Tracepoints** → **Trace on X after Y [and/or Z]...** from the Code window main menu to display the Trace on X after Y [and/or Z] dialog box. Figure 7-8 shows an example:

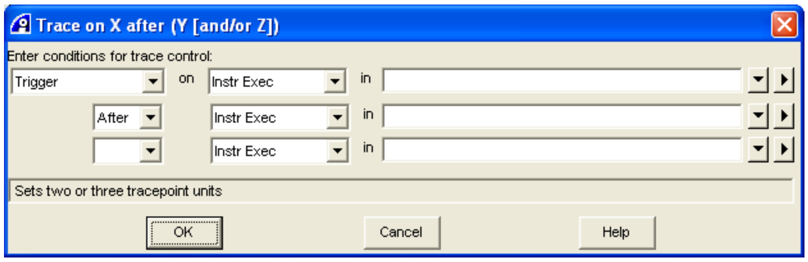

Trace on X after (Y [and/or Z])
Enter conditions for trace control:
Trigger
on
Instr Exec
in
After
Instr Exec
in
Instr Exec
in
Sets two or three tracepoint units
OK
Cancel
Help

Figure 7-8 Trace on X after Y [and/or Z] dialog box

——— **Note** ———

This dialog box is available only if you are using an ETM-based target.

———————————

2. For the first tracepoint unit:

   a. Select the type of tracepoint type that you want to set, for example **Trigger**.

   b. Select the tracepoint comparison type, for example **Data Access**.

   c. Specify the location to test. This can be:

      - a line number in the source code, with or without a module name prefix, for example `\TRACE\#56:0`
      - the address of an instruction, variable, or data value, or a range of addresses
      - a function name, for example, `Func_1`
      - a function entry point, for example, `Func_1\@entry`.

      The tracepoint unit triggers if the `PC` equals the corresponding address, or falls within the specified address range.

   This tracepoint unit adds the `hw_and:"then-next"` command qualifier to the CLI command generated by RealView Debugger, and is the first tracepoint in the chain.

3. For the second tracepoint unit:

   a. Select on of the following from the drop-down list:

      - **After** if you want the tracepoint to trigger after the specified event has occurred
      - **Before** if you want the tracepoint to trigger before the specified event has occurred.

   b. Choose the tracepoint type and address range in the same way as for the first tracepoint unit.

   This tracepoint unit adds the `hw_and:"then-prev"` command qualifier to the CLI command generated by RealView Debugger, and is the second tracepoint in the chain. If you selected **Before**, then it also adds the `hw_not:then` command qualifier.

4. If you want to specify a third tracepoint unit:
   a. Select **AND** or **OR** from the first drop-down list.
   b. Choose the tracepoint type and address range in the same way as for the first tracepoint unit.

   This tracepoint unit adds the following command qualifiers to the CLI command generated by RealView Debugger, and is the third tracepoint in the chain:
   - `hw_and:prev` command qualifier, if **AND** is selected
   - `hw_and:"then-h1"` command qualifier, if **OR** is selected.
   - `hw_not:then` command qualifier, if **Before** is selected.
5. Click **OK** to set the tracepoint as specified.

Example commands issued by the Trace on X after Y [and/or Z] dialog box:

```
trciexec,hw_and:"then-next",hw_out:"Tracepoint Type=Trigger" 0x8000
trciexec,hw_and:"then-prev" 0x8010
trciexec,hw_and:"then-h1" 0x8020
```

See also:
- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4
- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6
- *Chaining tracepoints* on page 7-28
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Using variables in the debugger* on page 1-28 for details of the `@entry` symbol
  - *Alphabetical command reference* on page 2-12.

7.11 Setting a Trace on X after Y executed N times tracepoint chain

You can set a tracepoint chain that becomes active when the secondary condition has been met a specified number of times.

To set a conditional Trace on X after Y executed N times tracepoint:

1. Select **Debug** → **Tracepoints** → **Trace on X after Y executed N times...** from the Code window main menu to display the Trace on X after Y executed N times dialog box. Figure 7-9 shows an example:

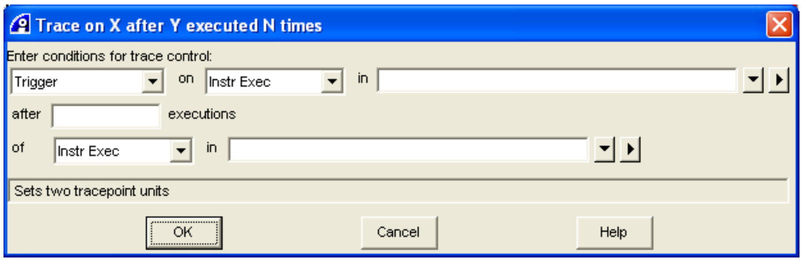

Trace on X after Y executed N times
Enter conditions for trace control:
Trigger
on
Instr Exec
in
after
executions
of
Instr Exec
in
Sets two tracepoint units
OK
Cancel
Help

Figure 7-9 Trace on X after Y executed N times dialog box

Note

This dialog box is available only if you are using an ETM-based target.

2. For the first tracepoint unit:
   a. Select the type of tracepoint type that you want to set, for example **Trigger**.
   b. Select the tracepoint comparison type, for example **Data Access**.
   c. Specify the location to test. This can be:
      - a line number in the source code, with or without a module name prefix, for example `\TRACE\#56:0`
      - the address of an instruction, variable, or data value, or a range of addresses
      - a function name, for example, `Func_1`
      - a function entry point, for example, `Func_1\@entry`.

      The tracepoint unit triggers if the `PC` equals the corresponding address, or falls within the specified address range.

   This tracepoint unit adds the `hw_and:"then-next"` command qualifier to the CLI command generated by RealView Debugger, and is the first tracepoint in the chain.

3. Enter the required number of executions for the secondary condition.

   This adds the `hw_pass:`*`count`* command qualifier to the CLI command generated by RealView Debugger for the second tracepoint in the chain.

4. For the second tracepoint unit:
   a. Select the tracepoint comparison type, for example **Data Access**.
   b. Specify the location to test. The location can be:
      - a line number in the source code, with or without a module name prefix
      - the address of an instruction, variable, or data value, or a range of addresses
      - a function name, for example, `Func_1`
      - a function entry point, for example, `Func_1\@entry`.

      The tracepoint unit triggers if the `PC` equals the corresponding address, or falls within the specified address range.

This tracepoint unit adds the `hw_and:"then-prev"` command qualifier to the CLI command generated by RealView Debugger, and is the second tracepoint in the chain.

5. Click **OK** to set the tracepoint as specified.

Example commands issued by the Trace on X after Y executed N times dialog box:

```
trciexec,hw_and:"then-next",hw_out:"Tracepoint Type=Trigger" main
trciexec,hw_pass:10,hw_and:"then-prev" @dhrystone\\DHRY_1\Proc_5
```

See also:

- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4
- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6
- *Setting up a conditional tracepoint* on page 10-91
- *Chaining tracepoints* on page 7-28
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Using variables in the debugger* on page 1-28 for details of the `@entry` symbol
  - *Alphabetical command reference* on page 2-12.

7.12 Setting a Trace on X after A==B tracepoint chain

You can specify a tracepoint chain that only becomes active after a specified value is written to or read from a specified memory location. For example, you can set a tracepoint on a specified function being executed but only after zero has been written to a specified variable.

To set a conditional Trace on X after A==B tracepoint:

1. Select **Debug** → **Tracepoints** → **Trace on X after A==B...** from the Code window main menu to display the Trace on X after A==B dialog box. Figure 7-10 shows an example:

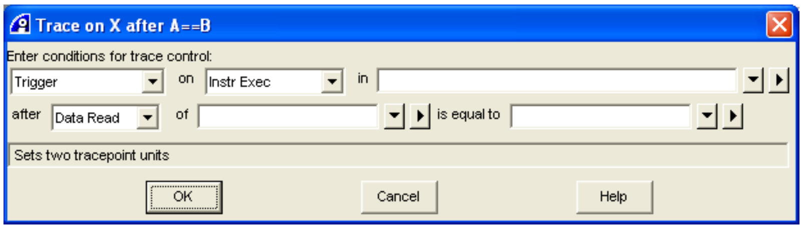


Figure 7-10 Trace on X after A==B dialog box

——— **Note** ———

This dialog box is available only if you are using an ETM-based target.

————————

2. For the first tracepoint unit:
   a. Select the type of tracepoint type that you want to set, for example **Trigger**.
   b. Select the tracepoint comparison type, for example **Data Access**.
   c. Specify the location to test. The location can be:
      - a line number in the source code, with or without a module name prefix, for example `\TRACE\#56:0`
      - the address of an instruction, variable, or data value, or a range of addresses
      - a function name, for example, `Func_1`
      - a function entry point, for example, `Func_1\@entry`.

   This tracepoint unit adds the `hw_and:"then-next"` command qualifier to the CLI command generated by RealView Debugger, and is the first tracepoint in the chain.

3. Set the second tracepoint unit:
   a. Select the tracepoint comparison type, for example **Data Access**.
   b. Specify a variable name or data value address.
   c. Specify the value you want to compare the variable against. You must enter a value in this field. This adds the `hw_dvalue:(`*`value`*`)` qualifier to the CLI command.

   This tracepoint unit adds the `hw_and:"then-prev"` command qualifier to the CLI command generated by RealView Debugger, and is the second tracepoint in the chain.

4. Click **OK** to set the tracepoint as specified.

Example commands issued by the Trace on X after A==B dialog box:

```
trciexec,hw_and:"then-next",hw_out:"Tracepoint Type=Trigger" main
trcdwrite,hw_dvalue:(50),hw_and:"then-prev" @dhrystone\\DHRY_1\main\Run_Index
```

See also:

- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4

- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6
- *Entering data value comparisons* on page 7-7
- *Chaining tracepoints* on page 7-28
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Using variables in the debugger* on page 1-28 for details of the `@entry` symbol
  - *Alphabetical command reference* on page 2-12.

7.13 Setting a Trace if A==B in X tracepoint chain

You can set a tracepoint chain on a specified value being written to or read from a specified address at the same time as another condition is satisfied. For example, you can set a tracepoint on the value of a specified variable being altered by a specified function.

To set a conditional Trace if A==B in X tracepoint:

1. Select **Debug** → **Tracepoints** → **Trace if A==B in X...** from the Code window main menu to display the Trace if A==B in X dialog box. Figure 7-11 shows an example:

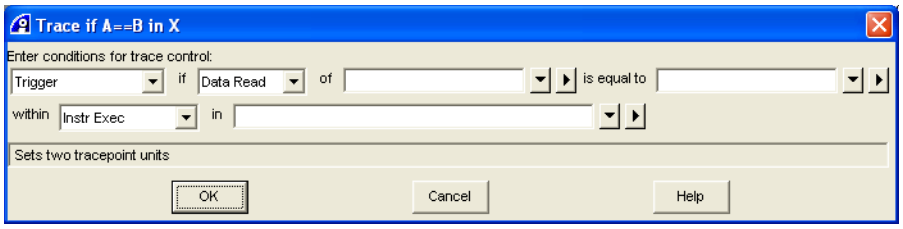


Figure 7-11 Trace if A==B in X dialog box

——— **Note** ———

This dialog box is available only if you are using an ETM-based target.

———————————

2. Set the first tracepoint unit:
   a. Select the type of tracepoint that you want to set, for example **Trigger**.
   b. Select the tracepoint comparison type, for example **Data Read**.
   c. Specify a variable name or data value address.
   d. Specify the value you want to compare against. You must enter a value in this field. This adds the `hw_dvalue:(`*`value`*`)` qualifier to the CLI command.

   This tracepoint unit adds the `hw_and:"then-next"` command qualifier to the CLI command generated by RealView Debugger, and is the first tracepoint in the chain.

3. Set the second tracepoint unit:
   a. Select the tracepoint comparison type, for example **Data Access**.
   b. Specify the location to test. The location can be:
      - a line number in the source code, with or without a module name prefix, for example `\TRACE\#56:0`
      - the address of an instruction, variable, or data value, or a range of addresses
      - a function name, for example, `Func_1`
      - a function entry point, for example, `Func_1\@entry`.

   This tracepoint unit adds the `hw_and:"then-prev"` command qualifier to the CLI command generated by RealView Debugger, and is the second tracepoint in the chain.

4. Click **OK** to set the tracepoint as specified.

Example commands issued by the Trace if A==B in X dialog box:

```
> trcdread,hw_and:"then-next",hw_out:"Tracepoint Type=Trigger",hw_dvalue:(50)
@dhrystone\\DHRY_1\main\Run_Index> trciexec,hw_and:prev main
```

See also:

- *Tracepoint types* on page 7-3
- *Tracepoint comparison types* on page 7-4

- *Parameters available for each comparison type* on page 7-5
- *Entering addresses, address ranges, and data address comparisons* on page 7-6
- *Entering data value comparisons* on page 7-7
- *Chaining tracepoints* on page 7-28
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — *Using variables in the debugger* on page 1-28 for details of the `@entry` symbol
  - — *Alphabetical command reference* on page 2-12.

7.14 Chaining tracepoints

If supported by your target, you can create complex conditional tracepoints by chaining individual tracepoints together. You can create chained tracepoints in the following ways:

- Using the following dialog boxes:
  - Trace on X after Y [and/or Z] dialog box
  - Trace on X after Y executed N times dialog box
  - Trace on X after A==B dialog box
  - Trace if A==B in X dialog box.
- Using trace command qualifiers.

See also:

- *Chaining tracepoints with command qualifiers*.

7.14.1 Chaining tracepoints with command qualifiers

You use the `hw_and` command qualifier to create chained tracepoints. For the first tracepoint in the chain, include the `hw_and:next` or `hw_and:"then-next"` command qualifier. For each subsequent tracepoint in the chain, include the `hw_and:prev` or `hw_and:"then-prev"` command qualifier.

Note

It is not recommended that you use the CLI to create chained tracepoints. If you want to use chained tracepoints in scripts, then create them using the tracepoint dialog boxes described in this section, and copy the commands generated by RealView Debugger.

See also

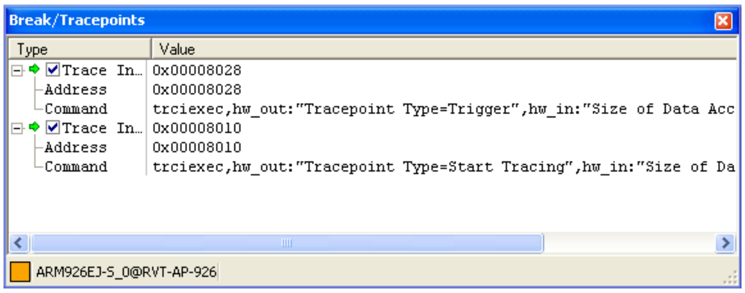
- *Setting a Trace on X after Y [and/or Z] tracepoint chain* on page 7-20
- *Setting a Trace on X after Y executed N times tracepoint chain* on page 7-22
- *Setting a Trace on X after A==B tracepoint chain* on page 7-24
- *Setting a Trace if A==B in X tracepoint chain* on page 7-26
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Using variables in the debugger* on page 1-28 for details of the `@entry` symbol
  - *Alphabetical command reference* on page 2-12.

Chapter 8
Managing Tracepoints

This chapter describes how to manage tracepoints in RealView® Debugger. It includes:

8.1 About managing tracepoints

Whenever you set a tracepoint of any type, it is displayed in the Break/Tracepoints view. Figure 8-1 shows an example:



```
Trace In…  0x00008028
  Address  0x00008028
  Command  trciexec,hw_out:"Tracepoint Type=Trigger",hw_in:"Size of Data Acc
Trace In…  0x00008010
  Address  0x00008010
  Command  trciexec,hw_out:"Tracepoint Type=Start Tracing",hw_in:"Size of Da
```

Figure 8-1 The Break/Tracepoints view

——— **Note** ———

A conditional tracepoint might be displayed as two or more chained tracepoints in the Break/Tracepoints view.

———————————

To see the address and command details for a specific tracepoint, expand the display by clicking ⊞.

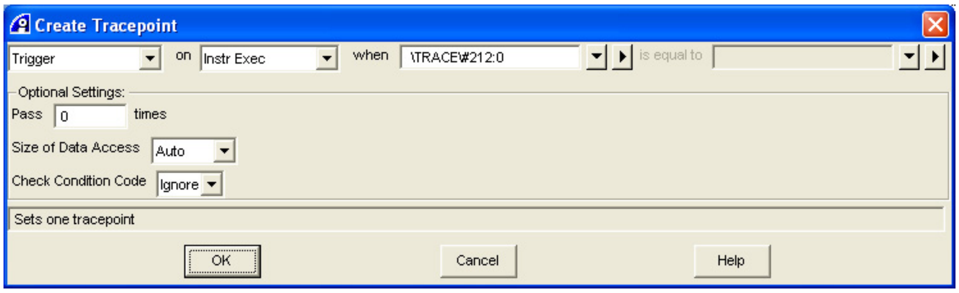
See also:

- Chapter 6 *Setting Unconditional Tracepoints*
- Chapter 7 *Setting Conditional Tracepoints*.

8.2 Editing tracepoints

You can edit an existing tracepoint to change one or more parameters, or to attach conditions to an unconditional tracepoint. To edit a tracepoint:

1. In the Break/Tracepoints view, right-click on the required tracepoint.
2. Select **Edit...** from the context menu to display the Create Tracepoint dialog box. Figure 8-2 shows an example:



Create Tracepoint

Trigger on Instr Exec when \TRACE#212:0 is equal to

Optional Settings:

Pass 0 times

Size of Data Access Auto

Check Condition Code Ignore

Sets one tracepoint

Figure 8-2 Create Tracepoint dialog box

3. Edit the tracepoint as required.
4. Click **OK** to confirm any changes and close the Create Tracepoint dialog box.

When you edit a tracepoint, the tracepoint command includes the `modify` command qualifier.

——— **Note** ———

You cannot change the tracepoint type for tracepoints that succeed other tracepoints in a chain. For these tracepoints, the tracepoint type drop-down list is disabled, and a message is displayed in the Information box at the bottom of the Create Tracepoint dialog box informing you that this tracepoint is changed. All other fields can be edited as for independent tracepoints.

———————————

See also:

- Chapter 7 *Setting Conditional Tracepoints*
- the following in the *RealView Debugger Command Line Reference Guide*:
  — *Alphabetical command reference* on page 2-12.

8.3 Disabling tracepoints

To disable a tracepoint, deselect the check box for that tracepoint. When you disable a tracepoint, the arrow icons in the left margin that indicate tracepoints become fainter, to indicate that the tracepoint is disabled:

- ➡ indicates that a trigger has been disabled
- ⬇ indicates that a trace start point or start of trace range has been disabled
- ⬆ indicates that a trace end point or end of trace range has been disabled.

To re-enable the tracepoint, select the check box for that tracepoint.

When you disable a tracepoint that is chained with others to form a conditional tracepoint, those tracepoints that succeed it in the chain are also disabled. When you enable a Tracepoint that is linked with others to form a conditional tracepoint, the tracepoints that precede it in the chain are also enabled. In both cases, a warning is displayed in the **Cmd** tab of the Output view.

8.4 Deleting tracepoints

To delete a tracepoint:

1. Right-click on the tracepoint entry in the Break/Tracepoints view.
2. Select **Delete** from the context menu.

If you delete a tracepoint that is chained with others to form a conditional tracepoint, those tracepoints that succeed it in the chain are also deleted, and a warning is displayed in the **Cmd** tab of the Output view.

You can also delete tracepoints using the following methods:

- Double-click on the tracepoint icon in the source or disassembly view.

  ——— **Note** ———

  For targets with *Embedded Trace Macrocell*™ (ETM™), do not use this method if you have set multiple tracepoints at line of code or disassembly.

  ————————

- For a *RealView ARMulator® ISS* (RVISS) target:
  1. Right-click on the tracepoint icon in the margin to display the context menu.
  2. Select **Remove Tracepoint** from the context menu.

  Only one tracepoint can be set at any line or address.

- For an ETM-based target:
  1. Right-click on the tracepoint icon in the margin to display the context menu.
  2. Select **Insert Tracepoint...** from the context menu. The New Tracepoint dialog box is displayed. Any tracepoint types that are currently set at the line or address have **Clear** options available at the top of list.

  Multiple tracepoints can be set at any line or address.

  ——— **Note** ———

  Where multiple tracepoints of the same type are listed, you cannot distinguish between them.

  ————————

8.5 Creating tracepoint favorites

You can save commonly used tracepoints in your personal Favorites List. However, you must be familiar with the CLI command syntax for the tracepoint commands.

Before you create a tracepoint favorite, familiarize yourself with the CLI command syntax for the tracepoint commands.

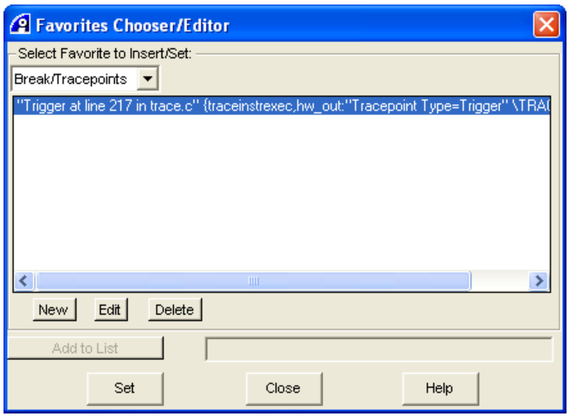
To create a new breakpoint favorite:

1. Connect to the target.

   Note

   You must connect to a target before you can use or modify your Favorites List.

2. Load your image.
3. Select **Debug** → **Breakpoints** → **Set Break/Tracepoint from List** → **Break/Tracepoint Favorites...** from the Code window main menu to display the Favorites/Chooser Editor dialog box. Figure 8-3 shows an example. Any previous tracepoints that have been added to the list are shown.



"Trigger at line 217 in trace.c" {traceinstrexec,hw_out:"Tracepoint Type=Trigger" \TRA

Figure 8-3 Favorites/Chooser Editor dialog box

4. Click **New** to display the New/Edit Favorite dialog box. Figure 8-4 shows an example:

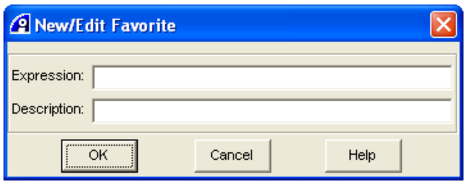


Figure 8-4 New/Edit Favorite dialog box

5. Enter the CLI command for the tracepoint in the Expression field.
6. Optionally, enter a short text description to identify the tracepoint for future use.
7. Click **OK** to confirm the entries and close the New/Edit Favorite dialog box.

   The Favorites Chooser/Editor dialog box is displayed with the newly-created tracepoint shown in the display list.

8. Either:
   - Click **Set** if you want to set the tracepoint.

- Click **Cancel** if you do not want to set the tracepoint at this time.

The Favorites/Chooser Editor dialog box closes.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Connecting to a target* on page 3-27
  - *Loading an executable image* on page 4-4.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

8.6 Finding source code corresponding to a tracepoint

If you have many tracepoints set, you can locate the line of source or disassembly where a specific tracepoint is set.

To locate the line of source code corresponding to a tracepoint:

1. Select **Break/Tracepoints** from the Code window **View** menu to display the Break/Tracepoints view.
2. Right-click on the required tracepoint entry in the Break/Tracepoints view to display the context menu.
3. Select **View Code** from the context menu.

   A blue arrow ⇨ is placed next to the line in the source or disassembly view to which the tracepoint corresponds.

Chapter 9
Analyzing Trace with the Analysis Window

This chapter describes the ways you can analyze captured trace information in RealView® Debugger with the Analysis window. It includes:

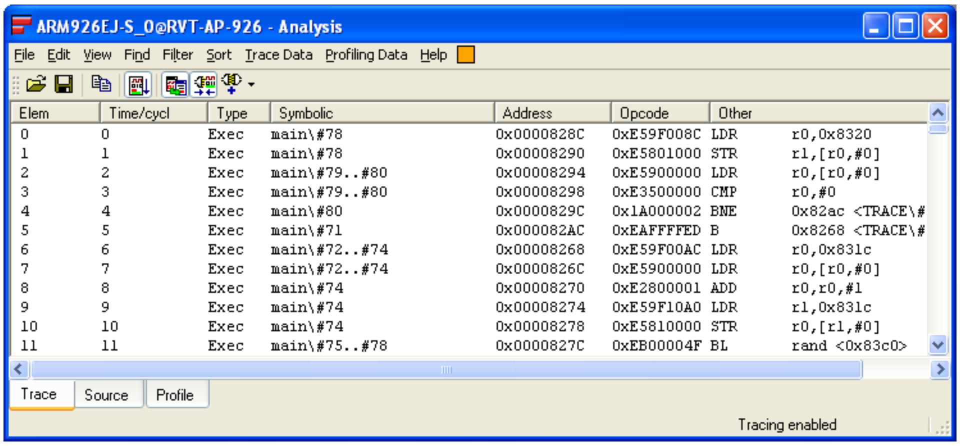
9.1 About analyzing trace with the Analysis window

The Analysis window enables you to analyze trace captured from:

- a development platform that supports an *Embedded Trace Macrocell*™ (ETM™) and, optionally, an *Embedded Trace Buffer*™ (ETB™)
- a *RealView ARMulator® ISS* (RVISS) model.

You can also perform basic profiling analysis.

An example Analysis window is shown in Figure 9-1.


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile

Elem	Time/cycl	Type	Symbolic	Address	Opcode	Other
0	0	Exec	main\#78	0x0000828C	0xE59F008C LDR	r0,0x8320
1	1	Exec	main\#78	0x00008290	0xE5801000 STR	r1,[r0,#0]
2	2	Exec	main\#79..#80	0x00008294	0xE5900000 LDR	r0,[r0,#0]
3	3	Exec	main\#79..#80	0x00008298	0xE3500000 CMP	r0,#0
4	4	Exec	main\#80	0x0000829C	0x1A000002 BNE	0x82ac <TRACE\#
5	5	Exec	main\#71	0x000082AC	0xEAFFFFED B	0x8268 <TRACE\#
6	6	Exec	main\#72..#74	0x00008268	0xE59F00AC LDR	r0,0x831c
7	7	Exec	main\#72..#74	0x0000826C	0xE5900000 LDR	r0,[r0,#0]
8	8	Exec	main\#74	0x00008270	0xE2800001 ADD	r0,r0,#1
9	9	Exec	main\#74	0x00008274	0xE59F10A0 LDR	r1,0x831c
10	10	Exec	main\#74	0x00008278	0xE5810000 STR	r0,[r1,#0]
11	11	Exec	main\#75..#78	0x0000827C	0xEB00004F BL	rand <0x83c0>

Tracing enabled

Figure 9-1 Example of the Analysis window

To view the Analysis window, select **Analysis Window** from the Code window **View** menu. The **Trace** tab view is displayed by default.

The following sections give an overview of the Analysis window features:

- *Understanding the current state of the trace*
- *Status messages in the captured trace* on page 9-3
- *Profiling in RealView Debugger* on page 9-3.

9.1.1 Understanding the current state of the trace

The status bar at the bottom of the Analysis window (see Figure 9-1) displays the current state of the trace. This can be one of:

Tracing enabled	The **Tracing Enabled** option in the **Edit** menu is selected. This message only occurs with trace targets that support the **Tracing Enabled** option.
Tracing disabled	The **Tracing Enabled** option in the **Edit** menu is not selected. This message only occurs with trace targets that support the **Tracing Enabled** option.
Not connected	There is no trace support for the target, or the target is not connected.
Ready	Trace functionality is available. This message only occurs with targets that do not support the **Tracing Enabled** option in the **Edit** menu.
Searching	A search of the trace buffer is in progress.

Generating profile	A trace profile is being generated. Displayed when the **Profile** tab is selected for the first time after either the initial trace capture, or an update of the trace data.
Updating	The Analysis window is being updated with data from the trace target.

9.1.2 Status messages in the captured trace

The **Trace** tab of the Analysis window shows the decompressed trace output. Some rows of the returned trace output are status messages that show information about the processor cycle. These status messages are displayed in red.

See also

- Appendix C *Status Messages in Captured Trace* for a complete list of the status messages.

9.1.3 Profiling in RealView Debugger

Profiling in RealView Debugger is limited to analyzing the time spent in each function, and the number of times each function is executed. For more advanced profiling, you are recommended to use the ARM Profiler.

The ARM Profiler software and license is provided with *RealView Development Suite* (RVDS) Professional edition. If you do not have RVDS Professional edition, you can purchase the ARM Profiler software and license separately.

See also

- *ARM Profiler User Guide*.

9.2 Changing the displayed trace details

The following sections describe how to display various trace information in the trace view:

- *Changing the columns displayed in the Trace view*
- *Displaying all trace entries* on page 9-6
- *Displaying instructions for trace entries* on page 9-6
- *Displaying data for trace entries* on page 9-7
- *Displaying instruction boundaries* on page 9-7
- *Displaying function boundaries* on page 9-7
- *Interleaving source with the trace output* on page 9-7
- *Displaying inferred registers* on page 9-8
- *Displaying the captured trace for each processor in a multiprocessor system* on page 9-10

9.2.1 Changing the columns displayed in the Trace view

You can choose which columns to display in the Trace view. To do this, select one of the following options from the **Trace Data** menu in the Analysis window:

Position Displays the `Elem` column that shows the position of each element within the trace buffer, where the value can represent either:

- An index within the trace buffer.
- A cycle number, if your *Trace Port Analyzer* (TPA) supports cycle-accurate tracing. To display cycle numbers in this column you must select **Cycle accurate tracing** in the Configure ETM dialog box before you begin tracing.

This column is displayed by default.

Absolute Time

Displays the `Time/`*`unit`* column that shows the timestamp value. To change the format of the values, select **Scale Time Units...** from the **View** menu.

This column shows information only when either **Enable Timestamps** or **Cycle accurate tracing** is selected in the Configure ETM dialog box. To scale between times and cycle numbers, select **Define Processor Speed for Scaling...** from the **View** menu.

This column is displayed by default.

Relative Time

Displays the `+Time` column that shows the delta timestamp value. This indicates the time taken between the previous instruction and the current instruction. To change the format of the values, select **Scale Time Units...** from the **View** menu.

The `+Time` column shows information only when either **Enable Timestamps** or **Cycle accurate tracing** is selected in the Configure ETM dialog box.

This column is not displayed by default.

Access Type Displays the `Type` column that shows the access type of the current element, which can be any of:

Code	Code access (fetch).
Data	Data access (read or write).
Exec	Instruction was executed.
Folded	Folded branch (folded by branch prediction unit and never issued).

FoldNoEx

Folded branch (folded by branch prediction unit and never issued) was not executed.

Instr Instructions on targets where the trace does not contain executed flags, such as XScale™ on-chip trace and RVISS targets.

——— **Note** ———

The instruction might or might not have been executed.

———

NoExec An instruction has not been executed.

——— **Note** ———

If an access type is prefixed by `R`, this indicates a read access. A `W` prefix indicates a write access.

———

This column is displayed by default.

Address as Symbol/line

Displays the `Symbolic` column that shows the symbolic position information for the current element, and takes one of the following forms:

- *`source_symbol_name+offset`*

  For example, `Arr_2_Glob+0x65` might be a data access to the variable address `Arr_2_Glob`, with an offset of `0x65`.

- *`source_symbol_name`*`[\[~]#`*`line_number`*`[..#`*`line_number`*`]]`

  source_symbol_name

  This can be any symbolic information, including a function, module, variable, or low-level symbol

  ~ This means that the corresponding instruction, and the instructions corresponding to the symbols with the same line number, implement the same line of code. For example, instructions that implement a branch or loop.

  #line_number[..#line_number]

  This means that the corresponding instructions span the specified lines of source code.

  For example, your trace output might have the following sequence of symbols:

  ```
  Func_2\#159
  Func_2\~#159
  Func_2\#161..#164
  ```

  The instructions corresponding to the symbols `Func_2\#159` and `Func_2\~#159` both implement the source code at line 159 in the function `Func_2()`.

  The instruction corresponding to the symbol `Func_2\#161..#164` spans lines 161 to 164 of the source in the function `Func_2()`.

This column is displayed by default.

Address as Value

Displays the `Address` column that shows the address of the instruction or data accessed.

This column is displayed by default.

Data Value in Hex

Displays the `Data/Hex` column that shows the data values in hexadecimal form.

This column is not displayed by default.

Data Value in Decimal

Displays the `Data/Dec` column that shows the data values in decimal form.

This column is not displayed by default.

Opcode Displays the `Opcode` column that shows the opcode of the instruction accessed.

This column is displayed by default.

Interpretation of Data/Opcode

Displays the `Other` column that shows the disassembly for the line of trace:

- For instructions, the disassembled instruction is displayed.
- For data, the display has the following format:

  `<Data> '`*`character`*`' | `*`hexvalue`*` ['`*`character`*`' | `*`hexvalue`*` ...]`

  character is a printable character or a C-style escape code.

  hexvalue is the hexadecimal value of the character if it is neither a printable character, nor an escape code.

  For example:

  `<Data> '\f' 0x03 '\0' '\0'`

This column is displayed by default.

Count of Hits

Displays the `Count` column that shows the number of times a particular address was accessed.

This column is not displayed by default.

See also

- *Enable Timestamping* on page 4-10
- *Cycle accurate tracing* on page 4-11.

9.2.2 Displaying all trace entries

To display all captured trace entries, select **All Trace** from the **Trace Data** menu in the Analysis window. This is the default.

See also

- *Changing the displayed trace details* on page 9-4.

9.2.3 Displaying instructions for trace entries

To display only the instructions in the captured trace:

1. Deselect **All Trace** from the **Trace Data** menu in the Analysis window.
2. Select **Instructions** from the **Trace Data** menu.

See also

- *Changing the displayed trace details* on page 9-4.

9.2.4 Displaying data for trace entries

To display only the data in the captured trace:

1. Deselect **All Trace** from the **Trace Data** menu in the Analysis window.
2. Select **Data** from the **Trace Data** menu.

See also

- *Changing the displayed trace details* on page 9-4.

9.2.5 Displaying instruction boundaries

To display instruction boundaries in the captured trace, select **Instruction Boundaries** from the **Trace Data** menu in the Analysis window.

The instruction boundaries are shown as ruled lines. This is useful if you want to see loaded and stored data associated with the instruction that loaded or stored the data. This also shows the traced instructions, even if the **Instructions** option is not selected.

See also

- *Changing the displayed trace details* on page 9-4.

9.2.6 Displaying function boundaries

To display function boundaries in the captured trace, select **Function Boundaries** from the **Trace Data** menu in the Analysis window.

Note

Function boundaries show where a function was called. Where the compiler inlines function calls, function boundaries are not shown.

What is displayed depends on other options you have selected. Some examples are as follows:

- If you display function boundaries with no instructions turned on, then the view shows the function callgraph.
- If you display **Inferred Registers** and **Function Boundaries** but no other instruction records, then this view shows values passed into and returned from functions. Therefore, you can track function behavior. When you are running AAPCS-compliant code on an ARM processor registers `r0-r3` contain function parameters, and `r0` contains a function's return value.

See also

- *Changing the displayed trace details* on page 9-4.

9.2.7 Interleaving source with the trace output

To display interleaved source in the captured trace, select **Interleave Source** from the **Trace Data** menu in the Analysis window.

Displays the source lines associated with each trace buffer entry interleaved with the buffer entry. The corresponding code is displayed after each buffer entry. Where several successive buffer entries are associated with the same or overlapping sets of source lines, the source line is shown after the first buffer entry and then suppressed. Source code that is executed repeatedly is shown for each execution.

See also

- *Changing the displayed trace details* on page 9-4.

9.2.8 Displaying inferred registers

You can display inferred register values interleaved with the trace data.

Note

The Inferred Registers view is only available if you are using an ARM architecture-based processor. In addition, traced instructions must be shown in the **Trace** tab. That is, when any of the **Instructions** and **Instruction boundaries** options is selected.

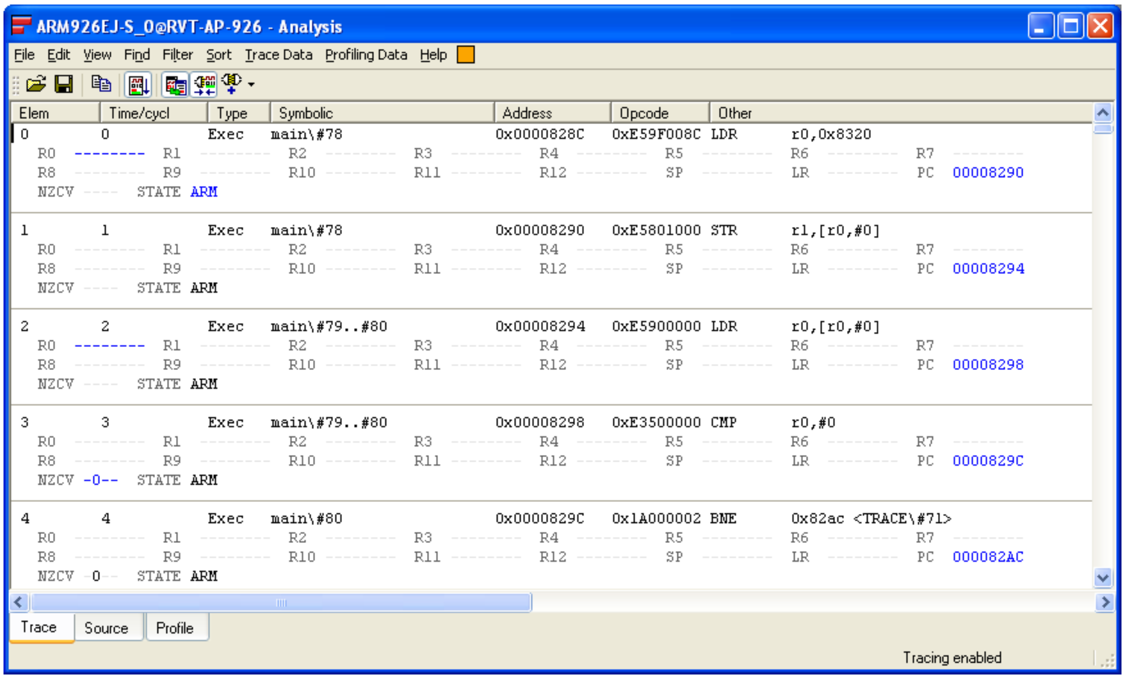
To display inferred register values in the captured trace:

1. Select **Inferred Registers** from the **Trace Data** menu in the Analysis window.

   The inferred registers are displayed, together with instruction boundaries. Figure 9-2 shows an example.

   Note

   Instruction boundaries are displayed even if the **Instruction boundaries** option is not selected.


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem     Time/cycl   Type   Symbolic               Address      Opcode         Other
0        0           Exec   main\#78               0x0000828C   0xE59F008C LDR r0,0x8320
  R0  --------  R1  --------  R2  --------  R3  --------  R4  --------  R5  --------  R6  --------  R7  --------
  R8  --------  R9  --------  R10 --------  R11 --------  R12 --------  SP  --------  LR  --------  PC  00008290
  NZCV ----  STATE ARM

1        1           Exec   main\#78               0x00008290   0xE5801000 STR r1,[r0,#0]
  R0  --------  R1  --------  R2  --------  R3  --------  R4  --------  R5  --------  R6  --------  R7  --------
  R8  --------  R9  --------  R10 --------  R11 --------  R12 --------  SP  --------  LR  --------  PC  00008294
  NZCV ----  STATE ARM

2        2           Exec   main\#79..#80          0x00008294   0xE5900000 LDR r0,[r0,#0]
  R0  --------  R1  --------  R2  --------  R3  --------  R4  --------  R5  --------  R6  --------  R7  --------
  R8  --------  R9  --------  R10 --------  R11 --------  R12 --------  SP  --------  LR  --------  PC  00008298
  NZCV ----  STATE ARM

3        3           Exec   main\#79..#80          0x00008298   0xE3500000 CMP r0,#0
  R0  --------  R1  --------  R2  --------  R3  --------  R4  --------  R5  --------  R6  --------  R7  --------
  R8  --------  R9  --------  R10 --------  R11 --------  R12 --------  SP  --------  LR  --------  PC  0000829C
  NZCV -0--  STATE ARM

4        4           Exec   main\#80               0x0000829C   0x1A000002 BNE 0x82ac <TRACE\#71>
  R0  --------  R1  --------  R2  --------  R3  --------  R4  --------  R5  --------  R6  --------  R7  --------
  R8  --------  R9  --------  R10 --------  R11 --------  R12 --------  SP  --------  LR  --------  PC  000082AC
  NZCV -0--  STATE ARM
```

Figure 9-2 Example of inferred registers

2. Optionally, select one or more of the following options from the from the **Trace Data** menu:

   Only Known Registers
   To view only those registers whose values are known.

   Only Changing Registers
   To view only those registers where the values have changed between instructions.

   Narrow Register View
   To view six registers on a line, instead of eight.

   These options are only available when **Inferred Registers** is selected.

Considerations when displaying inferred registers

Be aware of the following when displaying inferred registers:

- Inferred register values are inferred from the traced instructions, and are not read from the registers. This means that not all register values are always known.

  ——— **Note** ———
  Because the register values are inferred from the captured trace data, the inferred register values can be calculated and displayed without stopping the processor.

- The register data is shown following the instruction opcode, and shows the values in the registers after the instruction has executed. Register data is displayed in the following way:
  - Register names are gray.
  - Unchanged register values are black.
  - Changed register values are blue.
  - Unknown register values are represented by --------. If a register value became unknown in the current cycle, this appears in blue to indicate a changed register value.
- The amount of information that can be inferred depends on the amount of information contained in the trace. If you have not traced data, few register values can be inferred. With full data trace, most register values can be inferred.
- When any of the **Instructions** and **Instruction Boundaries** options is selected, the inferred register values show the values contained in the registers after the instruction was executed.
- When no instruction rows are displayed but **Function Boundaries** are displayed, the inferred register values shown are the values contained in the registers before the instruction at the function boundary was executed.

See also

- *Changing the displayed trace details* on page 9-4.

9.2.9 Displaying the captured trace for each processor in a multiprocessor system

When tracing multiple processors, you can cycle through the available active connections to display the captured trace for each processor.

Click the **Cycle Connections** button to cycle through the connections. Each connection you cycle to becomes the current connection.

Click the drop-down arrow to display the **Connection** menu. This menu lists the active connections with the current connection marked with an asterisk. Figure 9-3 shows an example:

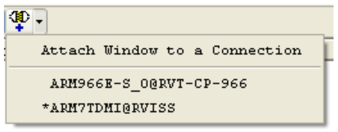


Figure 9-3 Connection menu

The menu also provides an option to attach the Code window to the current connection. When a Code window is attached to a connection a check mark is added to the **Attach Window to Connection** option and the connection that is attached to the Code window. Although you can still cycle through multiple connections, the connection details do not change in Code windows that are attached.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Attaching a Code window to a connection* on page 7-10.

9.3 Viewing the captured trace

The following sections describe the features available that enable you to view the trace information in different ways:

- *About the Trace tab*
- *Tracking trace buffer lines in the Code window* on page 9-12
- *Calculating the time between two points* on page 9-12
- *Showing the current position relative to the trigger point* on page 9-13
- *Changing the default format of time information* on page 9-13
- *Specifying the processor speed for cycle/time computations* on page 9-14
- *Automatically updating the trace information in the Analysis window* on page 9-14
- *Updating the captured trace information* on page 9-15
- *Copying trace to the clipboard* on page 9-15
- *Clearing the trace buffer* on page 9-15.

See also:

- *Enable Timestamping* on page 4-10
- *Cycle accurate tracing* on page 4-11
- *Changing the displayed trace details* on page 9-4
- *Status messages in the captured trace* on page 9-3
- the following in the *RealView Debugger User Guide*:
  - Chapter 10 *Changing the Execution Context*.

9.3.1 About the Trace tab

To view the trace data, click on the **Trace** tab. This tab enables you to view:

- Captured trace data, and shows:
  - the symbolic representation of instructions that have been traced
  - the symbolic representation of data that has been traced, interleaved between the traced instructions, if you have specified data capture in the trace configuration
  - the disassembly of traced instructions and data, with branches to functions showing the symbolic name of the function.

  The display of this information is controlled using the options in the Columns section of the **Trace Data** menu.

- Additional information that helps you to interpret the captured trace:
  - interleaved source
  - inferred registers
  - function boundaries
  - instruction boundaries.

  The display of this information is controlled using the options in the Rows section of the **Trace Data** menu.

  Note

  You can also hide and show the traced instructions and data as required.

The **Trace** tab also shows error and warning messages where appropriate.

See also

- *Viewing the captured trace*.

9.3.2 Tracking trace buffer lines in the Code window

Tracking of trace buffer lines in the Code window is enabled by default. RealView Debugger locates the line in the source or disassembly view that corresponds to the currently selected line in the Analysis window.

To toggle the tracking of trace buffer lines, either:

- click the **Show in Code** button
- select **Code Window Tracking** from the **View** menu.

Select this option to track addresses in the code window. RealView Debugger locates the line in the source or disassembly view that corresponds to the currently selected line in the Analysis window. The results of tracking depend on the currently selected view in the Code window:

Source file view

When you select a row of output representing an instruction in the Analysis window, RealView Debugger inserts a marker next to the corresponding source line.

——— **Note** ———

If the instruction you are selecting is at an address that does not correspond to one of your source files, no tracking occurs.

———————————

Disassembly view

When you select a row of output representing an instruction in the Analysis window, RealView Debugger inserts a marker next to the corresponding disassembly line in the **Disassembly** tab.

You can also track addresses in this manner by clicking on the desired row in the Analysis window.

——— **Note** ———

Be aware that:

- no tracking occurs if you select a row that does not represent an instruction, or does not have a data address value
- tracking changes the scope.

———————————

See also

- *Viewing the captured trace* on page 9-11.

9.3.3 Calculating the time between two points

You can calculate the time between two points:

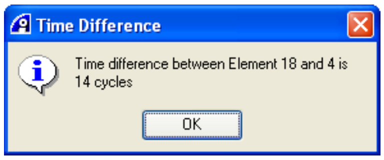
- between a selected line and the current line
- between the trigger point and the current line.

Time between a selected line and the current line

To calculate the time from a selected line to the current line:

1. Select the line of trace representing the starting point from which you want to measure.

2. Right-click on the row representing the finishing point for the measurement.
3. Select **Time Measure from Selected...** from the context menu to display the Prompt dialog box. Figure 9-4 shows an example:



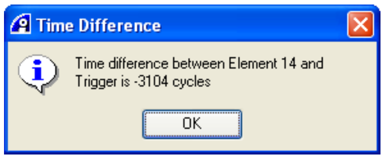
Time difference between Element 18 and 4 is 14 cycles

Figure 9-4 Time measurement from selected line dialog box

Time between the trigger point and the current line

You can display the time or number of cycles from the trigger to the selected line, depending on the Scale Time Units setting. To do this:

1. If required, change the value of Scale Time Units:
   - for targets with an ETM, the default is a time unit (Nanoseconds)
   - for simulators, the default is cycles.
2. Right-click on the row representing the finishing point for the measurement to display the context menu.
3. Select **Time Measure from Trigger** from the context menu to display the Prompt dialog box. Figure 9-5 shows an example:



Time difference between Element 14 and Trigger is -3104 cycles

Figure 9-5 Time measurement from trigger dialog box

See also

- *Viewing the captured trace* on page 9-11.

9.3.4 Showing the current position relative to the trigger point

To display the current position relative to the trigger point, select **Show Position Relative to Trigger** from the **View** menu.

See also

- *Viewing the captured trace* on page 9-11.

9.3.5 Changing the default format of time information

To change the default format in which time information is displayed in the Analysis window:

1. Select **Scale Time Units...** from the **View** menu to display the List Selection dialog box.
2. Select one of the following time formats:
   - **Default**
   - **Picoseconds**
   - **Nanoseconds**

- **Microseconds**
- **Milliseconds**
- **Seconds**
- **Cycles**.

3. Click **OK**.

Considerations when changing the default time format

The default format depends on both your TPA and your target processor. ETM-enabled processors have a default time format of nanoseconds. ETMv3 targets default to:

- **Cycles** if cycle accurate tracing is enabled
- **Picoseconds** if timestamping is enabled.

If cycle accurate tracing and timestamping are both disabled for ETMv3 targets, then setting the default format to **Default** provides instruction count information.

You must define the processor speed for scaling in the following cases:

- When the default format of the target is cycles and you select any other format.
- When the default format of the target is not cycles and you select **Cycles** as the display format.

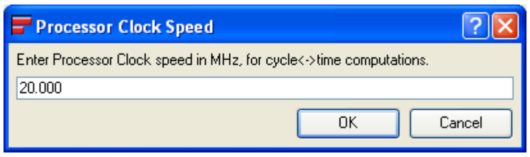
See also

- *Viewing the captured trace* on page 9-11.

9.3.6 Specifying the processor speed for cycle/time computations

To specify the clock speed of the target processor:

1. Select **Define Processor Speed for Scaling...** from the **View** menu to display the Prompt dialog box. Figure 9-6 shows an example:


Processor Clock Speed

Enter Processor Clock speed in MHz, for cycle<->time computations.

20.000

Figure 9-6 Defining processor speed

2. Enter a clock speed in MHz.
3. Click **Speed**.

This sets the scaling between cycles and timestamp values in the current buffer, as shown in the `Time/`*`unit`* column.

See also

- *Viewing the captured trace* on page 9-11.

9.3.7 Automatically updating the trace information in the Analysis window

To automatically update the Analysis window with new information when a new buffer load of trace data is returned, select **Automatic Update on New Buffer** from the **View** menu.

See also

- *Viewing the captured trace* on page 9-11.

9.3.8 Updating the captured trace information

To refresh the captured trace information, select **Update** from the **View** menu.

This is useful for instance if you have loaded the wrong symbol information for the trace data that you have collected from your target.

See also

- *Viewing the captured trace* on page 9-11.

9.3.9 Copying trace to the clipboard

To copy all or part of the trace output to the clipboard:

1. Selected the part of the trace output that you want to copy.
2. Click **Copy** to copy the selected trace.

See also

- *Viewing the captured trace* on page 9-11.

9.3.10 Clearing the trace buffer

To clear the information in the trace buffer, select **Clear Trace Buffer** from the **View** menu.

See also

- *Viewing the captured trace* on page 9-11.

9.4 Finding information in captured trace

The following sections describe how to locate a specific position in the captured trace output:

- *Finding the location of the trigger point*
- *Finding a specific element number in the trace output* on page 9-17
- *Finding a specific timestamp value* on page 9-18
- *Finding a specific address* on page 9-19
- *Finding a specific data value* on page 9-20
- *Finding a specific symbol name* on page 9-21
- *Finding the next instance of the previously searched item* on page 9-21
- *Finding the previous instance of the previously searched item* on page 9-21.

Each search is performed in a downwards direction starting at the current cursor position.

See also:

- *Setting the size of the trace buffer (RVISS targets only)* on page 3-9
- *Changing the columns displayed in the Trace view* on page 9-4
- *Changing the default format of time information* on page 9-13
- Chapter 6 *Setting Unconditional Tracepoints*.

9.4.1 Finding the location of the trigger point

To locate the row of trace output representing the trigger point within your code, select **Find Trigger** from the **Find** menu.

If you set a trigger point and it is not found, then:

- The trigger point might not have been hit, which might be because:
  - — The code has not been reached (for example the trigger is in an exception handler).
  - — The trigger point has been placed outside a region where trace is enabled (for example, before a trace start point is hit) or outside a trace range if not using start points. Make sure that the trigger point is within the region being traced.
- For RVISS targets, the trace buffer size is too small. In this case:
  1. Increase the size of the trace buffer.
  2. Select **Reload Image to Target** from the Code window **Target** main menu to reload the image without clearing your tracepoints.
  3. Run the application again to recapture the trace information.

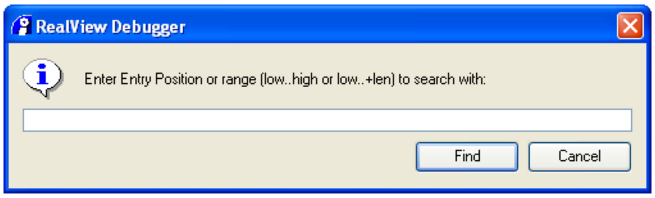
See also

- *Finding information in captured trace*.

9.4.2 Finding a specific element number in the trace output

To locate a specific element number in the `Elem` column:

1. Select **Find Position...** from the **Find** menu to display the Prompt dialog box. Figure 9-7 shows an example:


RealView Debugger

Enter Entry Position or range (low..high or low..+len) to search with:

Figure 9-7 Finding a position match

2. Enter one of the following types of information:

   Entry position

   Specify an element number, in decimal or hexadecimal format. Only an exact match is returned.

   Entry position range

   Specify a range of element numbers, where RealView Debugger displays the first found value within that range. The range you specify can be either:

   - `low..high`, such as `1..10`, where RealView Debugger locates the first occurrence of any Elem value within the range 1 to 10.
   - `low..+len`, such as `40..+10`, where RealView Debugger locates the first occurrence of any Elem value within the range 40 to 50. The `len` of 10 represents the offset value. You can also enter a negative value range, such as `-10..10`.

3. Click **Find**.

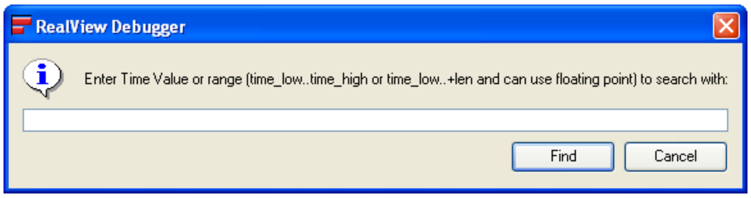
See also

- *Finding information in captured trace* on page 9-16.

9.4.3 Finding a specific timestamp value

To locate a specific timestamp value in the `Time/unit` column (or, in the case of the **Profile** tab, the `Exec%` column):

1. Select **Find Timestamp...** from the **Find** menu to display the Prompt dialog box. Figure 9-8 shows an example:



Enter Time Value or range (time_low..time_high or time_low..+len and can use floating point) to search with:

Figure 9-8 Finding a time match

2. Enter one of the following types of information:

   Time value

   Specify a timestamp value, in the format currently used for the `Time/unit` column. Only an exact match is returned.

   Time range

   Specify a range of timestamp values, in the format currently used for the `Time/unit` column. In this case, RealView Debugger displays the first found value within that range. The range you specify can be either:

   - *`time_low..time_high`*, such as `-100..-50`, where RealView Debugger locates the first occurrence of a timestamp value within the range -100 to -50.
   - *`time_low..+len`*, such as `-100..+10`, where RealView Debugger locates the first occurrence of a timestamp value within the range -100 to -90. The *`len`* of 10 represents the offset value.

   You can also use floating point values, such as `-50.5`.

   Note

   Depending on your system solution, the `Time/unit` column might contain cycle numbers instead of timestamp values. In this case, you can search using cycle numbers.

3. Click **Find**.

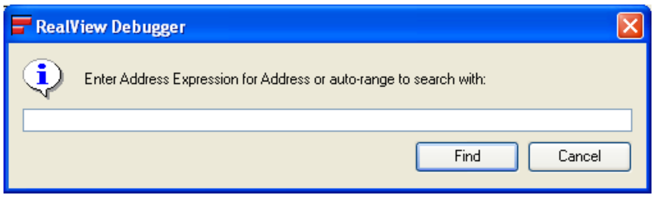
See also

- *Finding information in captured trace* on page 9-16.

9.4.4 Finding a specific address

To locate a specific address in the `Address` column that corresponds to an expression you enter:

1. Select **Find Address Expression...** from the **Find** menu to display the Prompt dialog box. Figure 9-9 shows an example:



Enter Address Expression for Address or auto-range to search with:

Figure 9-9 Finding an address expression match

2. Enter one of the following types of information:

   Address expression

   Specify an address expression, which can be one of:
   - a function name
   - a structure name
   - an array symbol.

   Auto-range

   An auto-range of address values is generated from an expression that you enter, which can be one of:
   - A function name, where the generated address range is from the start to the end of the function.
   - A structure, where the generated address range is from the start-to-end of the structure.
   - An array symbol, where the generated address range is from the start of the variable to the end, and the end address is the *start*+*sizeof*(*var*). For example, if the start value of where the array is stored in memory is `0x8000`, and the array size is 16 bytes, the end address is considered to be `0x8010` (that is, `0x8000`+16).

   RealView Debugger displays the first found address value within the auto-range that is generated.

3. Click **Find**.

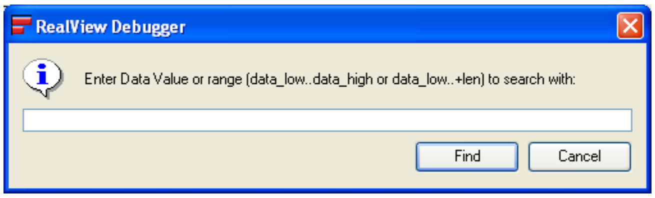
See also

- *Finding information in captured trace* on page 9-16.

9.4.5 Finding a specific data value

To locate a specific data value that is read from or written to memory, in the `Data` column:

1. Select **Find Data Value...** from the **Find** menu to display the Prompt dialog box. Figure 9-10 shows an example:



Enter Data Value or range (data_low..data_high or data_low..+len) to search with:

Figure 9-10 Finding a data value match

2. Enter one of the following types of information:

   Data value

   Specify a data value, in decimal or hexadecimal format.

   Data range

   Specify a range of data values, where RealView Debugger displays the first found value that is read from, or written to, memory, within that range. The range you specify can be either:

   - `*data_low*..*data_high*`, such as `1..10`, where RealView Debugger locates the first occurrence of any data value within the range 1 to 10 being read from, or written to, memory.
   - `*data_low*..+*len*`, such as `40..+10`, where RealView Debugger locates the first occurrence of any data value within the range 40 to 50 being read from, or written to, memory. The *len* of 10 represents the offset value.

3. Click **Find**.

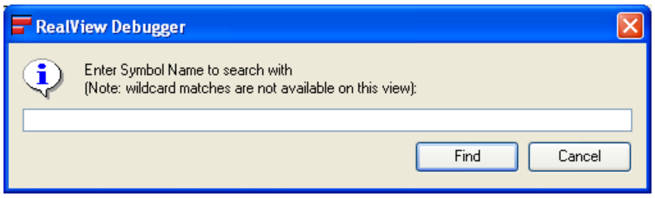
See also

- *Finding information in captured trace* on page 9-16.

9.4.6 Finding a specific symbol name

To locate a specific symbol-name string, such as a function name or variable, in the `Symbolic` column:

1. Select the **Trace** or **Profile** tab.
2. Select **Find Symbol Name...** from the **Find** menu to display the Prompt dialog box. Figure 9-11 shows an example:


RealView Debugger

Enter Symbol Name to search with
(Note: wildcard matches are not available on this view):

Figure 9-11 Finding a symbol name match

3. Enter the symbol-name to find.

   If you have selected the **Profile** tab, then the symbol name can contain the following characters:

*****	A multi-character wildcard.
?	A single-character wildcard.

   For example, to find the variable `Ptr_1_Glob`, you might use `Ptr_1_*`. Alternatively, you might use `Ptr_?_Glob`.

4. Click **Find**.

RealView Debugger displays the first found symbol name that matches your entry.

See also

- *Finding information in captured trace* on page 9-16.

9.4.7 Finding the next instance of the previously searched item

To locate the next instance in the trace output of the search item you last specified, press F3.

The trace output is searched for the next instance of the search item you have specified.

See also

- *Finding information in captured trace* on page 9-16.

9.4.8 Finding the previous instance of the previously searched item

To locate the previous instance in the trace output of the search item you last specified, press Shift+F3.

The trace output is searched from the current cursor position, for the previous instance of the search item you have specified.

See also

- *Finding information in captured trace* on page 9-16.

9.5 Filtering information in captured trace

This sections describe how to filter the results of a trace capture that has already been performed:

- *Filtering for specific element numbers*
- *Filtering for specific timestamp values* on page 9-23
- *Filtering for specific addresses* on page 9-24
- *Filtering for specific data values* on page 9-25
- *Filtering for specific symbol names* on page 9-25
- *Filtering for specific access types* on page 9-26
- *Filtering for specific percentages of execution time* on page 9-27
- *Setting the AND condition for selected filters* on page 9-27
- *Setting the OR condition for selected filters* on page 9-28
- *Inverting the sense of the selected filters* on page 9-28
- *Clearing filters* on page 9-28.

Filtering is useful if you want to refine your area of interest within the display. You can set an unlimited number of filters.

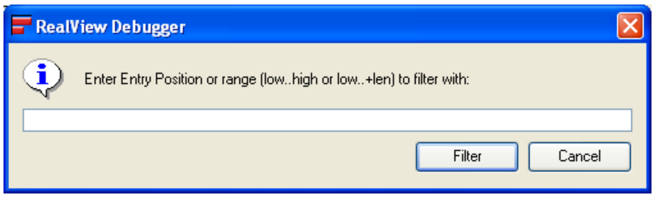
See also:

- *Changing the columns displayed in the Trace view* on page 9-4
- *Changing the default format of time information* on page 9-13.

9.5.1 Filtering for specific element numbers

To filter the information down to a specific element number, or range of numbers, in the `Elem` column:

1. Select **Filter on Position...** from the **Filter** menu to display the Prompt dialog box. Figure 9-12 shows an example:



Enter Entry Position or range (low..high or low..+len) to filter with:

Figure 9-12 Filtering on a position match

2. Enter one of the following types of information:

   Entry position

   Specify an element number, in decimal or hexadecimal format, if you want to filter the information to display only that row.

   Entry position range

   Specify a range of element numbers, where RealView Debugger filters the information down to only rows within that range. The range you specify can be either:

   - *`low..high`*, such as `1..10`, where RealView Debugger displays only those rows containing Elem values within the range 1 to 10.
   - *`low..+len`*, such as `40..+10`, where RealView Debugger displays only those rows containing Elem values within the range 40 to 50. The *`len`* of 10 represents the offset value. You can also enter a negative value range, such as `-10..10`.

3. Click **Filter**.

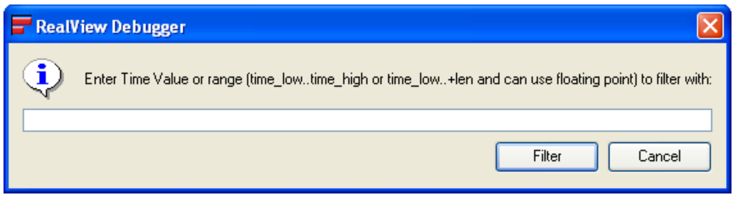
See also

- *Filtering information in captured trace* on page 9-22.

9.5.2 Filtering for specific timestamp values

To filter the information down to a specified timestamp value, or range of timestamp values, as contained in the `Time/unit` column:

1. Select **Filter on Timestamp...** from the **Filter** menu to display the Prompt dialog box. Figure 9-13 shows an example:



RealView Debugger

Enter Time Value or range (time_low..time_high or time_low..+len and can use floating point) to filter with:

Figure 9-13 Filtering on a time match

2. Enter one of the following types of information:

Time value

Specify a timestamp value, in the format currently used for the `Time/unit` column, if you want to filter the information to display only that row.

Time range

Specify a range of timestamp values, where RealView Debugger filters the information down to only rows within that range. The range you specify can be one of:

- *`time_low..time_high`*, such as `-100..-50`, where RealView Debugger displays only those rows containing timestamp values within the range -100 to -50.
- *`time_low..+len`*, such as `-100..+10`, where RealView Debugger displays only those rows containing timestamp values within the range -100 to +10. The *`len`* of 10 represents the offset value.

You can also use floating point values, such as `-90.5`.

Note

Depending on your system solution, the `Time/unit` column might contain cycle numbers instead of timestamp values. In this case, you can perform filtering using cycle numbers.

3. Click **Filter**.

See also

- *Filtering information in captured trace* on page 9-22.

9.5.3 Filtering for specific addresses

To filter the information down to a specific address, or range of addresses, in the `Address` column, that corresponds to an expression you enter:

1. Select **Filter on Address Expression...** from the **Filter** menu to display the Prompt dialog box. Figure 9-14 shows an example:

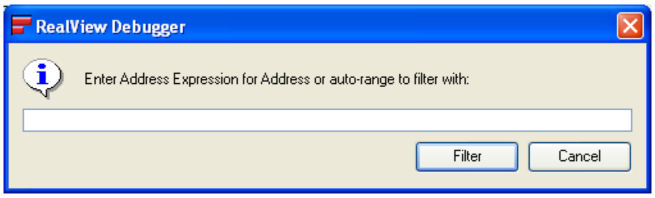

RealView Debugger
Enter Address Expression for Address or auto-range to filter with:
Filter
Cancel

Figure 9-14 Filtering on an address expression match

2. Enter one of the following types of information:

   Address expression

   Specify an address expression if you want to filter the information to display only that row. An address expression can be one of:
   - a function name
   - a structure name
   - an array symbol.

   Auto-range

   An auto-range of address values is generated from an expression that you enter, which can be any of:
   - A function name, where the generated address range is from the start-to-end of the function.
   - A structure, where the generated address range is from the start-to-end of the structure.
   - An array symbol, where the generated address range is from the start of the variable to the end, where the end is the *start*+*sizeof*(*var*). For example, if the start value of where the array is stored in memory is `0x8000`, and the array size is 16 bytes, the end address is considered to be `0x8010` (that is, `0x8000`+16).

   RealView Debugger filters the information down to only rows represented by the generated auto-range.

3. Click **Filter**.

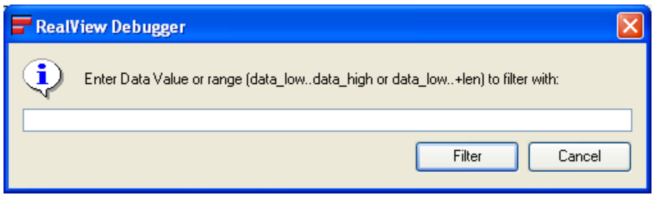
See also

- *Filtering information in captured trace* on page 9-22.

9.5.4 Filtering for specific data values

To filter the information down to a specified data value, or range of data values, read from or written to memory (in either of the `Data/Hex` or `Data/Dec` columns):

1. Select **Filter on Data Values...** from the **Filter** menu to display the Prompt dialog box. Figure 9-15 shows an example:


RealView Debugger

Enter Data Value or range (data_low..data_high or data_low..+len) to filter with:

Figure 9-15 Filtering on a data value match

2. Enter one of the following types of information:

   Data value

   Specify a data value, in decimal or hexadecimal format, if you want to filter the information to display only that row.

   Data range

   Specify a range of data values, where RealView Debugger filters the information down to only rows of data values that are read from, or written to, memory, within that range. The range you specify can be either:

   - *`data_low..data_high`*, such as `1..10`, where RealView Debugger displays only those rows containing data values that are read from, or written to, memory within the range 1 to 10.
   - *`data_low..+len`*, such as `40..+10`, where RealView Debugger displays only those rows containing data values that are read from, or written to, memory, within the range 40 to 50. The *`len`* of 10 represents the offset value.

3. Click **Filter**.

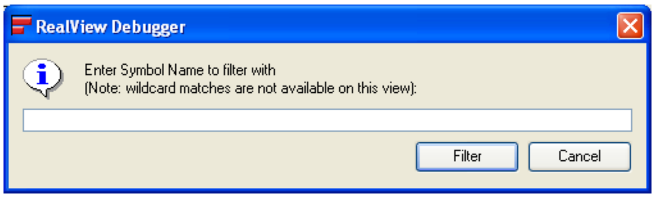
See also

- *Filtering information in captured trace* on page 9-22.

9.5.5 Filtering for specific symbol names

To filter the information down to a specified symbol-name string (such as a function name or variable), or range of symbol-name strings as contained in the `Symbol` column:

1. Select **Filter on Symbol Name...** from the **Filter** menu to display the Prompt dialog box. Figure 9-16 shows an example:


RealView Debugger

Enter Symbol Name to filter with
(Note: wildcard matches are not available on this view):

Figure 9-16 Filtering on a symbol name match

2. Enter the symbol name filter.

The symbol name filter can contain the following characters:

*****	A multi-character wildcard.
?	A single-character wildcard.

RealView Debugger displays only those rows containing the symbol name you specify.

For example, to display only the rows containing the variable `Ptr_1_Glob`, you might use `Ptr_1_*`. Alternatively, you might use `Ptr_?_Glob`.

3. Click **Filter**.

See also

- *Filtering information in captured trace* on page 9-22.

9.5.6 Filtering for specific access types

To filter the information down to one or more selected access types, as contained in the `Type` column:

1. Select **Filter on Access Type...** from the **Filter** menu to display the List Selection dialog box. Figure 9-17 shows an example:

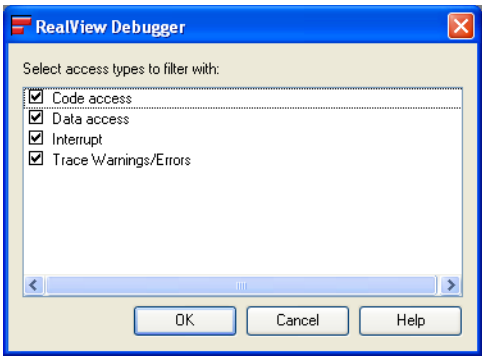


Figure 9-17 Filtering on an access type match

2. Select one or more access types to be included in the filtering operation. The following access types are available:
   - **Code access**
   - **Data access**
   - **Interrupt**
   - **Trace Warnings/Errors**.

   Note

   You can also perform Access Type filtering using the Rows entries in the **Trace Data** menu.

3. Click **OK**.

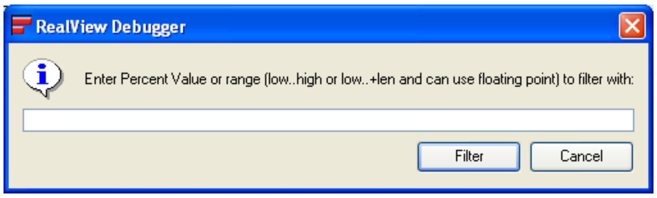
See also

- *Filtering information in captured trace* on page 9-22.

9.5.7 Filtering for specific percentages of execution time

To filter the information down to a specified percentage of execution time, as displayed in the `Exec%` column of the **Profile** tab:

1. Select the **Profile** tab.
2. Select **Filter on Percent Time...** from the **Filter** menu to display the Prompt dialog box. Figure 9-18 shows an example:



Enter Percent Value or range (low..high or low..+len and can use floating point) to filter with:

Figure 9-18 Filtering on a percent time match

3. Enter any of the following values:

   Percent value

   Specify a percent time value. For example, enter `50` if you want to filter the information to display only the row(s) representing the percentage-of-execution value of 50% or greater.

   Percent range

   Specify a *`low..high`* range of percent values, such as `50..60`, where RealView Debugger displays only those rows containing percentage-of-execution values within the range 50% to 60%.

   Percent range with offset

   Specify a *`low..+len`* range, such as `40..+10`, where RealView Debugger displays only those rows containing percentage-of-execution values within the range 40% to 50%. The *`len`* of 10 represents the offset value.

   You can also use floating point values such as `40.5`.

4. Click **Filter**.

See also

- *Filtering information in captured trace* on page 9-22.

9.5.8 Setting the AND condition for selected filters

Select **AND All Filters** from the **Filter** menu to set the `AND` condition for the selected filters.

For example, if you select **Filter on Position...** and **Filter on Data Value...**, then the filtering process returns trace information for only the areas of execution where both the position and data value match criteria you have entered are satisfied.

See also

- *Filtering information in captured trace* on page 9-22.

9.5.9 Setting the OR condition for selected filters

Select **OR All Filters** from the **Filter** menu to set an `OR` condition for the selected filters.

For example, if you select **Filter on Position...** and **Filter on Data Value...**, then the filtering process returns trace information for the areas of execution where either the position or data value match criterion you have entered is satisfied.

See also

- *Filtering information in captured trace* on page 9-22.

9.5.10 Inverting the sense of the selected filters

Select **Invert Filtering (NOT)** from the **Filter** menu to invert the sense of the selected filters.

Deselect the option to revert back to non-inverted filtering (the default).

For example, if you select **Filter on Position...** and **Filter on Data Value...**, then:

- for `AND` filtering, the filtering process returns trace information for the areas of execution except where both the position and data value match criteria you have entered are satisfied
- for `OR` filtering, the filtering process returns trace information for the areas of execution except where either the position or data value match criterion you have entered is satisfied.

See also

- *Filtering information in captured trace* on page 9-22.

9.5.11 Clearing filters

Select **Clear Filtering** from the **Filter** menu to clear any filters that you have set up on the results of a trace capture that has already been performed.

This does not affect the `AND`, `OR`, and invert (`NOT`) filtering options.

See also

- *Filtering information in captured trace* on page 9-22.

9.6 Viewing source for the captured trace

The **Source** tab in the Analysis window displays the source lines associated with each trace buffer entry. The buffer entries themselves are not displayed.

If you want to view the source with the captured trace, then you can interleave the source.

See also:
- *Interleaving source with the trace output* on page 9-7.

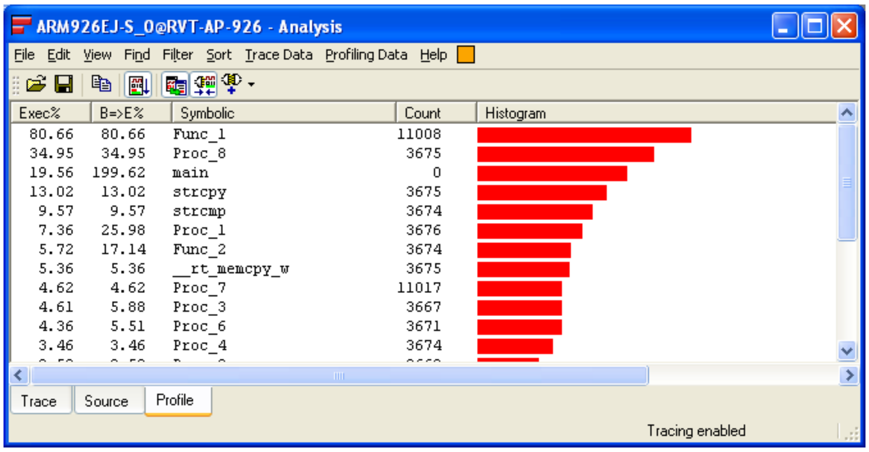
9.7 Viewing captured profiling information

Click the **Profile** tab in the Analysis window to view a statistical analysis of your trace information. You can use this tab to analyze control-flow information (branches), measure the time it takes to execute certain functions, and view call-graph data.

Note

The profile information is less accurate and in some cases might be incorrect if the captured trace is not continuous. To generate meaningful profiling information use trace start and end points at the boundaries of the functions you want to profile instead of, for example, setting ranges. It is important that you take into account which parts of your application image have been traced when interpreting profile information. For example, semihosting can affect the profiling results.

An example of the **Profile** tab is shown in Figure 9-19.



Exec%	B=>E%	Symbolic	Count	Histogram
80.66	80.66	Func_1	11008	
34.95	34.95	Proc_8	3675	
19.56	199.62	main	0	
13.02	13.02	strcpy	3675	
9.57	9.57	strcmp	3674	
7.36	25.98	Proc_1	3676	
5.72	17.14	Func_2	3674	
5.36	5.36	__rt_memcpy_w	3675	
4.62	4.62	Proc_7	11017	
4.61	5.88	Proc_3	3667	
4.36	5.51	Proc_6	3671	
3.46	3.46	Proc_4	3674	

Figure 9-19 Profile tab

The information displayed in this tab depends on the options you have selected in the **Profiling Data** menu.

See also:

- *Abandoning the profiling operation*
- *Collecting profiling data* on page 9-31
- *Changing the columns displayed in the Profile view* on page 9-31
- *Viewing call-graph data in the Profile view* on page 9-34
- *Interpreting profiling data* on page 9-35
- *Sorting profiling information* on page 9-40.

9.7.1 Abandoning the profiling operation

Depending on the trace options you have selected, the time taken to update the profiling information might be longer than you anticipate. If you want to abandon the profiling operation, then select another tab in the Analysis window.

See also

- *Viewing captured profiling information*.

9.7.2 Collecting profiling data

After you have collected the initial profiling data, you can choose to accumulate profiling data during subsequent runs. To do this, the following options are available in the Collection section of the **Profiling Data** menu:

Sum Profiling Data

Instructs RealView Debugger to sum multiple runs together, instead of displaying the data for individual runs separately. If you want to clear the accumulated profiling data and begin collecting data for a new series of runs, select the **Zero Profiling Data** option.

Zero Profiling Data

This option is only available when you select the **Sum Profiling Data** option. It instructs RealView Debugger to clear the accumulated profiling data and display the profiling data from the current trace only.

See also

- *Enable Timestamping* on page 4-10
- *Cycle accurate tracing* on page 4-11
- *Profiling in RealView Debugger* on page 9-3
- *Interleaving source with the trace output* on page 9-7
- *Changing the default format of time information* on page 9-13
- *Capturing profiling information* on page 10-84
- Chapter 4 *Configuring the ETM*.

9.7.3 Changing the columns displayed in the Profile view

You can choose which columns to display in the Profile view.

——— **Note** ———

If you are using a simulator, some columns might not be available. Columns that are not available are grayed out in the menu, and are not displayed by default.

————————

To change the columns displayed in the Profile view, select one of the following options from the **Profiling Data** menu in the Analysis window:

First Instance

Displays the `1st` column that shows the element number of the first instance of each function in the captured trace.

This column is not displayed by default.

Time% in Self

Displays the `Exec%` column that shows, as a percentage of the whole, the time spent in the range of this function or module, where the entire trace buffer represents 100%. This represents the `PC` in the range of the function itself, and does not include time spent in the descendents of the function.

This column is displayed by default.

——— **Note** ———

Inlined functions count towards the time of the function into which they are inlined.

————————

Time% Including Children (B=>E%)

Displays the `B=>E%` column that shows, as a percentage of the whole, the time elapsed from the beginning to the end of the range. This includes time spent in the descendents of the function.

This column is displayed by default.

Range Type Displays the `Type` column that shows the type of range. This is usually `Func`, indicating a function. Where there are no functions, for example if the code is in assembly language, the range type is `Module`.

This column is not displayed by default.

Range Symbol

Displays the `Symbolic` column that shows the name of the function or module that is profiled.

This column is displayed by default

Address as Value

Displays the `Address` column that shows the address of the symbol that is profiled.

This column is not displayed by default

Exec/B=>E/B=>E Average

Displays the following columns:

Exec/cycl

This column shows the total time spent in execution(s) of this function.

This column is not displayed by default.

B=>E This column shows the total beginning to end time of all executions of this function.

This column is not displayed by default.

B=>E Avg This column shows the average of all the individual times spent on the execution of this function.

This column is not displayed by default.

Count of Calls

Displays the `Count` column that shows the number of times that the function was entered and exited.

This column is displayed by default.

——— **Note** ———

If the trace begins or ends within a function, then that instance of the function is not counted. For an instance of a function to be counted, both entry to and exit from the function must be traced.

———————

Min/Max Times

Displays the following columns:

Min B=>E This column shows the minimum time spent executing the function. This is especially useful when a particular function is executed several times, but for different tasks, and you want to see the lowest value of the execution times involved. The time is displayed in the format that is currently selected for analysis.

This column is not displayed by default

Max B=>E This column shows the maximum time spent executing the function. This is especially useful when a particular function is executed several times, but for different tasks, and you want to see the highest value of the execution times involved. The time is displayed in the format that is currently selected for analysis.

This column is not displayed by default.

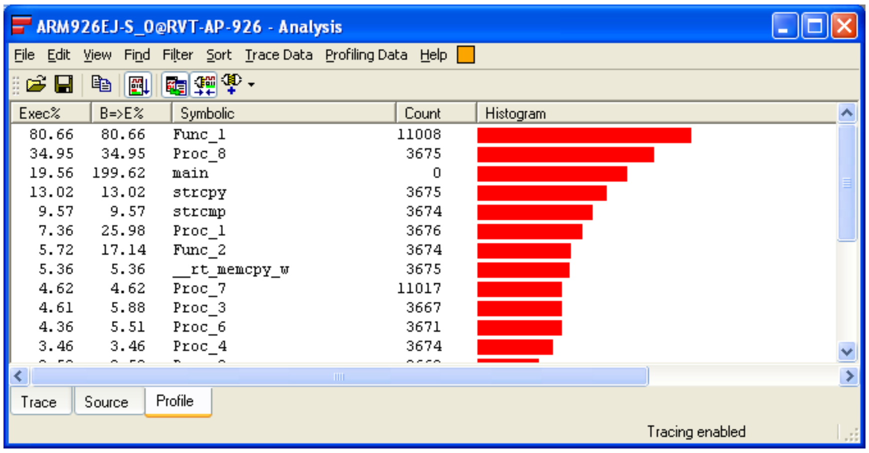
Histogram View

Displays the `Histogram` column that shows the graphically the time spent in each function. You can view the histogram as a linear or a logarithmic function. By default, the histogram is displayed as a linear function.

To view the histogram as a logarithmic function of the Exec% or B=>E, select **Use Logarithmic Scale for Histogram** from the **Profiling Data** menu.

This column is displayed by default.

Figure 9-20 shows an example histogram.


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

Exec%	B=>E%	Symbolic	Count	Histogram
80.66	80.66	Func_1	11008	
34.95	34.95	Proc_8	3675	
19.56	199.62	main	0	
13.02	13.02	strcpy	3675	
9.57	9.57	strcmp	3674	
7.36	25.98	Proc_1	3676	
5.72	17.14	Func_2	3674	
5.36	5.36	__rt_memcpy_w	3675	
4.62	4.62	Proc_7	11017	
4.61	5.88	Proc_3	3667	
4.36	5.51	Proc_6	3671	
3.46	3.46	Proc_4	3674	

Figure 9-20 Example histogram

——— **Note** ———

The histogram is not available if timing information is not present. If you want to view profiling information, you must enable either timestamping or cycle-accurate tracing, or both. If you have an ETMv3 processor, then instruction count-based profiling information is shown, unless you enable one of these options. You can configure both of these options using the Configure ETM dialog box. Better results in profiling are generally obtained with cycle-accurate tracing enabled.

See also

- *Enable Timestamping* on page 4-10
- *Cycle accurate tracing* on page 4-11
- *Profiling in RealView Debugger* on page 9-3
- *Interleaving source with the trace output* on page 9-7
- *Changing the default format of time information* on page 9-13
- *Capturing profiling information* on page 10-84
- Chapter 4 *Configuring the ETM*.

9.7.4 Viewing call-graph data in the Profile view

Call-graph data enables you to view a list of parents and/or children for each function. The data for the function being profiled is colored black, and the histogram for that function, if present, is colored red.

When parents and/or children are displayed, a ruler line separates each group of function, parents and children.

To view Call-graph data, select one or both of the following options from the **Profiling Data** menu:

Parents of Function

Displays the parents of the function. A parent is a function that makes a call to the function being profiled. The **Exec%** and **B=>E%** values shown for parents are the times spent in the child when called from this parent.

Parents are displayed before the function line, and the Symbolic information is indented. The data for parents is a light gray color, and the parent histograms, if present, are colored blue.

This option is disabled by default.

Children of Function

Displays the children of the function. A child is a function that is called by the function being profiled. The **Exec%** and **B=>E%** values shown for children are time spent in this function when called from the parent.

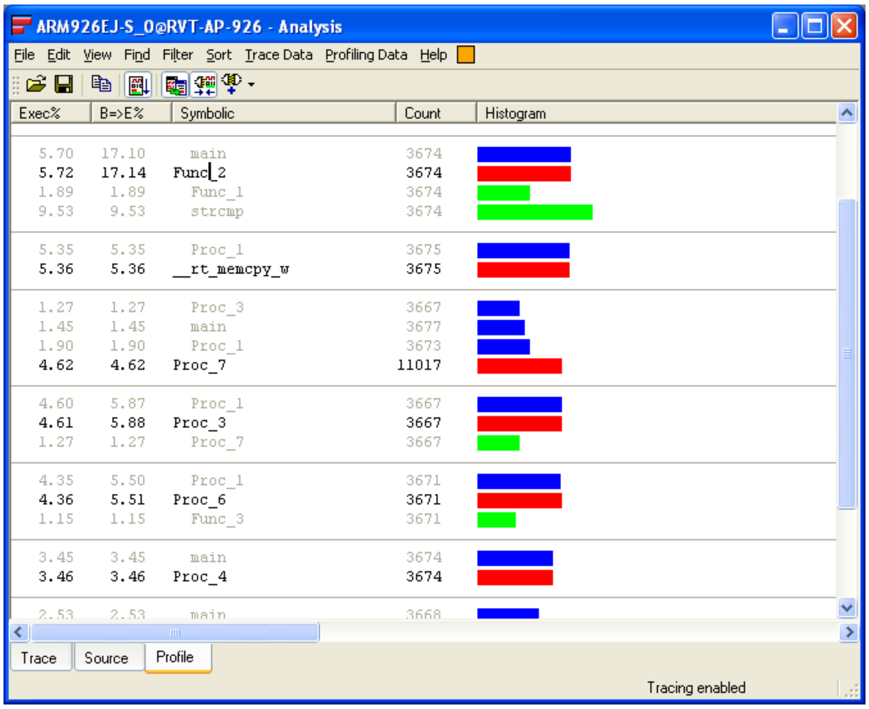
Children are displayed after the function line, and the Symbolic information is indented. The data for children is a light gray color, and the child histograms, if present, are colored green.

This option is disabled by default.

Note

Where the compiler inlines function calls, these inline functions are not shown as children.

Figure 9-21 on page 9-35 shows example call-graph data.


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Exec%   B=>E%   Symbolic            Count   Histogram

 5.70   17.10     main               3674   🟦
 5.72   17.14   Func_2               3674   🟥
 1.89    1.89     Func_1             3674   🟩
 9.53    9.53     strcmp             3674   🟩

 5.35    5.35     Proc_1             3675   🟦
 5.36    5.36   __rt_memcpy_w        3675   🟥

 1.27    1.27     Proc_3             3667   🟦
 1.45    1.45     main               3677   🟦
 1.90    1.90     Proc_1             3673   🟦
 4.62    4.62   Proc_7              11017   🟥

 4.60    5.87     Proc_1             3667   🟦
 4.61    5.88   Proc_3               3667   🟥
 1.27    1.27     Proc_7             3667   🟩

 4.35    5.50     Proc_1             3671   🟦
 4.36    5.51   Proc_6               3671   🟥
 1.15    1.15     Func_3             3671   🟩

 3.45    3.45     main               3674   🟦
 3.46    3.46   Proc_4               3674   🟥

 2.53    2.53     main               3668   🟦
```

Figure 9-21 Example of call-graph information

The precise meaning of the call-graph values depends on the Data Interpretation option you have selected.

See *Interpreting profiling data* for information on how to select these options.

See also

- *Enable Timestamping* on page 4-10
- *Cycle accurate tracing* on page 4-11
- *Profiling in RealView Debugger* on page 9-3
- *Interleaving source with the trace output* on page 9-7
- *Changing the default format of time information* on page 9-13
- *Capturing profiling information* on page 10-84
- Chapter 4 *Configuring the ETM*.

9.7.5 Interpreting profiling data

To help you to interpret the profiling data, you can choose one or more Data Interpretation options from the **Profiling Data** menu. These options are:

Ignore Holes in Trace

Instructs RealView Debugger to ignore holes in the trace data caused by discontinuities. Discontinuities might occur, for example, when the processor is in debug state, or when tracing is disabled. When this option is selected, holes are not included in the whole time used to calculate the values for **Exec%** and **B=>E%**.

Use Logarithmic Scale for Histogram

When this option is selected, the length of the histogram bar is calculated using a logarithmic function of the **Exec%** or **B=>E%** value. When this option is disabled, a simple linear scale is used.

Parent/Child %ages Relative to Whole Time

Enables you to specify the value that time percentages are calculated relative to. When this option is selected, the values shown for **Exec%** and **B=>E%** for parents and children are shown as a percentage of the whole time traced. This setting is the default.

See *Viewing the Parent/Child %ages Relative to Whole Time* for detailed information on how to interpret call-graph data when this option is selected.

Parent/Child %ages Relative to Function B=>E

Enables you to specify the value that time percentages are calculated relative to. When this option is selected, the values shown for **Exec%** and **B=>E%** for parents and children are shown relative to the total B=>E time of the function.

See *Viewing the Parent/Child %ages Relative to Function B=>E* on page 9-37 for detailed information on how to interpret call-graph data when this option is selected.

Parent/Child %ages Relative to Parent/Child B=>E

Enables you to specify the value that time percentages are calculated relative to. When this option is selected, the values shown for **Exec%** and **B=>E%** for parents and children are shown relative to the B=>E time of the parents and children.

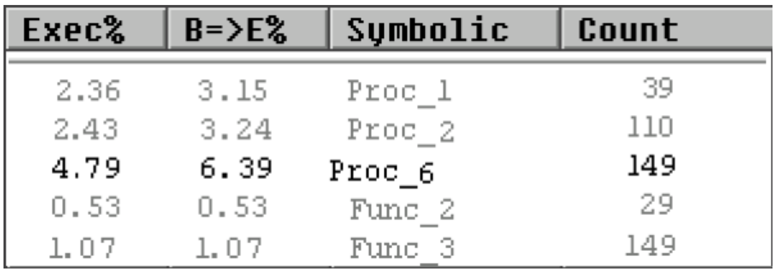
See *Viewing the Parent/Child %ages Relative to Parent/Child B=>E* on page 9-38 for detailed information on how to interpret call-graph data when this option is selected.

Note

B=>E refers to *Beginning to End*.

Viewing the Parent/Child %ages Relative to Whole Time

This is the default view. When this option is selected, the values shown for `Exec%` and `B=>E%` for parents and children are shown as a percentage of the whole time traced. Figure 9-22 gives an example of call-graph data for a function with its parents and children.



Exec%	B=>E%	Symbolic	Count
2.36	3.15	Proc_1	39
2.43	3.24	Proc_2	110
4.79	6.39	Proc_6	149
0.53	0.53	Func_2	29
1.07	1.07	Func_3	149

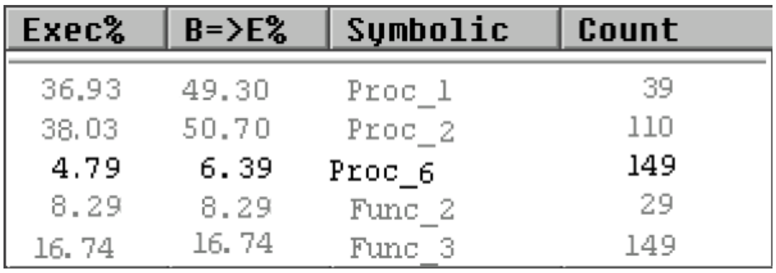
Figure 9-22 Example call-graph data (Parent/Child %ages Relative to Whole Time)

In this example, when **Parent/Child %ages Relative to Whole Time** is selected:

- The Exec% column shows that:
  - 2.36% of the total execution time was spent in code of the function Proc_6 when called from the parent Proc_1.
  - 2.43% of the total execution time was spent in code of the function Proc_6 when called from the parent Proc_2.
  - 4.79% of the total execution time was spent in code of the function Proc_6.
  - 0.53% of the total execution time was spent in code of the function Func_2 when called as a child from Proc_6.
  - 1.07% of the total execution time was spent in code of the function Func_3 when called as a child from Proc_6.
- The B=>E% column shows that:
  - 3.15% of the total execution time was spent in calls to function Proc_6 and its children when called from the parent Proc_1.
  - 3.24% of the total execution time was spent in calls to function Proc_6 and its children when called from the parent Proc_2.
  - 6.39% of the total execution time was spent in calls to function Proc_6 and its children.
  - 0.53% of the total execution time was spent in code of the function Func_2 and its children when called as a child from Proc_6.
  - 1.07% of the total execution time was spent in code of the function Func_3 and its children when called as a child from Proc_6.
- The Count column shows that:
  - There were 39 calls from the function Proc_1 to the function Proc_6.
  - There were 110 calls from the function Proc_2 to the function Proc_6
  - There were 149 calls to the function Proc_6.
  - There were 29 calls to the function Func_2 from the function Proc_6.
  - There were 149 calls to the function Func_3 from the function Proc_6.

Viewing the Parent/Child %ages Relative to Function B=>E

When this option is selected, the values shown for Exec% and B=>E% for parents and children are shown relative to the total B=>E time of the function. Figure 9-23 gives an example of call-graph data for a function with its parents and children.



Exec%	B=>E%	Symbolic	Count
36.93	49.30	Proc_1	39
38.03	50.70	Proc_2	110
4.79	6.39	Proc_6	149
8.29	8.29	Func_2	29
16.74	16.74	Func_3	149

Figure 9-23 Example call-graph data (Parent/Child %ages Relative to Function B=>E)

In this example, when **Parent/Child %ages Relative to Function B=>E** is selected:

- The `Exec%` column shows that:
  - — 36.93% of the total time spent in Proc_6 and its children was spent in `Proc_6` when called from `Proc_1`.
  - — 38.03% of the total time spent in `Proc_6` and its children was spent in `Proc_6` when called from `Proc_2`.
  - — 8.29% of the total time spent in `Proc_6` and its children was spent in `Func_2` when called from `Proc_6`.
  - — 16.74% of the total time spent in `Proc_6` and its children was spent in `Func_3` when called from `Proc_6`.

  ——— **Note** ———

  These four figures add up to 100% because the total time spent in `Proc_6` and its children is divided between the time spent in `Proc_6` when called from its parents, and the time spent in the children when called from `Proc_6`. This is because, in this example, there are no grandchildren.

- The `B=>E%` column shows that:
  - — 49.30% of the total time was spent in `Proc_6` and its children when `Proc_6` was called from `Proc_1`.
  - — 50.70% of the total time was spent in `Proc_6` and its children when Proc_6 was called from `Proc_2`.
  - — 8.29% of the total time spent in `Proc_6` and its children was spent in `Func_2` and its children when called from `Proc_6`.

    ——— **Note** ———

    This figure is the same as that in the **Exec%** column only because there are no grandchildren.

  - — 16.74% of the total time spent in `Proc_6` and its children was spent in `Func_3` and its children when called from `Proc_6`.

    ——— **Note** ———

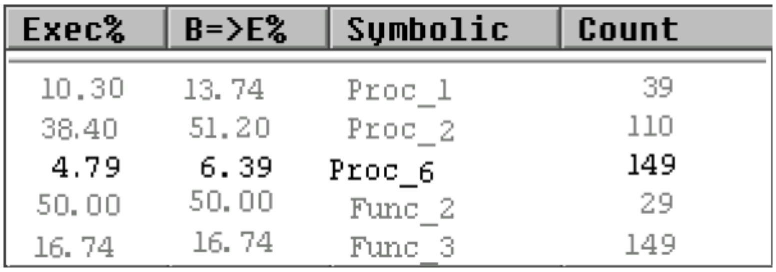
    This figure is the same as that in the **Exec%** column only because there are no grandchildren.

  ——— **Note** ———

  The value for the parents always totals 100%. The total time is split between the parents.

Viewing the Parent/Child %ages Relative to Parent/Child B=>E

When this option is selected, the values shown for `Exec%` and `B=>E%` for parents and children are shown relative to the B=>E time of the parents and children. Figure 9-24 on page 9-39 gives an example of call-graph data for a function with its parents and children.



Exec%	B=>E%	Symbolic	Count
10.30	13.74	Proc_1	39
38.40	51.20	Proc_2	110
4.79	6.39	Proc_6	149
50.00	50.00	Func_2	29
16.74	16.74	Func_3	149

Figure 9-24 Example call-graph data (Parent/Child %ages Relative to Parent/Child B=>E)

In this example, when **Parent/Child %ages Relative to Parent/Child B=>E** is selected:

- The `Exec%` column shows that:
  - 10.30% of the total time spent in `Proc_1` and its children was spent in `Proc_6` when called from `Proc_1`.
  - 38.40% of the total time spent in `Proc_2` and its children was spent in `Proc_6` when called from `Proc_2`.
  - 50.00% of the total time spent in `Func_2` and its children was spent in `Func_2` when called from `Proc_6`
  - 16.74% of the total time spent in `Func_3` and its children was spent in `Func_3` when called from `Proc_6`.
- The `B=>E%` column shows that:
  - 13.74% of the total time spent in `Proc_6` and its children was spent in `Proc_6` and its children when called from `Proc_1`.
  - 51.20% of the total time spent in `Proc_6` and its children was spent in `Proc_6` and its children when called from `Proc_2`.
  - 50.00% of the total time spent in `Proc_6` and its children was spent in `Func_2` and its children when called from `Proc_6`.

    Note

    This figure is the same as that in the **Exec%** column only because there are no grandchildren.

  - 16.74% of the total time spent in `Proc_6` and its children was spent in `Func_3` and its children when called from `Proc_6`.

    Note

    This figure is the same as that in the **Exec%** column only because there are no grandchildren.

See also

- *Enable Timestamping* on page 4-10
- *Cycle accurate tracing* on page 4-11
- *Profiling in RealView Debugger* on page 9-3
- *Interleaving source with the trace output* on page 9-7
- *Changing the default format of time information* on page 9-13
- *Capturing profiling information* on page 10-84
- Chapter 4 *Configuring the ETM*.

9.7.6 Sorting profiling information

The options in the **Sort** menu enable you to sort the information in the **Profile** tab by a specified column. Information can be sorted in ascending or descending order. There are two ways you can change the sorting order of the Analysis window output:

- Select one of the **By**... options in the **Sort** menu that determine which column is to be used as the sort key. You can also select the **Reverse Sort** option to reverse the order of the selected sort.
- Click on the column header for the column you want to sort by. If you click on the same column header again, the sorting order is reversed.

  Note

  If you click on a column header to sort by a particular column, the corresponding item in the **Sort** menu is automatically selected. If you click on the column header again to reverse the order, the item **Reverse Sort** is automatically selected.

The complete **Sort** menu options are as follows:

By Time Sorts the output by the `Exec` column. This is the default sort option.

By Address Sorts the output by the `Address` column.

By Name Sorts the output by the `Symbolic` column.

By Access Type

Sorts the output by the `Type` column.

By Count Sorts the output by the `Count` column.

By First Instance

Sorts the output by the `1st` column.

By Total B=>E Time

Sorts the output by the `B=>E` column.

By Average B=>E Time

Sorts the output by the `Avg B=>E` column.

By Minimum B=>E Time

Sorts the output by the `Min B=>E` column.

By Maximum B=>E Time

Sorts the output by the `Max B=>E` column.

Reverse Sort

Reverses the order of the sort you have selected. To return to ascending-order sorting (the default) on the specified column, you must deselect this option.

See also

- *Enable Timestamping* on page 4-10
- *Cycle accurate tracing* on page 4-11
- *Profiling in RealView Debugger* on page 9-3
- *Interleaving source with the trace output* on page 9-7
- *Changing the default format of time information* on page 9-13

- *Capturing profiling information* on page 10-84
- Chapter 4 *Configuring the ETM*.

9.8 Copying selected text from the trace output

To copy selected text from the trace output to the clipboard:

1. Click the **Trace** tab in the Analysis window.
2. Select the text in the trace output that you want to copy.
3. Select **Copy** from the **Edit** menu to copy the selected text to the clipboard.
4. Paste the copied text to a suitable text editor.

9.9 Saving and loading trace information

The following sections describe how to store and retrieve captured trace and profiling information:

- *Trace file formats*
- *Saving the trace buffer to a file* on page 9-44
- *Saving a filtered trace buffer to a file* on page 9-45
- *Loading the trace buffer from a file* on page 9-46
- *Closing a loaded file* on page 9-46.

9.9.1 Trace file formats

The supported trace file formats depend on the type of target.

Trace file formats for ETM-enabled targets

For ETM-enabled targets the following trace file formats are supported:

Raw trace output

Stores non-defunneled trace in compressed form to an XML file. You can reload this file type into the Analysis window.

Compressed trace for current source

Stores defunneled trace in compressed form to an XML file for the current trace source. You can reload this file type into the Analysis window.

Compressed trace for specified source

Stores defunneled trace in compressed form to an XML file for a specified trace source. You can reload this file type into the Analysis window.

Decompressed trace (Output Only)

Stores the uncompressed trace in an XML file. This file type cannot be reloaded into the Analysis window.

Note

Be aware that very large XML files can be created when saving uncompressed trace.

Text file containing display lines (Output Only)

Stores a tabulated text file, with the extension `.txt`, containing what is displayed in the current tab view of the Analysis window. This file type cannot be reloaded into the Analysis window.

Trace file formats for RVISS targets

For RVISS targets the following trace file formats are supported:

Full dump of Trace contents

Stores a binary file, with the extension `.trc`, containing the complete information that RealView Debugger uses to generate the trace information, including any profiling information.

Minimal dump of Trace contents (timing+address+type)

Stores a binary file, with the extension `.trm`, containing only the timing, address, and type information that RealView Debugger uses to generate the trace information, including any profiling information. When you load this file in the future, RealView Debugger reconstructs the full trace information from these three attributes.

——— Note ———

If you load a file of this type in a future trace session, the data values present at the memory locations might be different from those present when you originally saved this file. In addition, any errors and warnings are not stored.

———————————

Profiling data

Stores a binary file, with the extension `.trp`, containing only the profiling information that RealView Debugger uses. Unlike the **Minimal dump of Trace contents** option, RealView Debugger cannot reconstruct full trace information based on the contents of this file. However, if you want to save only profiling information, it is recommended you use this file type because it takes up significantly less space than a `.trc` file.

Text file containing display lines

Stores the current tab view of the Analysis window in a tabulated text file, with the extension `.txt`. This file type cannot be reloaded into the Analysis window.

9.9.2 Saving the trace buffer to a file

To save the current trace information to a file:

1. Select **Save Trace to File...** from the **File** menu to display the List Selection dialog box.
2. Select the type of trace file that you want to save.
3. Click **OK**. The next step depends on the type of trace file you selected:
   - if you selected **Compressed trace for specified source**, continue at step 4
   - if you selected **Decompressed trace (Output Only)**, continue at step 5
   - for any other type, continue at step 6.
4. If you selected **Compressed trace for specified source**, you are prompted to enter the ID of the trace source:
   a. Enter the ID of the required trace source.
   b. Click **Set**.
   c. Continue at step 6.
5. If you selected **Decompressed trace (Output Only)**, you are prompted to continue:
   a. Click **OK** to continue.
      Otherwise, click **Cancel**.
   b. Continue at step 6.

   ——— Note ———

   Be aware that very large XML files can be created when saving decompressed trace.

   ———————————

6. A Save Trace to File dialog box is displayed.

7. Locate the directory where you want to save the trace file.
8. Specify a filename. The default filename extension depends on the file type you have selected.
9. Click **Save**.

See also

- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the `TRACEBUFFER` command.

9.9.3 Saving a filtered trace buffer to a file

To save the post-filtered trace information to a file:

1. Select **Save Filtered Trace to File...** from the **File** menu to display the List Selection dialog box.
2. Select the type of trace file that you want to save.
3. Click **OK**.

   If you select **Compressed trace for specified source**, continue at step 4. Otherwise, continue at step 5.
4. You are prompted to enter the ID of the trace source:
   a. Enter the ID of the required trace source.
   b. Click **Set**.
5. A Save Trace to File dialog box is displayed.
6. Locate the directory where you want to save the trace file.
7. Specify a filename. The default filename extension depends on the file type you have selected.
8. Click **Save**.

See also

- *Filtering information in captured trace* on page 9-22
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the `TRACEBUFFER` command.

9.9.4 Loading the trace buffer from a file

To load a previously saved trace buffer from a file, which you can re-analyze:

1. Select **Load Trace from File...** from the **File** menu to display the Select Trace File to Load dialog box.
2. Locate the required trace file that you previously saved.
3. Click **Open**.

This is useful in cases where you are performing a trace capture that takes a long time to reach the point of interest, and you do not want to have to repeat the process. You can also analyze the profiling information of the saved trace buffer even after you continue to make modifications to the source code.

——— **Note** ———

If you have captured trace to a file using the RVISS Tracer feature, you cannot load it into the Analysis window.

————————

See also

- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the `TRACEBUFFER` command.
- the Tracer feature in the *RealView ARMulator ISS User Guide*.

9.9.5 Closing a loaded file

Select **Close Loaded File** from the **File** menu to close the file that is currently loaded in the Analysis window, and clear the trace information from the window.

9.10 Changing the current connection

If you have multiple connections established, then you can change the current connection from the Analysis window.

Click **Cycle Connections** to change the current connection.

If you have captured trace information for the target associated with the connection, then the Analysis window changes to show the captured trace for that target.

See also:

- *Attaching the Analysis window to the current connection* on page 9-48
- the following in the *RealView Debugger User Guide*:
  - *Changing the current target connection* on page 3-50.

9.11 Attaching the Analysis window to the current connection

To attach the Analysis window that has the focus to the current connection:

1. Change the current connection, if required.
2. Select **Attach Window to a Connection** from the **File** menu.

   The Analysis window title changes to include `[Target]` to shows that the window is attached to the connection.

See also:

- *Changing the current connection* on page 9-47
- the following in the *RealView Debugger User Guide*:
  - *Attaching a Code window to a connection* on page 7-10.

9.12 Mapping Analysis window options to CLI commands and qualifiers

The following sections identify the equivalent CLI commands and qualifiers to use for various Analysis window menu options:

- *File menu options*
- *Edit menu options* on page 9-50
- *View menu options* on page 9-51
- *Find menu options* on page 9-51
- *Filter menu options* on page 9-52.

Note

When running in command line mode, the trace output is not displayed on the screen. Therefore, you must save the trace buffer to a file. You can perform find and filter operations on the trace buffer using CLI commands. However, if you want to analyze the trace output, you must start RealView Debugger in GUI mode, and load the trace buffer file into the Analysis window.

9.12.1 File menu options

Table 9-1 shows the **File** menu options and the equivalent qualifiers to use with the `TRACEBUFFER` command to perform the same function in the CLI.

Table 9-1 File menu to TRACEBUFFER command qualifier mapping

File menu option	Command qualifier
Load Trace from File…	`loadfile`
Save Trace to File…, then one of the following options from the List Selection dialog box:	
• **Text file containing display lines**	`savefile,ascii`
• **Full dump of trace contents**	`savefile,full`
• **Minimal dump of trace contents**	`savefile,minimal`
• **Profiling data**.	`savefile,profile`
Save Filtered Trace to File…, then one of the following options from the List Selection dialog box:	
• **Text file containing display lines**	`savefile,ascii,filtered`
• **Full dump of trace contents**	`savefile,full,filtered`
• **Minimal dump of trace contents**	`savefile,minimal,filtered`
• **Profiling data**.	`savefile,profile,filtered`
Close Loaded File	`closefile`

See also

- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — *Alphabetical command reference* on page 2-12.

9.12.2 Edit menu options

Table 9-2 shows the **Edit** menu options and the equivalent commands and qualifiers to use to perform the same function in the CLI.

Table 9-2 Edit menu to command qualifier mapping

Edit menu option	Command qualifier
Connect Analyzer…	`ANALYZER ,connect`
Disconnect Analyzer	`ANALYZER ,disconnect`
Tracing Enabled (check mark)	`ANALYZER ,enable`
Tracing Enabled (no check mark)	`ANALYZER ,disable`
Configure Analyzer Properties...	`ETM_CONFIG`
Set Trace Buffer Size…	`ANALYZER ,set_size:(`*`n`*`)`
Store Control-flow Changes Only (check mark)	`ANALYZER ,collect_flow`
Store Control-flow Changes Only (no check mark)	`ANALYZER ,collect_all`
Buffer Full Mode → Stop Processor on Buffer Full	`ANALYZER ,full_stop`
Buffer Full Mode → Stop Collecting on Buffer Full	`ANALYZER ,full_ignore`
Buffer Full Mode → Continue Collecting on Buffer Full	`ANALYZER ,full_ring`
Trigger Mode → Collect Trace Before Trigger	`ANALYZER ,before`
Trigger Mode → Collect Trace Around Trigger	`ANALYZER ,around`
Trigger Mode → Collect Trace After Trigger	`ANALYZER ,after`
Trigger Mode → Stop Processor on Trigger (check mark)	`ANALYZER ,stop_on_trigger`
Trigger Mode → Stop Processor on Trigger (no check mark)	`ANALYZER ,continue_on_trigger`
Data Tracing Mode → Address Only	`ANALYZER ,addronly`
Data Tracing Mode → Data Only	`ANALYZER ,dataonly`
Data Tracing Mode → Data and Address	`ANALYZER ,fulltrace`
Automatic Tracing Mode → Off (Use tracepoints)	`ANALYZER ,auto_off`
Automatic Tracing Mode → Instructions Only	`ANALYZER ,auto_instronly`
Automatic Tracing Mode → Data Only (ETMv3 only)	`ANALYZER ,auto_dataonly`
Automatic Tracing Mode → Instructions and Data	`ANALYZER ,auto_both`
Set/Edit Event Triggers…	`ANALYZER ,triggers`
Clear All Event Triggers	`ANALYZER ,clear_triggers`
Physical to Logical Address Mapping…	`ANALYZER ,map_log_phys`

See also

- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13

- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

9.12.3 View menu options

Table 9-3 shows the **View** menu options and the equivalent commands and qualifiers to use to perform the same function in the CLI.

Table 9-3 View menu to command qualifier mapping

View menu option	Command qualifier
Update	`TRACEBUFFER ,refresh`
Clear Trace Buffer	`ANALYZER ,clear`
Show Position Relative to Trigger (check mark)	`TRACEBUFFER ,pos_relative`
Show Position Relative to Trigger (no check mark)	`TRACEBUFFER ,pos_absolute`
Scale Time Units...	`TRACEBUFFER ,scaletime=`*n*
Define Processor Speed for Scaling…	`TRACEBUFFER ,speed=`*n*

See also

- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

9.12.4 Find menu options

Table 9-4 shows the **Find** menu options and the equivalent qualifiers to use with the `TRACEBUFFER` command to perform the same function in the CLI.

Note

The equivalent CLI commands can be used only when running RealView Debugger in GUI mode.

Table 9-4 Find menu to TRACEBUFFER command qualifier mapping

Find menu option	Command qualifier
Find Trigger	`find_trigger`
Find Position…	`find_position`
Find Timestamp…	`find_time`
Find Address Expression…	`find_address`
Find Data Value…	`find_data`
Find Symbol Name…	`find_name`

See also

- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

9.12.5 Filter menu options

Table 9-5 shows the **Filter** menu options and the equivalent qualifiers to use with the `TRACEBUFFER` command to perform the same function in the CLI.

——— **Note** ———

The equivalent CLI commands can be used only when running RealView Debugger in GUI mode.

Table 9-5 Filter menu to TRACEBUFFER command qualifier mapping

Filter menu option	Command qualifier
Filter on Position…	`posfilter`
Filter on Timestamp…	`timefilter`
Filter on Address Expression…	`addrfilter` | `addressfilter`
Filter on Data Value…	`dvafilter` | `datavaluefilter`
Filter on Symbol Name…	`namefilter`
Filter on Access Type…	`typefilter` | `accesstypefilter`
Filter on Percent Time…	`percentfilter`
AND All Filters (check mark)	`and_filter`
OR Filters (check mark)	`or_filter`
Invert Filtering (NOT) (check mark)	`invert_filter`
Invert Filtering (NOT) (no check mark)	`normal_filter`
Clear Filtering	`clearfilter`

See also

- *Equivalent ETM_CONFIG CLI command qualifiers* on page 4-13
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

Chapter 10
Tracing Tutorial

This chapter provides a tutorial on how you can use the tracing features of RealView® Debugger to solve typical development problems and analyze certain elements of the execution of your application. It includes:

10.1 About the tracing tutorial

It is recommended that you follow this tutorial before using trace on your own application images. The tutorial is designed to introduce you to the trace features of RealView Debugger, and does not assume that you have any experience of using RealView Debugger.

See also:

- *About the trace hardware used in the trace tutorial*
- *About the sample application used in the trace tutorial*
- Chapter 9 *Analyzing Trace with the Analysis Window*.

10.1.1 About the trace hardware used in the trace tutorial

The tutorial assumes that you have the following RealView Debugger-supported trace hardware components, which must be installed and connected properly:

- A JTAG interface unit, such as RealView ICE.
- A *Trace Port Analyzer* (TPA), such as RealView Trace.
- An *Embedded Trace Macrocell*™ (ETM™) enabled ARM® architecture-based processor.

  Note

  This tutorial uses an ARM926EJ-S™ processor on an ARM Integrator™/AP development board. If you are using a different development platform, the tracing features available depends on your ETM version. Therefore, some features described in this tutorial might not be available.

See also

- *About the trace hardware setup* on page 2-3
- Appendix A *Setting up the Trace Hardware*
- Appendix B *Setting up the Trace Software*.

10.1.2 About the sample application used in the trace tutorial

This tutorial uses the `trace` sample application provided in the *RealView Development Suite* (RVDS) examples directory. It simulates a small system that:

- reads a set of input data samples and computes the sample average
- provides a framework for common instruction and data trace scenarios.

The application is designed to run on any hardware platform because it simulates data input and output rather than relying on specific peripherals. Data sampling and processing is initiated on a random time basis using the `rand()` function. Instead of reading data from an input device, such as an analog-to-digital converter, new input data is generated from previous input data. The sample average and input samples are output by writing to a fixed address in memory, which is intended to simulate the write buffer of a serial port.

Building the application

The build files, `build.bat` and `build.mk`, are supplied to build the application using the RVDS build tools. The batch file compiles `trace.c` and links the application using the scatter-loading file `trace.scat`. By default, the image is compiled for the ARM7TDMI® processor (ARM architecture v4T). The image can run on all ARM architecture-based cores that support the

ARMv4T architecture. If you want to compile for a specific processor or architecture, use the `--cpu` option. However, the addresses given in the tutorial might be different, and there might also be differences in the trace output.

The scatter-loading file places the executable image at `0x8000`, followed by the RW and ZI data sections. The application uses a one-region memory model, with the heap and stack placed 256 bytes after the ZI section. The simulated write buffer is located in a separate section at address `0x20000`. To run the application from a different address or relocate any of the memory sections, you must modify the `trace.scat` file accordingly.

Viewing the image memory map

To view the image memory map in RealView Debugger:

1. Select **Memory Map tab** from the **View** menu. The Memory Map tab in the Process Control view is displayed.
2. Right-click in the **Memory Map** tab to display the context menu.
3. Select **Memory Mapping** from the context menu to enable memory mapping. Figure 10-1 shows an example:

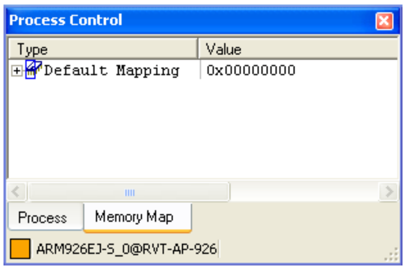
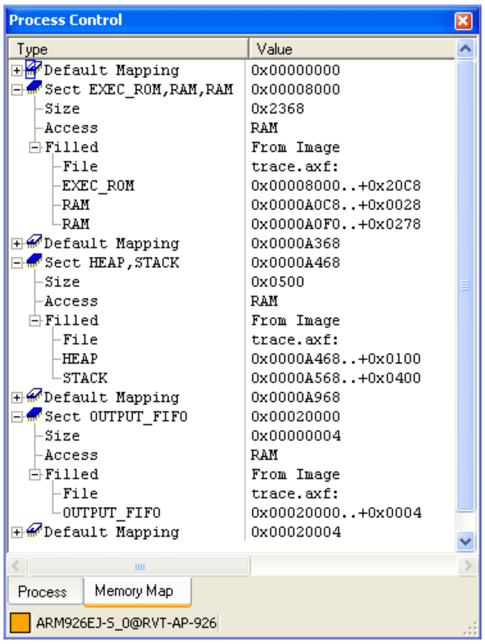


Figure 10-1 Memory Map tab

4. Load the `trace.axf` image. The **Memory Map** tab is updated with the image details.
5. Expand the `Sect` and `Filled` groups in the memory map to see the image details. Figure 10-2 on page 10-4 shows an example:



Process Control

```
Type                        Value
⊞Default Mapping            0x00000000
⊟Sect EXEC_ROM,RAM,RAM      0x00008000
  ├Size                     0x2368
  ├Access                   RAM
  ⊟Filled                   From Image
    ├File                   trace.axf:
    ├EXEC_ROM               0x00008000..+0x20C8
    ├RAM                    0x0000A0C8..+0x0028
    └RAM                    0x0000A0F0..+0x0278
⊞Default Mapping            0x0000A368
⊟Sect HEAP,STACK            0x0000A468
  ├Size                     0x0500
  ├Access                   RAM
  ⊟Filled                   From Image
    ├File                   trace.axf:
    ├HEAP                   0x0000A468..+0x0100
    └STACK                  0x0000A568..+0x0400
⊞Default Mapping            0x0000A968
⊟Sect OUTPUT_FIFO           0x00020000
  ├Size                     0x00000004
  ├Access                   RAM
  ⊟Filled                   From Image
    ├File                   trace.axf:
    └OUTPUT_FIFO            0x00020000..+0x0004
⊞Default Mapping            0x00020004
```

Figure 10-2 Image details in the memory map

Considerations for the Versatile AB926EJ-S development board

If you are using a Versatile AB926EJ-S development board for this tutorial, then you must lower the clock speed to less than 140 MHz. If you modify the program `trace.c` to do this, then all line numbers and addresses mentioned in the tutorial must be adjusted accordingly.

See also

- *About the trace hardware setup* on page 2-3
- the following in the *RealView Debugger User Guide*:
  - *Viewing image details* on page 4-9
- *ARM® Compiler toolchain Using the Compiler*
- *ARM® Compiler toolchain Compiler Reference*
- *ARM® Compiler toolchain Using the Linker*
- *ARM® Compiler toolchain Linker Reference*.

10.2 Preparing to start the tracing tutorial

Before you start the tracing tutorial, you must prepare your trace hardware and set up RealView Debugger to begin tracing.

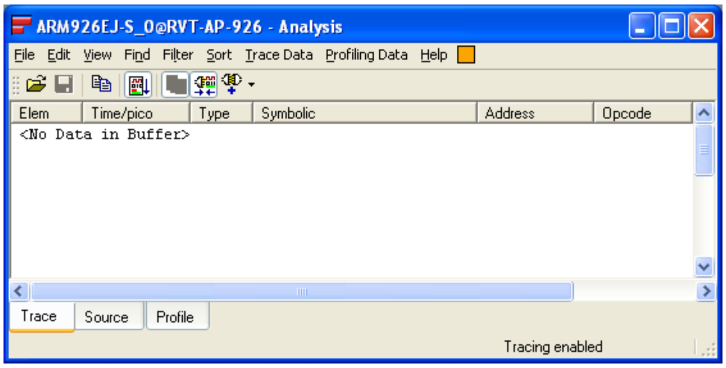
See also:

- *Procedure*.

10.2.1 Procedure

To prepare to start the tutorial, do the following:

1. Make sure that your target supports trace.
2. Set up your trace hardware.
3. Start RealView Debugger and make sure that your trace software is properly configured.
4. Connect to the target:
   a. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
   b. Create a Debug Configuration for the `RealView ICE` Debug Interface, which identifies the targets of your development platform.
   c. Change the Debug Configuration name assigned by RealView Debugger to a more meaningful name.
      This tutorial uses an ARM926EJ-S processor on an ARM Integrator/AP development board, so change the name to **RVT-AP-926**.
   d. Expand the `RVT-AP-926` configuration to show the targets on your development platform.
   e. Double-click on the target that you want to use during the tutorial, to connect to that target (`ARM926EJ-S_0` in this example). Details for the connection are displayed in the Code window.
   f. Select **Close** from the **File** menu to close the Connect to Target window.
5. Connect to the logic analyzer:
   a. Select **Analysis Window** from the **View** menu to display the Analysis window. Figure 10-3 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Elem Time/pico Type Symbolic Address Opcode
Trace Source Profile
Tracing enabled

```
<No Data in Buffer>
```

Figure 10-3 Analysis window

   b. Click **Toggle Analyzer** on the Analysis window toolbar.

6. Load the `trace.axf` image:
   a. Select **Load Image...** from the **Target** menu to display the Select Local File to Load dialog box.
   b. Locate the `trace.axf` image and click on the filename.
   c. Click **Open**. The Select Local File to Load dialog box closes, and the image is loaded. The Code window is updated with details of the image.

See also

- *Requirements for tracing* on page 1-3
- *About the trace hardware setup* on page 2-3
- *About the tracing tutorial* on page 10-2
- Appendix A *Setting up the Trace Hardware*
- Appendix B *Setting up the Trace Software*
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
  - *Changing the name of a Debug Configuration* on page 3-17
  - *Connecting to a target* on page 3-27.

10.3 Running from application reset to a breakpoint

This example demonstrates how to capture trace from the image entry point to the point where execution stops.

See also:

- *Tracing features used*
- *Procedure*
- *Examining the captured trace* on page 10-8.

10.3.1 Tracing features used

The following tracing features are used in this procedure:

- automatic tracing of instructions only is performed by default
- captured trace is displayed when execution stops (in this example, a breakpoint is used to stop execution at a specific location).

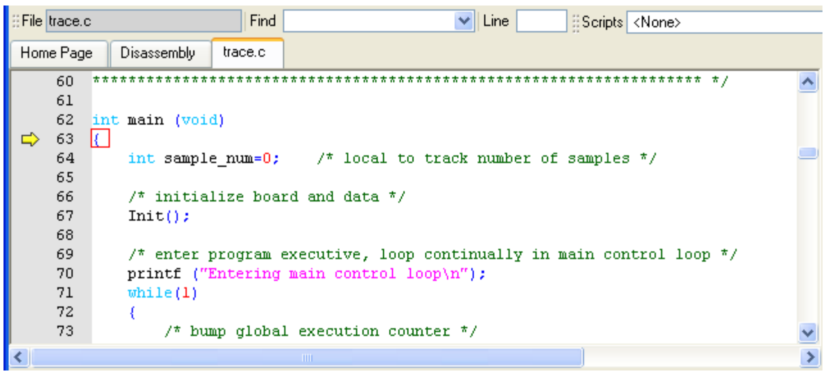
10.3.2 Procedure

To capture trace between the image entry point and a specified location:

1. Prepare to start the tutorial if you have not already done this.

   See *Preparing to start the tracing tutorial* on page 10-5.

2. At the call to `Init()` on line 67, double-click in the gray margin. A simple breakpoint is set on that line. Figure 10-4 shows an example:



```
60  ************************************************************************ */
61
62  int main (void)
63  {
64      int sample_num=0;    /* local to track number of samples */
65
66      /* initialize board and data */
67      Init();
68
69      /* enter program executive, loop continually in main control loop */
70      printf ("Entering main control loop\n");
71      while(1)
72      {
73          /* bump global execution counter */
```

Figure 10-4 Simple breakpoint set to force display of captured trace

RealView Debugger generates a `BREAKINSTRUCTION` CLI command, which is also displayed in the **Cmd** tab of the Output view:

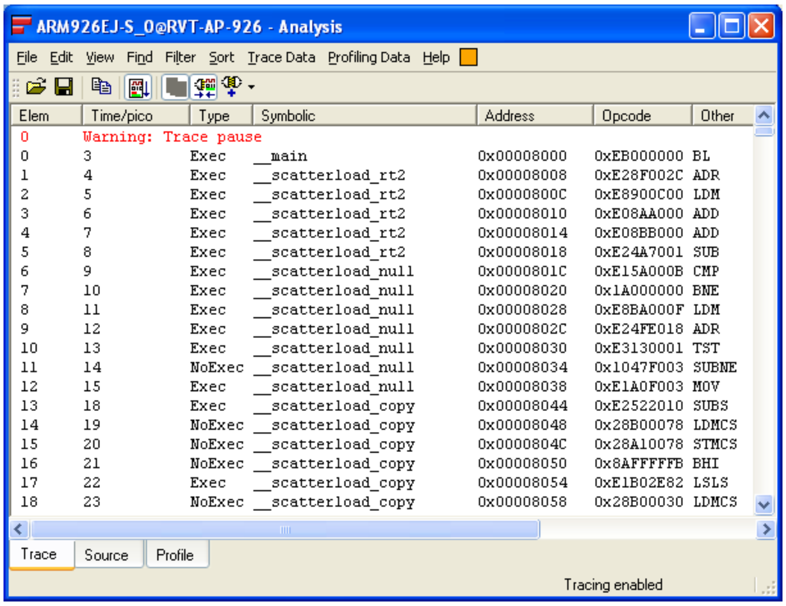
```
breakinstruction,failover \TRACE\#67:0
```

——— **Note** ———

The number at the end of the command is a character position, and depends on the position of the cursor when you double-clicked in the margin.

———————————

3. Click **Run** on the Debug toolbar to start execution. The image runs until the breakpoint is activated. The captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-5 on page 10-8 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem  Time/pico     Type   Symbolic                 Address     Opcode      Other
0     Warning: Trace pause
0     3             Exec   __main                   0x00008000  0xEB000000 BL
1     4             Exec   __scatterload_rt2        0x00008008  0xE28F002C ADR
2     5             Exec   __scatterload_rt2        0x0000800C  0xE8900C00 LDM
3     6             Exec   __scatterload_rt2        0x00008010  0xE08AA000 ADD
4     7             Exec   __scatterload_rt2        0x00008014  0xE08BB000 ADD
5     8             Exec   __scatterload_rt2        0x00008018  0xE24A7001 SUB
6     9             Exec   __scatterload_null       0x0000801C  0xE15A000B CMP
7     10            Exec   __scatterload_null       0x00008020  0x1A000000 BNE
8     11            Exec   __scatterload_null       0x00008028  0xE8BA000F LDM
9     12            Exec   __scatterload_null       0x0000802C  0xE24FE018 ADR
10    13            Exec   __scatterload_null       0x00008030  0xE3130001 TST
11    14            NoExec __scatterload_null       0x00008034  0x1047F003 SUBNE
12    15            Exec   __scatterload_null       0x00008038  0xE1A0F003 MOV
13    18            Exec   __scatterload_copy       0x00008044  0xE2522010 SUBS
14    19            NoExec __scatterload_copy       0x00008048  0x28B00078 LDMCS
15    20            NoExec __scatterload_copy       0x0000804C  0x28A10078 STMCS
16    21            NoExec __scatterload_copy       0x00008050  0x8AFFFFFB BHI
17    22            Exec   __scatterload_copy       0x00008054  0xE1B02E82 LSLS
18    23            NoExec __scatterload_copy       0x00008058  0x28B00030 LDMCS
```

Figure 10-5 Captured trace shown in the Analysis window

Messages similar to the following are also displayed in the **Cmd** tab of the Output view:

```
Stopped at 0x0000823C due to SW Instruction Breakpoint
Stopped at 0x0000823C: TRACE\main Line 67:4
```

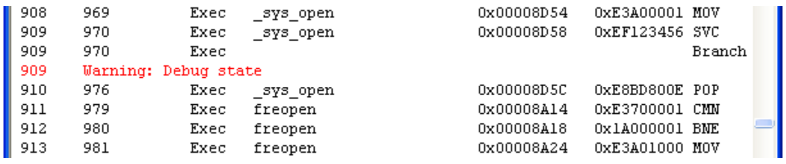
See also

- *Error messages* on page C-2
- *Warning messages* on page C-4
- the following in the *RealView Debugger User Guide*:
  - — Chapter 11 *Setting Breakpoints*
  - — Chapter 12 *Controlling the Behavior of Breakpoints*.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — Chapter 2 *RealView Debugger Commands* for details of the `BREAKINSTRUCTION` command.

10.3.3 Examining the captured trace

Examine the captured trace in the **Trace** tab:

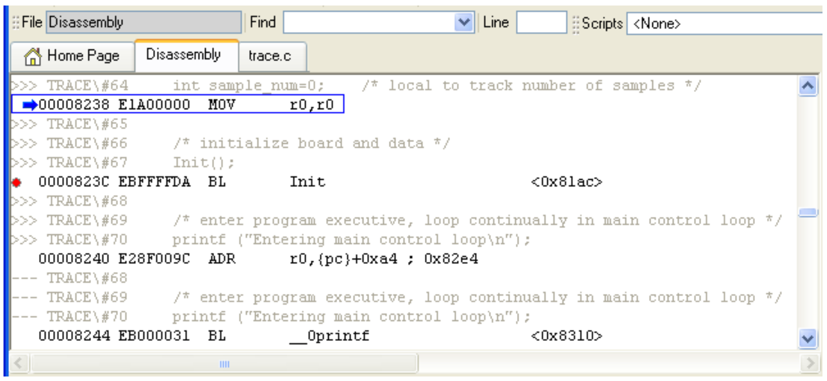
- Tracing starts from the application entry point, `0x8000`. Figure 10-5 shows an example.
- The scatter-loading C library (`__scatterload_*`) runs first.
- Scroll down through the trace until you see the `Warning Debug State` message. Figure 10-6 shows an example. This shows that the debug state is entered for the *Supervisor Call* (SVC).



```
908   969           Exec   _sys_open                0x00008D54  0xE3A00001 MOV
909   970           Exec   _sys_open                0x00008D58  0xEF123456 SVC
909   970           Exec                                                   Branch
909   Warning: Debug state
910   976           Exec   _sys_open                0x00008D5C  0xE8BD800E POP
911   979           Exec   freopen                  0x00008A14  0xE3700001 CMN
912   980           Exec   freopen                  0x00008A18  0x1A000001 BNE
913   981           Exec   freopen                  0x00008A24  0xE3A01000 MOV
```

Figure 10-6 Debug state entered

- Do the following:
  1. Click on a line of captured trace.
  2. Press Ctrl+End to display the end of the captured trace.
  3. Click on the last line of the captured trace.
  4. In the Code window, click the **Disassembly** tab in the Code window. This shows the line of disassembly corresponding to the selected line of the captured trace, indicated by the blue arrow and box. Figure 10-7 shows an example:

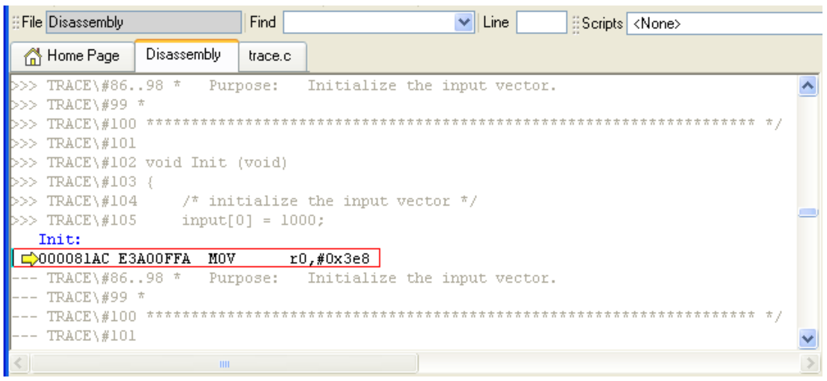


```
>>> TRACE\#64     int sample_num=0;    /* local to track number of samples */
 00008238 E1A00000  MOV      r0,r0
>>> TRACE\#65
>>> TRACE\#66     /* initialize board and data */
>>> TRACE\#67     Init();
 0000823C EBFFFFDA  BL       Init                        <0x81ac>
>>> TRACE\#68
>>> TRACE\#69     /* enter program executive, loop continually in main control loop */
>>> TRACE\#70     printf ("Entering main control loop\n");
 00008240 E28F009C  ADR      r0,{pc}+0xa4 ; 0x82e4
--- TRACE\#68
--- TRACE\#69     /* enter program executive, loop continually in main control loop */
--- TRACE\#70     printf ("Entering main control loop\n");
 00008244 EB000031  BL       __0printf                   <0x8310>
```

Figure 10-7 Tracing stops before the breakpointed instruction

Tracing stops before the instruction at the breakpoint is executed.

  5. Click the **Step** button to step to the next instruction. Observe that execution stops at the first address of the `Init()` function. Figure 10-8 shows an example:



```
>>> TRACE\#86..98 *   Purpose:   Initialize the input vector.
>>> TRACE\#99 *
>>> TRACE\#100 ******************************************************************* */
>>> TRACE\#101
>>> TRACE\#102 void Init (void)
>>> TRACE\#103 {
>>> TRACE\#104      /* initialize the input vector */
>>> TRACE\#105      input[0] = 1000;
   Init:
 000081AC E3A00FFA  MOV      r0,#0x3e8
--- TRACE\#86..98 *   Purpose:   Initialize the input vector.
--- TRACE\#99 *
--- TRACE\#100 ******************************************************************* */
--- TRACE\#101
```

Figure 10-8 Execution stopped in Init() function

- Any warning and error messages are displayed in red text in the Analysis window. Figure 10-6 on page 10-8 shows an example. The following warning messages are usually displayed:

  Warning: Debug State

  This indicates that tracing was suspended for several processor cycles because the processor entered debug state.

  Warning: Trace Pause

  This indicates that tracing was temporarily suspended because of the trace conditions that have been set.

See also

- *Error messages* on page C-2
- *Warning messages* on page C-4

- the following in the *RealView Debugger User Guide*:
  - Chapter 11 *Setting Breakpoints*
  - Chapter 12 *Controlling the Behavior of Breakpoints*.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `BREAKINSTRUCTION` command.

10.4 Running to a second breakpoint

This example demonstrates that trace captured between the points where execution starts and stops replaces previously captured trace. It also shows you how to view the source within the Trace view.

See also:

- *Tracing features used*
- *Procedure*
- *Examining the captured trace* on page 10-13
- *Interleaving source in the Trace view* on page 10-14.

10.4.1 Tracing features used

The following tracing features are used in this procedure:

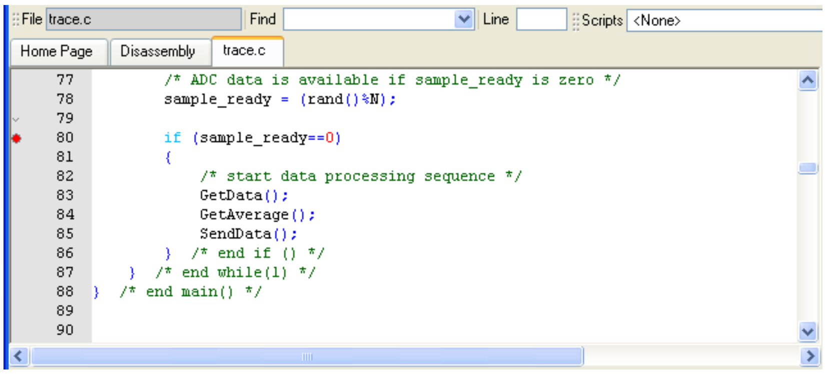
- Automatic tracing of instructions only is performed by default.
- Captured trace is displayed when execution stops. In this example, a second breakpoint is used to stop execution at a different location from the first breakpoint.

  See *Running from application reset to a breakpoint* on page 10-7.
- Source code tracking as you move the cursor through the captured trace.
- Interleaving source in the Trace view.

10.4.2 Procedure

To capture new trace when execution stops at another location:

1. Double-click in the margin at line 80 to set a breakpoint. Figure 10-9 shows an example:



```
77          /* ADC data is available if sample_ready is zero */
78          sample_ready = (rand()%N);
79
80          if (sample_ready==0)
81          {
82              /* start data processing sequence */
83              GetData();
84              GetAverage();
85              SendData();
86          }  /* end if () */
87      }  /* end while(1) */
88  }  /* end main() */
89
90
```

Figure 10-9 Simple breakpoint set to force display of captured trace

RealView Debugger generates a `BREAKINSTRUCTION` CLI command, which is also displayed in the **Cmd** tab of the Output view:

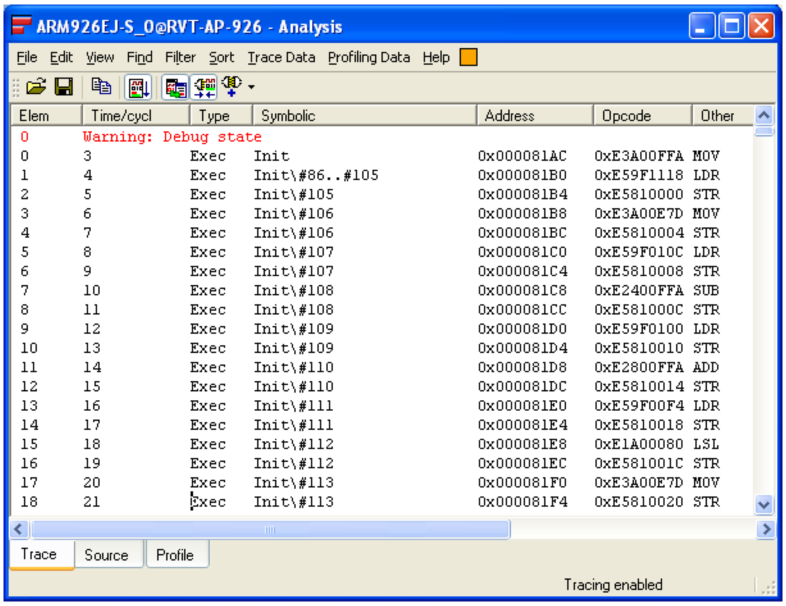
```
breakinstruction,failover \TRACE\#80:0
```

——— **Note** ———

The number at the end of the command is a character position, and depends on the position of the cursor when you double-clicked in the margin.

———————————

2. Click **Run** on the Debug toolbar to restart execution. The image runs until the breakpoint is activated. The captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-10 on page 10-12 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Elem	Time/cycl	Type	Symbolic	Address	Opcode	Other
0	Warning: Debug state					
0	3	Exec	Init	0x000081AC	0xE3A00FFA	MOV
1	4	Exec	Init\#86..#105	0x000081B0	0xE59F1118	LDR
2	5	Exec	Init\#105	0x000081B4	0xE5810000	STR
3	6	Exec	Init\#106	0x000081B8	0xE3A00E7D	MOV
4	7	Exec	Init\#106	0x000081BC	0xE5810004	STR
5	8	Exec	Init\#107	0x000081C0	0xE59F010C	LDR
6	9	Exec	Init\#107	0x000081C4	0xE5810008	STR
7	10	Exec	Init\#108	0x000081C8	0xE2400FFA	SUB
8	11	Exec	Init\#108	0x000081CC	0xE581000C	STR
9	12	Exec	Init\#109	0x000081D0	0xE59F0100	LDR
10	13	Exec	Init\#109	0x000081D4	0xE5810010	STR
11	14	Exec	Init\#110	0x000081D8	0xE2800FFA	ADD
12	15	Exec	Init\#110	0x000081DC	0xE5810014	STR
13	16	Exec	Init\#111	0x000081E0	0xE59F00F4	LDR
14	17	Exec	Init\#111	0x000081E4	0xE5810018	STR
15	18	Exec	Init\#112	0x000081E8	0xE1A00080	LSL
16	19	Exec	Init\#112	0x000081EC	0xE581001C	STR
17	20	Exec	Init\#113	0x000081F0	0xE3A00E7D	MOV
18	21	Exec	Init\#113	0x000081F4	0xE5810020	STR

Figure 10-10 Captured trace shown in the Analysis window

Messages similar to the following are also displayed in the **Cmd** tab of the Output view:

```
Stopped at 0x00008278 due to SW Instruction Breakpoint
Stopped at 0x00008278: TRACE\main Line 80:8
```

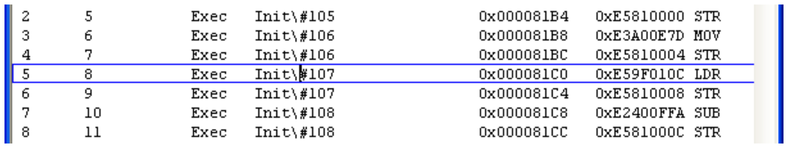
See also

- the following in the *RealView Debugger User Guide*:
  - — Chapter 11 *Setting Breakpoints*
  - — Chapter 12 *Controlling the Behavior of Breakpoints*.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — Chapter 2 *RealView Debugger Commands* for details of the `BREAKINSTRUCTION` command.

10.4.3 Examining the captured trace

Examine the captured trace in the **Trace** tab:

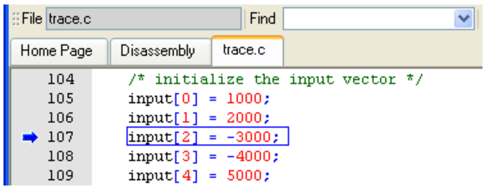
- The previously captured trace is lost and replaced with the newly captured trace.
- Observe how source code is tracked when a sample of trace is selected:
  1. Click on a trace sample for the `Init` procedure, for example, the first `Init\#107` sample. Figure 10-11 shows an example:



```
2   5    Exec  Init\#105   0x000081B4  0xE5810000 STR
3   6    Exec  Init\#106   0x000081B8  0xE3A00E7D MOV
4   7    Exec  Init\#106   0x000081BC  0xE5810004 STR
5   8    Exec  Init\#107   0x000081C0  0xE59F010C LDR
6   9    Exec  Init\#107   0x000081C4  0xE5810008 STR
7   10   Exec  Init\#108   0x000081C8  0xE2400FFA SUB
8   11   Exec  Init\#108   0x000081CC  0xE581000C STR
```

Figure 10-11 Select trace sample

  2. In the Code window, click the **trace.c** tab in the Code window. The source line that corresponds to the selected trace sample is highlighted. Figure 10-12 shows an example:



```
104      /* initialize the input vector */
105      input[0] = 1000;
106      input[1] = 2000;
107      input[2] = -3000;
108      input[3] = -4000;
109      input[4] = 5000;
```

Figure 10-12 Source line corresponding to select trace sample

     A message similar to the following is also displayed in the **Cmd** tab of the Output view:

     `Scoped to: (0x000081C0): TRACE\Init Line 107:4`

  3. Move the Analysis window so that it does not overlay the source view in the Code window.
  4. Click on another trace sample in the **Trace** tab. The corresponding source line is highlighted.
  5. Press the Up and Down arrow keys as required to see the source code tracking in action.
  6. In the Code window, click the **Disassembly** tab to display the disassembly view.
  7. Repeat steps 4 and 5 to observe the source code tracking in the disassembly view. The memory address highlighted in the disassembly view matches the address for the selected trace sample.

See also

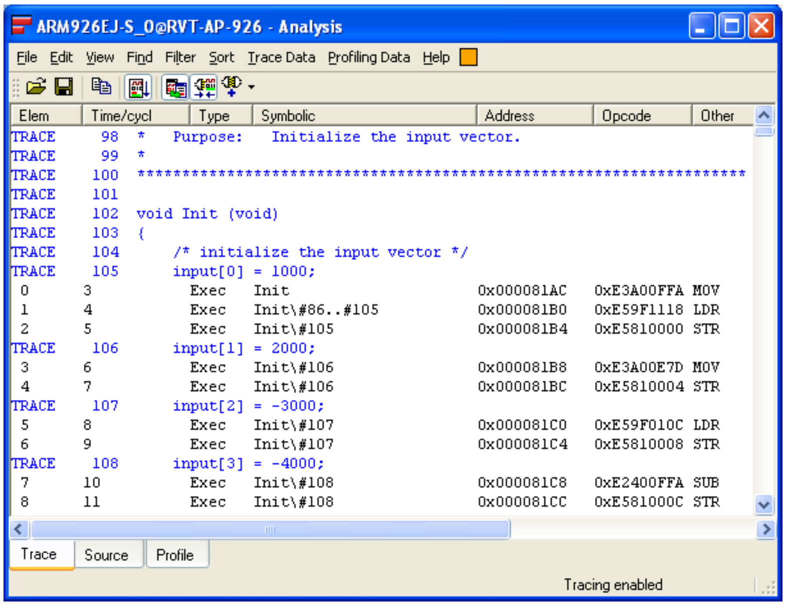
- the following in the *RealView Debugger User Guide*:
  - Chapter 11 *Setting Breakpoints*
  - Chapter 12 *Controlling the Behavior of Breakpoints*.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `BREAKINSTRUCTION` command.

10.4.4 Interleaving source in the Trace view

You might want to see the source code that corresponds to the captured trace. Although you can view this in the **Source** tab of the Analysis window, the Source view does not show you the captured trace. If you want to see the source and the captured trace together, then you can interleave the source in the Trace view.

To interleave the source in the Trace view:

1. Select **Interleaved Source** from the **Trace Data** menu in the Analysis window. The trace shows the interleaved source corresponding to the captured trace. Figure 10-13 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem   Time/cycl  Type   Symbolic                           Address      Opcode     Other
TRACE     98   *    Purpose:    Initialize the input vector.
TRACE     99   *
TRACE    100   ********************************************************************
TRACE    101
TRACE    102   void Init (void)
TRACE    103   {
TRACE    104       /* initialize the input vector */
TRACE    105       input[0] = 1000;
 0       3           Exec    Init                         0x000081AC    0xE3A00FFA MOV
 1       4           Exec    Init\#86..#105               0x000081B0    0xE59F1118 LDR
 2       5           Exec    Init\#105                    0x000081B4    0xE5810000 STR
TRACE    106       input[1] = 2000;
 3       6           Exec    Init\#106                    0x000081B8    0xE3A00E7D MOV
 4       7           Exec    Init\#106                    0x000081BC    0xE5810004 STR
TRACE    107       input[2] = -3000;
 5       8           Exec    Init\#107                    0x000081C0    0xE59F010C LDR
 6       9           Exec    Init\#107                    0x000081C4    0xE5810008 STR
TRACE    108       input[3] = -4000;
 7       10          Exec    Init\#108                    0x000081C8    0xE2400FFA SUB
 8       11          Exec    Init\#108                    0x000081CC    0xE581000C STR
```

Figure 10-13 Interleaved source in the Trace view

2. Scroll down through the trace to see more examples of the interleaved source.

3. Select **Interleaved Source** from the **Trace Data** menu in the Analysis window. The interleaved source is hidden.

See also

- *Interleaving source with the trace output* on page 9-7
- the following in the *RealView Debugger User Guide*:
  - — Chapter 11 *Setting Breakpoints*
  - — Chapter 12 *Controlling the Behavior of Breakpoints*.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — Chapter 2 *RealView Debugger Commands* for details of the `BREAKINSTRUCTION` command.

10.5 Using a trace start point

The captured trace in the previous demonstrations included the startup code. However, you can bypass this startup code and begin tracing from a specific location.

See also:

- *Tracing features used*
- *Procedure*
- *Examining the captured trace* on page 10-17.

10.5.1 Tracing features used

The following tracing features are used in this procedure:

- Use of a Trace Start Point to specify the location where trace capture is to begin. This turns off automatic tracing.

10.5.2 Procedure

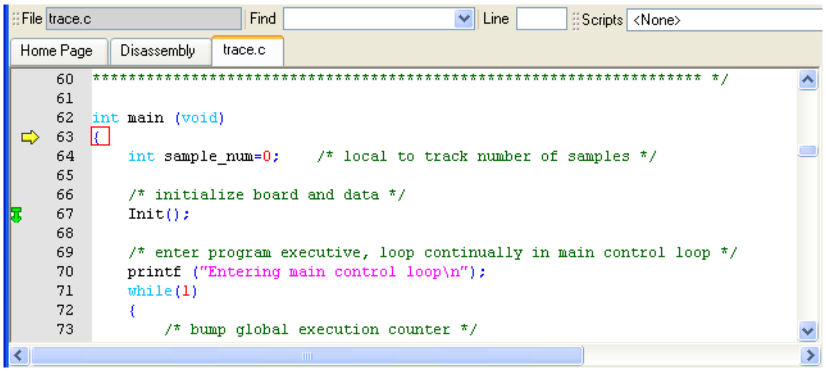
To capture trace from a specific location:

1. Reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Reload** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`).

   Note

   The reload maintains the breakpoints that were previously set at line 67 and line 80.

2. Remove the first breakpoint:
   a. Click the **trace.c** tab to display the source code.
   b. Right-click on the red breakpoint icon at line 67 to display the context menu.
   c. Select **Remove Breakpoint** from the context menu. The breakpoint is removed.
3. Set a Trace Start Point at line 67:
   a. Right-click on the gray margin at line 67 to display the context menu.
   b. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.
   c. Select **Trace Start Point** from the list of tracepoint types.
   d. Click **OK**. The dialog box closes, and the Trace Start Point is set as indicated by the green arrow. Figure 10-14 on page 10-16 shows an example:



```
60  ********************************************************************* */
61
62  int main (void)
63  {
64      int sample_num=0;    /* local to track number of samples */
65
66      /* initialize board and data */
67      Init();
68
69      /* enter program executive, loop continually in main control loop */
70      printf ("Entering main control loop\n");
71      while(1)
72      {
73          /* bump global execution counter */
```

Figure 10-14 Trace Start Point set

RealView Debugger generates a `TRACE` CLI command, which is also displayed in the **Cmd** tab of the Output view:

```
trace,prompt \TRACE\#67:0
```

——— **Note** ———

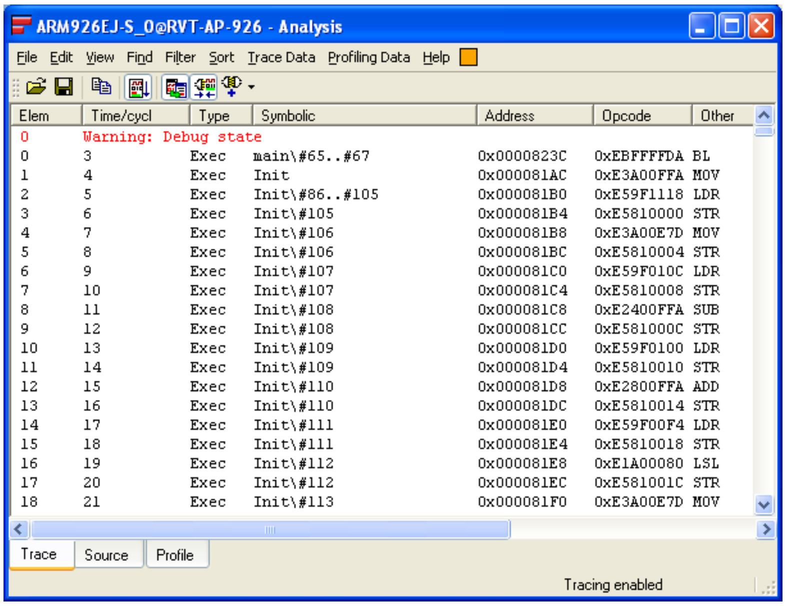
The number at the end of the command is a character position, and depends on the position of the mouse pointer when you displayed the context menu.

———————————

4. Click **Run** on the Debug toolbar to start execution. The image runs until the breakpoint at line 80 is activated. Messages similar to the following are also displayed in the **Cmd** tab of the Output view:

```
Stopped at 0x00008278 due to SW Instruction Breakpoint
Stopped at 0x00008278: TRACE\main Line 80:8
```

The captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-15 shows an example:



ARM926EJ-S_0@RVT-AP-926 - Analysis

Elem	Time/cycl	Type	Symbolic	Address	Opcode	Other
0	Warning: Debug state					
0	3	Exec	main\#65..#67	0x0000823C	0xEBFFFFDA	BL
1	4	Exec	Init	0x000081AC	0xE3A00FFA	MOV
2	5	Exec	Init\#86..#105	0x000081B0	0xE59F1118	LDR
3	6	Exec	Init\#105	0x000081B4	0xE5810000	STR
4	7	Exec	Init\#106	0x000081B8	0xE3A00E7D	MOV
5	8	Exec	Init\#106	0x000081BC	0xE5810004	STR
6	9	Exec	Init\#107	0x000081C0	0xE59F010C	LDR
7	10	Exec	Init\#107	0x000081C4	0xE5810008	STR
8	11	Exec	Init\#108	0x000081C8	0xE2400FFA	SUB
9	12	Exec	Init\#108	0x000081CC	0xE581000C	STR
10	13	Exec	Init\#109	0x000081D0	0xE59F0100	LDR
11	14	Exec	Init\#109	0x000081D4	0xE5810010	STR
12	15	Exec	Init\#110	0x000081D8	0xE2800FFA	ADD
13	16	Exec	Init\#110	0x000081DC	0xE5810014	STR
14	17	Exec	Init\#111	0x000081E0	0xE59F00F4	LDR
15	18	Exec	Init\#111	0x000081E4	0xE5810018	STR
16	19	Exec	Init\#112	0x000081E8	0xE1A00080	LSL
17	20	Exec	Init\#112	0x000081EC	0xE581001C	STR
18	21	Exec	Init\#113	0x000081F0	0xE3A00E7D	MOV

Tracing enabled

Figure 10-15 Trace capture from a Trace Start Point

See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 11 *Setting Breakpoints*

  - Chapter 12 *Controlling the Behavior of Breakpoints*.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `TRACE` command.

10.5.3 Examining the captured trace

Examine the captured trace in the **Trace** tab:

- Tracing begins at the location of the Trace Start Point.
- All the C library code is excluded from the trace capture.
- Except for the first element, trace capture is identical to the captured trace in *Running to a second breakpoint* on page 10-11. The difference is because of the step operation performed in *Examining the captured trace* on page 10-8.

10.6 Using trace start and end points to trace and time a function

Sometimes you might want to time the execution of a function. Therefore, you want to limit the trace capture to a specific function.

See also:

- *Tracing features used*
- *Procedure*
- *Examining the captured trace* on page 10-20
- *Considerations when performing timing measurements* on page 10-22.

10.6.1 Tracing features used

The following tracing features are used in this procedure:

- a Trace Start Point to specify the location where trace capture is to begin
- a Trace End Point to specify the location where trace capture is to stop
- cycle accurate tracing

  Note

  The availability of cycle accurate tracing depends on the ETM version you are using.

- calculating the time difference between two elements in the captured trace.

The Trace End Point does not cause captured trace to be displayed. Therefore, execution must stop to force the display of the captured trace. In this example, you manually stop execution.

The display of discontinuities in the captured trace are identified by the message `Warning: Trace Pause`. This message shows where instruction executions or data accesses occur, but are not captured because of the current trace conditions.

10.6.2 Procedure

To limit the captured trace to a specific function:

1. Remove the tracepoint and breakpoint you set previously. To do this, select **Debug → Breakpoints → Clear All Break/Tracepoints** from the Code window main menu.
2. Reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Reload** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`).
3. Enable cycle accurate tracing for the ETM:
   a. Select **Configure Analyzer Properties...** from the **Edit** menu of the Analysis window to display the Configure ETM dialog box.
   b. Select **Cycle accurate tracing**.
   c. Click **OK** to close the Configure ETM dialog box. RealView Debugger generates an `ETM_CONFIG` CLI command, which is also displayed in the **Cmd** tab of the Output view:

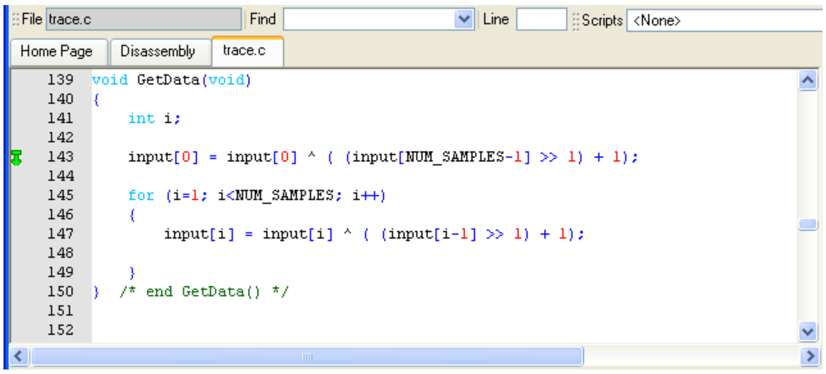
      ```
      etm_config,syncfreq:0,packauto,cycle
      ```
4. Set a Trace Start Point at the start of `GetData()`:
   a. Click the **trace.c** tab.

b. Right-click in the gray margin at line 143 to display the context menu.

c. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

d. Select **Trace Start Point** from the list of tracepoint types.

e. Click **OK**. The dialog box closes, and the Trace Start Point is set as indicated by the green arrow . Figure 10-16 shows an example:



```
139 void GetData(void)
140 {
141     int i;
142
143     input[0] = input[0] ^ ( (input[NUM_SAMPLES-1] >> 1) + 1);
144
145     for (i=1; i<NUM_SAMPLES; i++)
146     {
147         input[i] = input[i] ^ ( (input[i-1] >> 1) + 1);
148
149     }
150 }  /* end GetData() */
151
152
```

Figure 10-16 Trace Start Point set at start of a function

RealView Debugger generates a `TRACE` CLI command, which is also displayed in the **Cmd** tab of the Output view:

```
trace,prompt \TRACE\#143:0
```

Note

The number at the end of the command is a character position, and depends on the position of the mouse pointer when you displayed the context menu.

---

5. Set a Trace End Point at the end of `GetData()`:

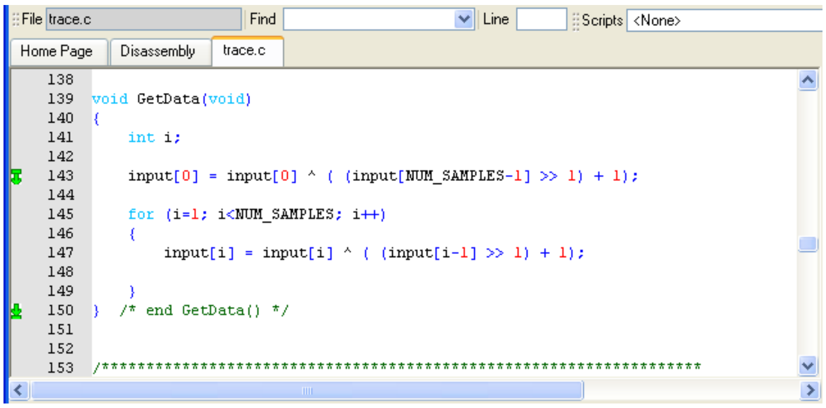
a. Click the **trace.c** tab.

b. Right-click in the gray margin at line 150 to display the context menu.

c. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

d. Select **Trace End Point** from the list of tracepoint types.

e. Click **OK**. The dialog box closes, and the Trace End Point is set as indicated by the green arrow . Figure 10-17 shows an example:



```
138
139 void GetData(void)
140 {
141     int i;
142
143     input[0] = input[0] ^ ( (input[NUM_SAMPLES-1] >> 1) + 1);
144
145     for (i=1; i<NUM_SAMPLES; i++)
146     {
147         input[i] = input[i] ^ ( (input[i-1] >> 1) + 1);
148
149     }
150 }  /* end GetData() */
151
152
153 /*********************************************************************
```

Figure 10-17 Trace End Point set at end of a function

RealView Debugger generates a `TRACE` CLI command, which is also displayed in the **Cmd** tab of the Output view:

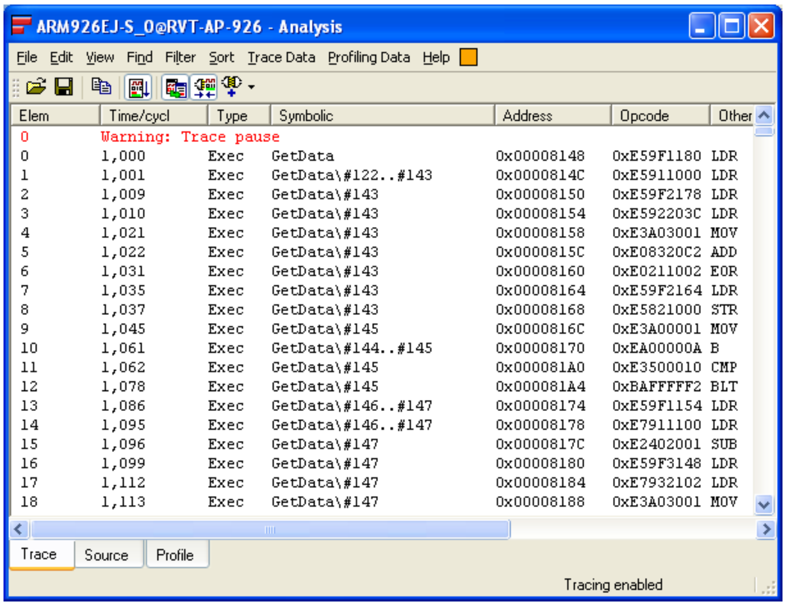
```
trace,prompt \TRACE\#150:0
```

Note

The number at the end of the command is a character position, and depends on the position of the mouse pointer when you displayed the context menu.

6. Click **Run** on the Debug toolbar to start execution. Captured trace might be displayed in the Analysis window before you perform the next step.

7. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-18 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem    Time/cycl   Type   Symbolic                 Address      Opcode      Other
0       Warning: Trace pause
0       1,000       Exec   GetData                  0x00008148   0xE59F1180 LDR
1       1,001       Exec   GetData\#122..#143       0x0000814C   0xE5911000 LDR
2       1,009       Exec   GetData\#143             0x00008150   0xE59F2178 LDR
3       1,010       Exec   GetData\#143             0x00008154   0xE592203C LDR
4       1,021       Exec   GetData\#143             0x00008158   0xE3A03001 MOV
5       1,022       Exec   GetData\#143             0x0000815C   0xE08320C2 ADD
6       1,031       Exec   GetData\#143             0x00008160   0xE0211002 EOR
7       1,035       Exec   GetData\#143             0x00008164   0xE59F2164 LDR
8       1,037       Exec   GetData\#143             0x00008168   0xE5821000 STR
9       1,045       Exec   GetData\#145             0x0000816C   0xE3A00001 MOV
10      1,061       Exec   GetData\#144..#145       0x00008170   0xEA00000A B
11      1,062       Exec   GetData\#145             0x000081A0   0xE3500010 CMP
12      1,078       Exec   GetData\#145             0x000081A4   0xBAFFFFF2 BLT
13      1,086       Exec   GetData\#146..#147       0x00008174   0xE59F1154 LDR
14      1,095       Exec   GetData\#146..#147       0x00008178   0xE7911100 LDR
15      1,096       Exec   GetData\#147             0x0000817C   0xE2402001 SUB
16      1,099       Exec   GetData\#147             0x00008180   0xE59F3148 LDR
17      1,112       Exec   GetData\#147             0x00008184   0xE7932102 LDR
18      1,113       Exec   GetData\#147             0x00008188   0xE3A03001 MOV
```

Figure 10-18 Captured trace

See also

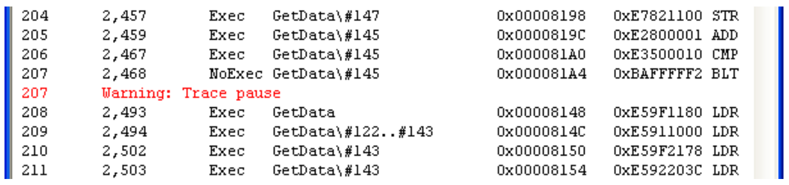
- *Displaying captured trace on a stop condition* on page 2-21
- *Cycle accurate tracing* on page 4-11
- *Trace Start Point and Trace End Point* on page 5-4
- *Calculating the time between two points* on page 9-12
- *Changing the default format of time information* on page 9-13
- *Specifying the processor speed for cycle/time computations* on page 9-14
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `ETM_CONFIG` and `TRACE` commands.

10.6.3 Examining the captured trace

Examine the captured trace in the **Trace** tab:

- Trace begins at the Trace Start Point (at line 142). The `Time/cycl` column shows the time of each trace sample in cycles, which is the default for cycle accurate tracing.

- Scroll down to the Trace End Point. This is indicated by the break in trace capture identified by the message `Warning: Trace Pause`. Figure 10-19 shows an example:
  - — The Trace End Point is the element above this message.
  - — The element below this message shows that tracing starts again at the Trace Start Point.



```
204     2,457     Exec    GetData\#147           0x00008198   0xE7821100 STR
205     2,459     Exec    GetData\#145           0x0000819C   0xE2800001 ADD
206     2,467     Exec    GetData\#145           0x000081A0   0xE3500010 CMP
207     2,468     NoExec  GetData\#145           0x000081A4   0xBAFFFFF2 BLT
207     Warning: Trace pause
208     2,493     Exec    GetData                0x00008148   0xE59F1180 LDR
209     2,494     Exec    GetData\#122..#143     0x0000814C   0xE5911000 LDR
210     2,502     Exec    GetData\#143           0x00008150   0xE59F2178 LDR
211     2,503     Exec    GetData\#143           0x00008154   0xE592203C LDR
```

Figure 10-19 Trace discontinuity

- Scroll through the captured trace to see that only `GetData()` is traced.
- Calculate the execution time in cycles for `GetData()`:
  1. Click on the line immediately below the message `Warning: Trace Pause`, which is at a Trace Start Point. This is to be the first point for the timing measurement.
  2. Scroll down to the next trace pause message using the scroll bar to the right of the Analysis window.

     Note

     Do not click in the **Trace** tab, otherwise the start point for the timing measurement is no longer selected.

  3. Right-click on the line with an element number that is one less than the element number for the warning message.
  4. Select **Time Measured from Selected...** from the context menu to display the Prompt dialog box. This shows the execution time of the function in cycles. It also shows which elements in the captured trace are used for the calculations.

     You can compute this manually by subtracting the cycle counts of the respective elements in the `Time/cycl` column of the **Trace** tab.
  5. Click **OK** to close the Prompt dialog box.
- For the same trace samples you used to calculate the execution time in cycles, calculate the execution time using a different timing unit and the correct clock speed for your processor:
  1. Specify an appropriate timing unit for your processor:
     a. Select **Scale Time Units...** from the **View** menu of the Analysis window to display the time scale dialog box.

     b. Select the appropriate unit (typically, microseconds or milliseconds). For this example, select **Microseconds**.

     c. Click **OK** to close the dialog box. The `Time/cycl` column title changes to `Time/usec`, and the timing values in the column change accordingly. Also, the trace display changes to show the previously selected trace sample.

     Note

     If you have clicked in the **Trace** tab since calculating the execution time in cycles, then a different trace sample might be selected. Click on the line immediately after the first `Warning: Trace Pause` message.

2. Specify the processor clock speed:
   a. Select **Define Processor Speed for Scaling...** from the **View** menu of the Analysis window to display the Processor Clock Speed dialog box.
   b. Enter the clock speed in MHz for your processor.

      ——— **Note** ———

      By default, a clock frequency of 20 MHz is used for the calculations.

      ———————————
   c. Click **OK** to close the dialog box. The timing values in the `Time/usec` column change to reflect the new clock speed.
3. Scroll down using the scroll bar to the right of the Analysis window until the next `Warning: Trace Pause` message is visible.
4. Right-click on the line with an element number that is one less than the element number for the warning message.
5. Select **Time Measured from Selected...** from the context menu to display the Prompt dialog box. This shows the absolute execution time of the function in microseconds (equivalent to the previously measured time in cycles).

   You can compute this manually by subtracting the timing values of the respective elements in the `Time/usec` column of the **Trace** tab.
6. Click **OK** to close the Prompt dialog box.

See also

- *Displaying captured trace on a stop condition* on page 2-21
- *Cycle accurate tracing* on page 4-11
- *Trace Start Point and Trace End Point* on page 5-4
- *Calculating the time between two points* on page 9-12
- *Changing the default format of time information* on page 9-13
- *Specifying the processor speed for cycle/time computations* on page 9-14
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `ETM_CONFIG` and `TRACE` commands.

10.6.4 Considerations when performing timing measurements

Although this is a good technique for calculating the execution time for specific functions:

- be aware that different calls to a function might give different time measurements
- it includes any interrupts and task switches that occur between the trace start and stop points.

See also

- *Displaying captured trace on a stop condition* on page 2-21
- *Cycle accurate tracing* on page 4-11
- *Trace Start Point and Trace End Point* on page 5-4
- *Calculating the time between two points* on page 9-12

- *Changing the default format of time information* on page 9-13
- *Specifying the processor speed for cycle/time computations* on page 9-14
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `ETM_CONFIG` and `TRACE` commands.

10.7 Comparing trace start and end points with a trace range

Although you can define a region where trace is to be captured using trace start and end points or a trace range, they behave differently. For example, called functions are not traced within a trace range, but they are traced between trace start and end points.

See also:

- *Tracing features used*
- *Tracing with trace start and end points*
- *Examining trace captured with trace start and end points* on page 10-27
- *Tracing with a trace range* on page 10-28
- *Examining trace captured with a trace range* on page 10-31.

10.7.1 Tracing features used

The following tracing features are used in this procedure:

- a Trace Start Point to specify the location where trace capture is to begin
- a Trace End Point to specify the location where trace capture is to stop
- searching for a function and an address
- a Start of Trace Range to specify the location where trace capture is to begin
- an End of Trace Range to specify the location where trace capture is to stop
- source code tracking as you move the cursor through the captured trace.

Discontinuities in the captured trace are identified by the message `Warning: Trace Pause`. This message shows where instruction executions or data accesses occur, but are not captured because of the current trace conditions.

10.7.2 Tracing with trace start and end points

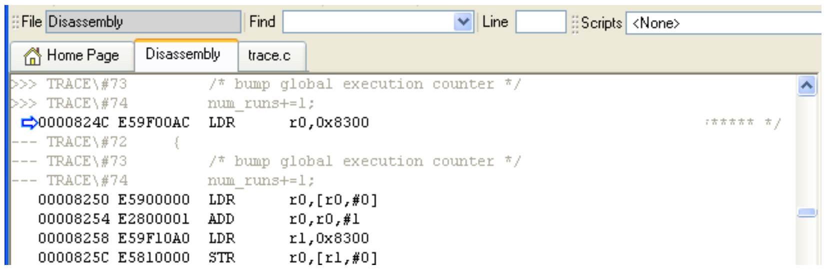
To capture trace using trace start and end points:

1. Remove the trace start and end points and the breakpoint you set previously:
    a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
    b. Select **Unload** from the context menu. The image is unloaded.
    c. Right-click on the `Load Not Loaded` entry in the Process Control view to display the context menu.
    d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The tracepoints are removed.
2. Disable cycle accurate tracing for the ETM:
    a. Select **Configure Analyzer Properties...** from the **Edit** menu of the Analysis window to display the Configure ETM dialog box.
    b. Deselect the **Cycle accurate tracing** check box.
    c. Click **OK** to close the Configure ETM dialog box. RealView Debugger generates an `ETM_CONFIG` CLI command, which is also displayed in the **Cmd** tab of the Output view:

    ```
    etm_config,syncfreq:0,packauto
    ```
3. Set a Trace Start Point at the address for line 74 in `trace.c`:
    a. In the Code window, click the **trace.c** tab.
    b. Right-click on line 74 in the **trace.c** tab to display the context menu.

c. Select **Locate Disassembly** from the context menu to display the Prompt dialog box.

d. The **Disassembly** tab is displayed, and the corresponding address is indicated by a hollow blue arrow. Figure 10-20 shows an example:



```
>>> TRACE\#73          /* bump global execution counter */
>>> TRACE\#74          num_runs+=1;
 ⇨0000824C E59F00AC  LDR      r0,0x8300                                  ;***** */
--- TRACE\#72      {
--- TRACE\#73          /* bump global execution counter */
--- TRACE\#74          num_runs+=1;
   00008250 E5900000  LDR      r0,[r0,#0]
   00008254 E2800001  ADD      r0,r0,#1
   00008258 E59F10A0  LDR      r1,0x8300
   0000825C E5810000  STR      r0,[r1,#0]
```

Figure 10-20 Disassembly view

RealView Debugger generates a `DISASSEMBLE` CLI command, which is also displayed in the **Cmd** tab of the Output view, for example:

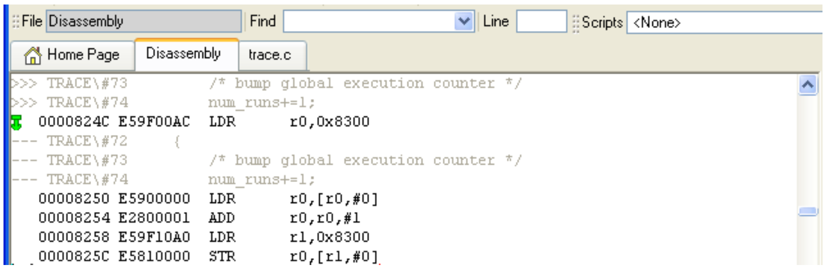
```
dis \TRACE\#74:12
```

e. Right-click in the margin at indicated address to display the context menu.

f. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

g. Select **Trace Start Point** from the list of tracepoint types.

h. Click **OK**. The dialog box closes, and the Trace Start Point is set as indicated by the green arrow. Figure 10-21 shows an example:



```
>>> TRACE\#73          /* bump global execution counter */
>>> TRACE\#74          num_runs+=1;
   0000824C E59F00AC  LDR      r0,0x8300
--- TRACE\#72      {
--- TRACE\#73          /* bump global execution counter */
--- TRACE\#74          num_runs+=1;
   00008250 E5900000  LDR      r0,[r0,#0]
   00008254 E2800001  ADD      r0,r0,#1
   00008258 E59F10A0  LDR      r1,0x8300
   0000825C E5810000  STR      r0,[r1,#0]
```

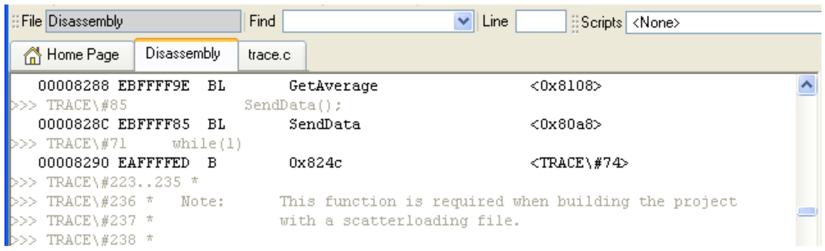
Figure 10-21 Trace Start Point set

RealView Debugger generates a `TRACE` CLI command, which is also displayed in the **Cmd** tab of the Output view, for example:

```
trace,prompt 0x0000824C
```

4. Set a Trace End Point at the address of the first instance of `while(1)`:

a. Scroll down the **Disassembly** tab until the first instance of `while(1)` is visible. Figure 10-22 shows an example:



```
   00008288 EBFFFF9E  BL       GetAverage                <0x8108>
>>> TRACE\#85              SendData();
   0000828C EBFFFF85  BL       SendData                  <0x80a8>
>>> TRACE\#71      while(1)
   00008290 EAFFFFED  B        0x824c                    <TRACE\#74>
>>> TRACE\#223..235 *
>>> TRACE\#236 *   Note:      This function is required when building the project
>>> TRACE\#237 *              with a scatterloading file.
>>> TRACE\#238 *
```

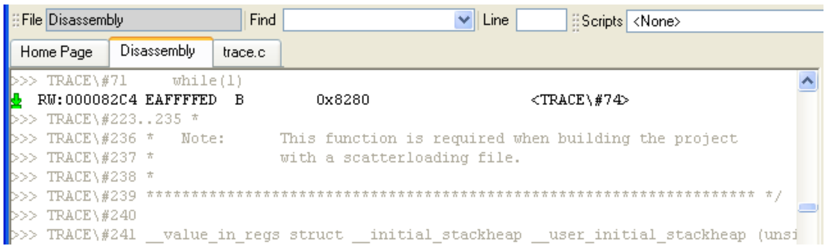
Figure 10-22 Disassembly view

b. Right-click in the margin at the address corresponding to `while(1)` to display the context menu.

c. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

d. Select **Trace End Point** from the list of tracepoint types.

e. Click **OK**. The dialog box closes, and the Trace End Point is set as indicated by the green arrow. Figure 10-23 shows an example:



```
>>> TRACE\#71      while(1)
  RW:000082C4 EAFFFED  B        0x8280                  <TRACE\#74>
>>> TRACE\#223..235 *
>>> TRACE\#236 *   Note:      This function is required when building the project
>>> TRACE\#237 *              with a scatterloading file.
>>> TRACE\#238 *
>>> TRACE\#239 ******************************************************************** */
>>> TRACE\#240
>>> TRACE\#241 __value_in_regs struct __initial_stackheap __user_initial_stackheap (unsi
```

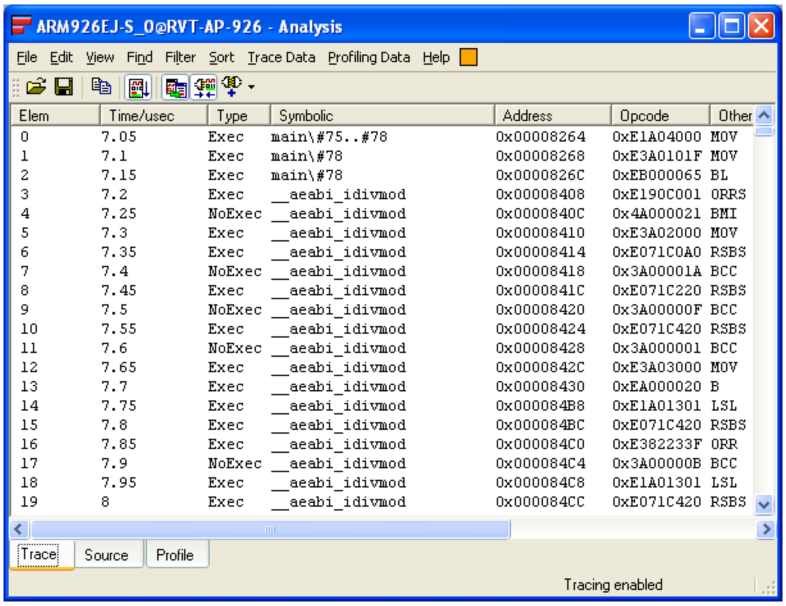
Figure 10-23 Trace End Point set

RealView Debugger generates a TRACE CLI command, which is also displayed in the **Cmd** tab of the Output view, for example:

```
trace,prompt 0x00008290
```

5. Click **Run** on the Debug toolbar to start execution.

6. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-24 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem    Time/usec  Type    Symbolic            Address     Opcode     Other
0       7.05       Exec    main\#75..#78       0x00008264  0xE1A04000 MOV
1       7.1        Exec    main\#78            0x00008268  0xE3A0101F MOV
2       7.15       Exec    main\#78            0x0000826C  0xEB000065 BL
3       7.2        Exec    __aeabi_idivmod     0x00008408  0xE190C001 ORRS
4       7.25       NoExec  __aeabi_idivmod     0x0000840C  0x4A000021 BMI
5       7.3        Exec    __aeabi_idivmod     0x00008410  0xE3A02000 MOV
6       7.35       Exec    __aeabi_idivmod     0x00008414  0xE071C0A0 RSBS
7       7.4        NoExec  __aeabi_idivmod     0x00008418  0x3A00001A BCC
8       7.45       Exec    __aeabi_idivmod     0x0000841C  0xE071C220 RSBS
9       7.5        NoExec  __aeabi_idivmod     0x00008420  0x3A00000F BCC
10      7.55       Exec    __aeabi_idivmod     0x00008424  0xE071C420 RSBS
11      7.6        NoExec  __aeabi_idivmod     0x00008428  0x3A000001 BCC
12      7.65       Exec    __aeabi_idivmod     0x0000842C  0xE3A03000 MOV
13      7.7        Exec    __aeabi_idivmod     0x00008430  0xEA000020 B
14      7.75       Exec    __aeabi_idivmod     0x000084B8  0xE1A01301 LSL
15      7.8        Exec    __aeabi_idivmod     0x000084BC  0xE071C420 RSBS
16      7.85       Exec    __aeabi_idivmod     0x000084C0  0xE382233F ORR
17      7.9        NoExec  __aeabi_idivmod     0x000084C4  0x3A00000B BCC
18      7.95       Exec    __aeabi_idivmod     0x000084C8  0xE1A01301 LSL
19      8          Exec    __aeabi_idivmod     0x000084CC  0xE071C420 RSBS
```

Figure 10-24 Captured trace in Analysis window

See also

- *Displaying captured trace on a stop condition* on page 2-21
- *Trace Start Point and Trace End Point* on page 5-4
- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Error messages* on page C-2

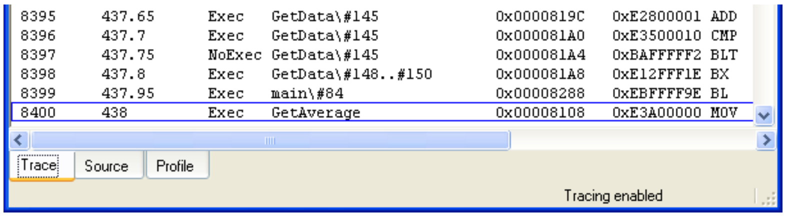
- *Warning messages* on page C-4
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — Chapter 2 *RealView Debugger Commands* for details of the `DISASSEMBLE`, `ETM_CONFIG`, and `TRACE` commands.

10.7.3 Examining trace captured with trace start and end points

Examine the captured trace in the **Trace** tab:

- Observe that between trace start and end points:
  - — the captured trace includes the tracing of called functions
  - — there are no discontinuities after the trace capture begins.
- Display the source analysis view for the captured trace:
  1. Click the **Source** tab.
  2. Scroll through the source analysis view.

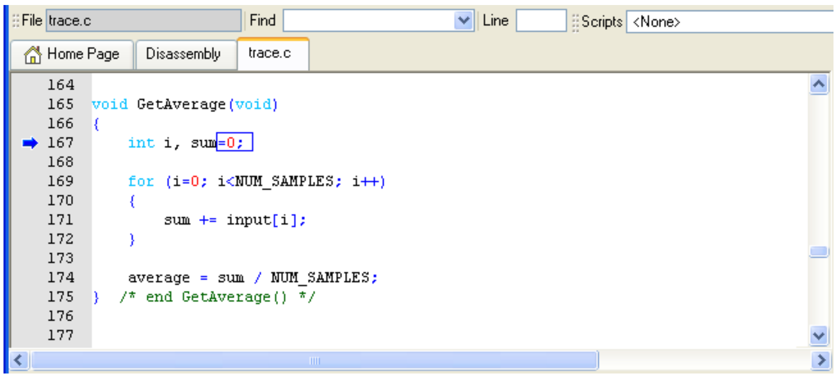
  Observe that program flow does not go into `if (sample_ready == 0)` very often. Also, the same four or five lines of C appear, waiting for `sample_ready` to equal zero.
- Search the captured trace for functions:
  1. Click the **Trace** tab to display the captured trace.
  2. Select **Find Symbol Name...** from the **Find** menu to display the Enter Value dialog box.
  3. Enter the name of a function that is called within the `while(1)` loop, without the parentheses. For example, enter **GetAverage**.
  4. Click **Find**. The first instance of `GetAverage` in the captured trace is selected. Figure 10-25 shows an example:



```
8395    437.65    Exec    GetData\#145          0x0000819C  0xE2800001 ADD
8396    437.7     Exec    GetData\#145          0x000081A0  0xE3500010 CMP
8397    437.75    NoExec  GetData\#145          0x000081A4  0xBAFFFFF2 BLT
8398    437.8     Exec    GetData\#148..#150    0x000081A8  0xE12FFF1E BX
8399    437.95    Exec    main\#84              0x00008288  0xEBFFFF9E BL
8400    438       Exec    GetAverage            0x00008108  0xE3A00000 MOV
```

Figure 10-25 Searched function in Trace view

  5. Click on the first instance of `GetAverage`.
  6. Click the **trace.c** tab in the Code window. The current context is at the entry point to `GetAverage`. Figure 10-26 shows an example:



```
void GetAverage(void)
{
    int i, sum=0;

    for (i=0; i<NUM_SAMPLES; i++)
    {
        sum += input[i];
    }

    average = sum / NUM_SAMPLES;
}  /* end GetAverage() */
```

Figure 10-26 Source code corresponding to searched function

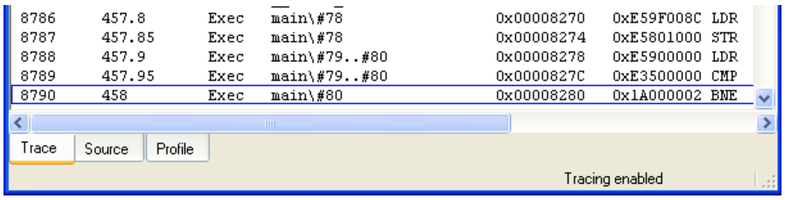
——— **Note** ———

If you are using an *Embedded Trace Buffer*™ (ETB™) to capture trace, you might not have enough trace output to see all function calls.

———

7. Scroll through the captured trace. The complete execution of `GetAverage` is traced, not just the call into the function.

- Search the captured trace for an address:
  1. Select **Find Address Expression...** from the **Find** menu to display the Enter Value dialog box.
  2. Enter the address **0x8280**, the Trace Start Point address.
  3. Click **Find**. The next trace sample with an address `0x8280` is selected and displayed. Figure 10-27 shows an example:



```
8786    457.8     Exec    main\#78          0x00008270   0xE59F008C LDR
8787    457.85    Exec    main\#78          0x00008274   0xE5801000 STR
8788    457.9     Exec    main\#79..#80     0x00008278   0xE5900000 LDR
8789    457.95    Exec    main\#79..#80     0x0000827C   0xE3500000 CMP
8790    458       Exec    main\#80          0x00008280   0x1A000002 BNE
```

Figure 10-27 Searched address in the Trace view

  4. Press F3 to find each subsequent sample with an address of `0x8280`. In this case, there are no trace pause messages.

- To summarize, any instruction execution that occurs between the start and stop points is traced, including function calls and any exception handlers that execute.

See also

- *Displaying captured trace on a stop condition* on page 2-21
- *Trace Start Point and Trace End Point* on page 5-4
- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Error messages* on page C-2
- *Warning messages* on page C-4
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `DISASSEMBLE`, `ETM_CONFIG`, and `TRACE` commands.

10.7.4 Tracing with a trace range

To capture trace using a trace range:

1. Remove the trace start and end points you set previously:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.

d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The tracepoints are removed.

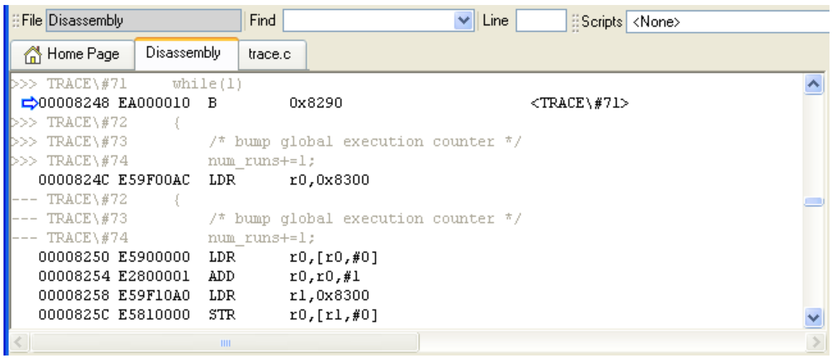
2. Set a Start of Trace Range for instructions only:

a. In the Code window, click the **trace.c** tab.

b. Right-click on line 71 in the **trace.c** tab to display the context menu.

c. Select **Locate Disassembly** from the context menu to display the Prompt dialog box.

d. The **Disassembly** tab is displayed, and the corresponding address is indicated by a hollow blue arrow. Figure 10-28 shows an example:



```
>>> TRACE\#71     while(1)
 00008248 EA000010  B       0x8290                 <TRACE\#71>
>>> TRACE\#72     {
>>> TRACE\#73         /* bump global execution counter */
>>> TRACE\#74         num_runs+=1;
 0000824C E59F00AC  LDR     r0,0x8300
--- TRACE\#72     {
--- TRACE\#73         /* bump global execution counter */
--- TRACE\#74         num_runs+=1;
 00008250 E5900000  LDR     r0,[r0,#0]
 00008254 E2800001  ADD     r0,r0,#1
 00008258 E59F10A0  LDR     r1,0x8300
 0000825C E5810000  STR     r0,[r1,#0]
```

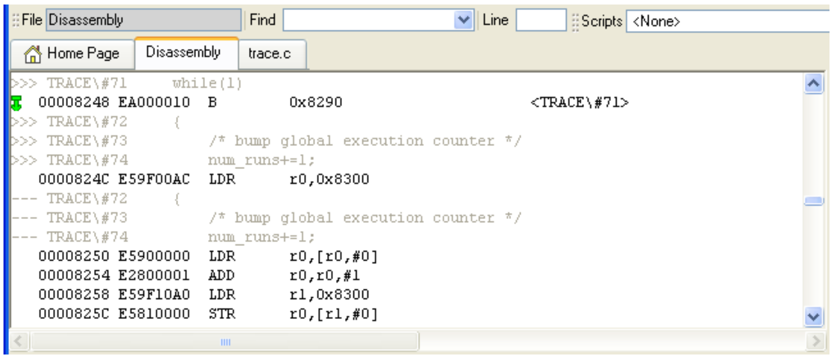
Figure 10-28 Disassembly view

e. Right-click in the margin at the address corresponding to `while(1)` to display the context menu.

f. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

g. Select **Start of Trace Range (Instruction Only)** from the list of tracepoint types.

h. Click **OK**. The dialog box closes, and a Start of Trace Range tracepoint is set as indicated by the green arrow. Figure 10-29 shows an example:

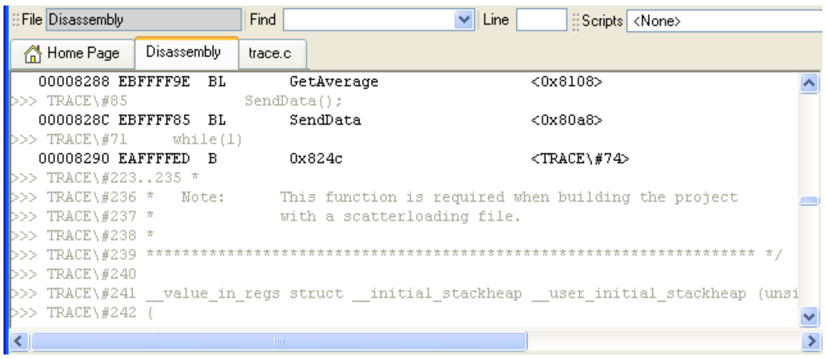


```
>>> TRACE\#71     while(1)
 00008248 EA000010  B       0x8290                 <TRACE\#71>
>>> TRACE\#72     {
>>> TRACE\#73         /* bump global execution counter */
>>> TRACE\#74         num_runs+=1;
 0000824C E59F00AC  LDR     r0,0x8300
--- TRACE\#72     {
--- TRACE\#73         /* bump global execution counter */
--- TRACE\#74         num_runs+=1;
 00008250 E5900000  LDR     r0,[r0,#0]
 00008254 E2800001  ADD     r0,r0,#1
 00008258 E59F10A0  LDR     r1,0x8300
 0000825C E5810000  STR     r0,[r1,#0]
```

Figure 10-29 Start of Trace Range set

3. Set an End of Trace Range for instructions only at the address of the next instance of `while(1)`:

a. Scroll down the **Disassembly** tab until the next instance of `while(1)` is visible. Figure 10-30 on page 10-30 shows an example:



```
  00008288 EBFFFF9E  BL       GetAverage                <0x8108>
>>> TRACE\#85             SendData();
  0000828C EBFFFF85  BL       SendData                  <0x80a8>
>>> TRACE\#71      while(1)
  00008290 EAFFFFED  B        0x824c                    <TRACE\#74>
>>> TRACE\#223..235 *
>>> TRACE\#236 *   Note:     This function is required when building the project
>>> TRACE\#237 *             with a scatterloading file.
>>> TRACE\#238 *
>>> TRACE\#239 ******************************************************************** */
>>> TRACE\#240
>>> TRACE\#241 __value_in_regs struct __initial_stackheap __user_initial_stackheap (unsi
>>> TRACE\#242 {
```

Figure 10-30 Disassembly view

b. Right-click in the margin at the address corresponding to `while(1)` to display the context menu.

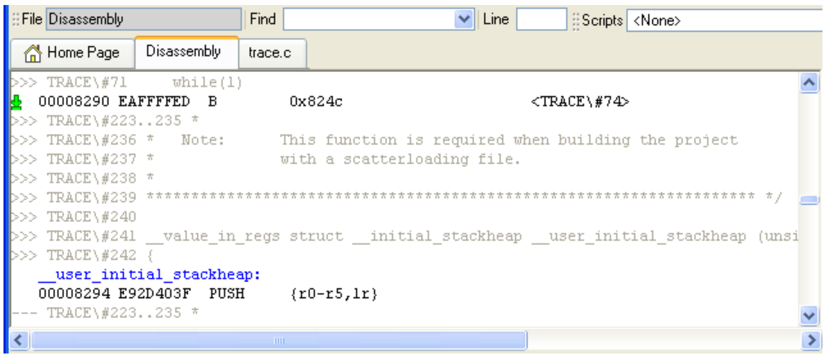
c. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

d. Select **End of Trace Range (Instruction Only)** from the list of tracepoint types.

Note

This is the only trace range option available, because you must complete the current trace range before another trace range can be set.

e. Click **OK**. The dialog box closes, and the End of Trace Range tracepoint is set as indicated by the green arrow . Figure 10-31 shows an example:

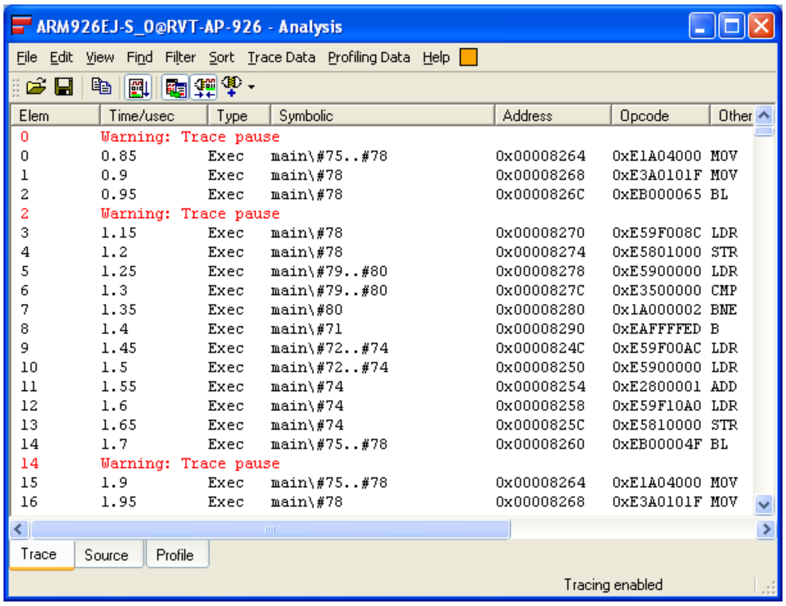


```
>>> TRACE\#71      while(1)
  00008290 EAFFFFED  B        0x824c                    <TRACE\#74>
>>> TRACE\#223..235 *
>>> TRACE\#236 *   Note:     This function is required when building the project
>>> TRACE\#237 *             with a scatterloading file.
>>> TRACE\#238 *
>>> TRACE\#239 ******************************************************************** */
>>> TRACE\#240
>>> TRACE\#241 __value_in_regs struct __initial_stackheap __user_initial_stackheap (unsi
>>> TRACE\#242 {
  __user_initial_stackheap:
  00008294 E92D403F  PUSH     {r0-r5,lr}
--- TRACE\#223..235 *
```

Figure 10-31 End of Trace Range set

4. Click **Run** on the Debug toolbar to start execution.

5. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-32 on page 10-31 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem   Time/usec  Type  Symbolic         Address     Opcode        Other
0      Warning: Trace pause
0      0.85       Exec  main\#75..#78    0x00008264  0xE1A04000 MOV
1      0.9        Exec  main\#78         0x00008268  0xE3A0101F MOV
2      0.95       Exec  main\#78         0x0000826C  0xEB000065 BL
2      Warning: Trace pause
3      1.15       Exec  main\#78         0x00008270  0xE59F008C LDR
4      1.2        Exec  main\#78         0x00008274  0xE5801000 STR
5      1.25       Exec  main\#79..#80    0x00008278  0xE5900000 LDR
6      1.3        Exec  main\#79..#80    0x0000827C  0xE3500000 CMP
7      1.35       Exec  main\#80         0x00008280  0x1A000002 BNE
8      1.4        Exec  main\#71         0x00008290  0xEAFFFFED B
9      1.45       Exec  main\#72..#74    0x0000824C  0xE59F00AC LDR
10     1.5        Exec  main\#72..#74    0x00008250  0xE5900000 LDR
11     1.55       Exec  main\#74         0x00008254  0xE2800001 ADD
12     1.6        Exec  main\#74         0x00008258  0xE59F10A0 LDR
13     1.65       Exec  main\#74         0x0000825C  0xE5810000 STR
14     1.7        Exec  main\#75..#78    0x00008260  0xEB00004F BL
14     Warning: Trace pause
15     1.9        Exec  main\#75..#78    0x00008264  0xE1A04000 MOV
16     1.95       Exec  main\#78         0x00008268  0xE3A0101F MOV
```

Figure 10-32 Captured trace in Analysis window

See also

- *Displaying captured trace on a stop condition* on page 2-21
- *Trace Start Point and Trace End Point* on page 5-4
- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Error messages* on page C-2
- *Warning messages* on page C-4
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — Chapter 2 *RealView Debugger Commands* for details of the `DISASSEMBLE`, `ETM_CONFIG`, and `TRACE` commands.

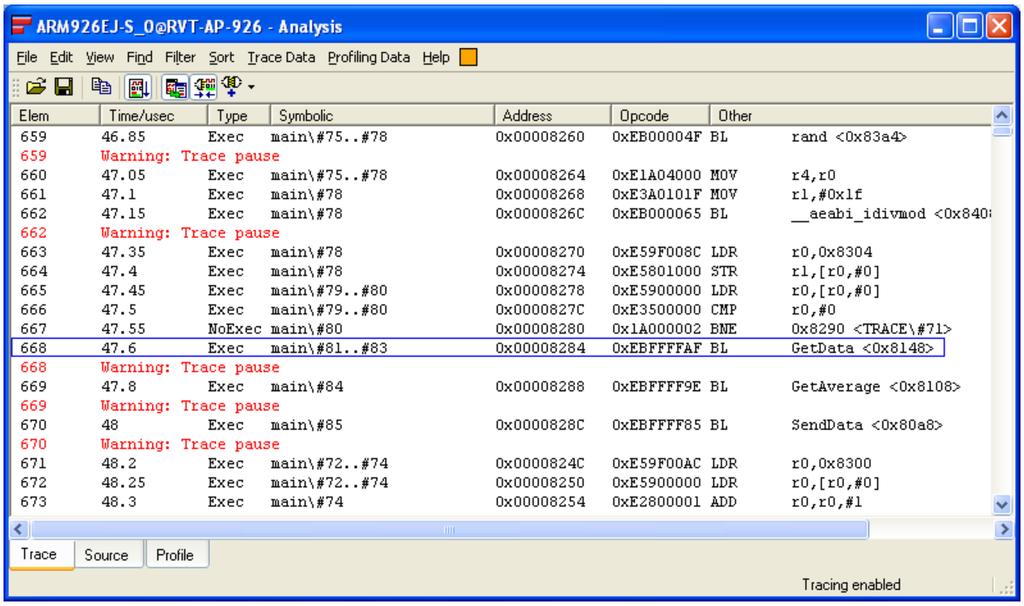
10.7.5 Examining trace captured with a trace range

Examine the captured trace in the **Trace** tab:

- Observe that within a trace range:
  - — Called functions are not traced, only the calls to functions.
  - — There are many discontinuities, which are identified by the message `Warning: Trace Pause`. This shows where instruction executions or data accesses occur, but are not captured because of the current trace conditions.
- Display the source analysis view for the captured trace:
  1. Click the **Source** tab.
  2. Scroll through the source analysis view.

  Observe that program flow does not go into `if (sample_ready == 0)` very often. Also, the same four or five lines of C appear, waiting for `sample_ready` to equal zero.

- Track the program flow:
  1. Click the **trace.c** tab in the Code window to display the source view.
  2. Move the Analysis window so that it does not overlay the source view in the Code window.
  3. Click the **Trace** tab in the Analysis window to view the captured trace.
  4. Select the first element in the captured trace.
  5. Press Down to move through the trace, and observe the flow through the source code.
  6. Scroll down through the captured trace until you see the first sample with the type `NoExec` and the symbolic name `main\#80`. This shows the trace sample when `sample_ready` equals zero.
  7. Select the trace sample with the type `NoExec`.
  8. Press Down until the trace sample containing the call to `GetData()` is highlighted. Observe that only the function calls to the three functions are traced, with a discontinuity between them. Figure 10-33 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem   Time/usec  Type   Symbolic            Address     Opcode         Other
659    46.85      Exec   main\#75..#78       0x00008260  0xEB00004F BL  rand <0x83a4>
659    Warning: Trace pause
660    47.05      Exec   main\#75..#78       0x00008264  0xE1A04000 MOV r4,r0
661    47.1       Exec   main\#78            0x00008268  0xE3A0101F MOV r1,#0x1f
662    47.15      Exec   main\#78            0x0000826C  0xEB000065 BL  __aeabi_idivmod <0x840
662    Warning: Trace pause
663    47.35      Exec   main\#78            0x00008270  0xE59F008C LDR r0,0x8304
664    47.4       Exec   main\#78            0x00008274  0xE5801000 STR r1,[r0,#0]
665    47.45      Exec   main\#79..#80       0x00008278  0xE5900000 LDR r0,[r0,#0]
666    47.5       Exec   main\#79..#80       0x0000827C  0xE3500000 CMP r0,#0
667    47.55      NoExec main\#80            0x00008280  0x1A000002 BNE 0x8290 <TRACE\#71>
668    47.6       Exec   main\#81..#83       0x00008284  0xEBFFFFAF BL  GetData <0x8148>
668    Warning: Trace pause
669    47.8       Exec   main\#84            0x00008288  0xEBFFFF9E BL  GetAverage <0x8108>
669    Warning: Trace pause
670    48         Exec   main\#85            0x0000828C  0xEBFFFF85 BL  SendData <0x80a8>
670    Warning: Trace pause
671    48.2       Exec   main\#72..#74       0x0000824C  0xE59F00AC LDR r0,0x8300
672    48.25      Exec   main\#72..#74       0x00008250  0xE5900000 LDR r0,[r0,#0]
673    48.3       Exec   main\#74            0x00008254  0xE2800001 ADD r0,r0,#1
```

Figure 10-33 Captured trace showing function calls only are traced

- Trace ranges can not be used to capture unexpected program flow changes, such as exceptions handlers.
- You can calculate the time elapsed during a discontinuity by measuring the time difference between the trace samples captured immediately before and after the discontinuity.

See also

- *Displaying captured trace on a stop condition* on page 2-21
- *Trace Start Point and Trace End Point* on page 5-4
- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Error messages* on page C-2

- *Warning messages* on page C-4
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `DISASSEMBLE`, `ETM_CONFIG`, and `TRACE` commands.

10.8 Excluding a trace range to avoid tracing a function

Sometimes, you might want to avoid tracing certain parts of your image so that you can concentrate the trace on areas of interest. For example, you might have one or more functions that you are not interested in tracing.

——— **Note** ———

You can have multiple trace exclusion ranges, but the number is limited by the resources of the ETM.

———————————

See also:

- *Tracing features used*
- *Procedure*
- *Examining the captured trace* on page 10-37.

10.8.1 Tracing features used

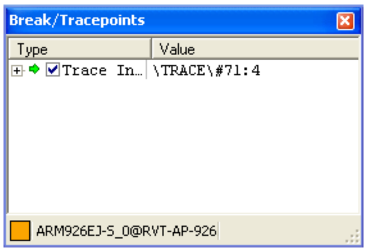
The following tracing features are used in this procedure:

- a Trace Start Point to specify the location where trace capture is to begin
- a Start of Excluded Trace Range to specify the location where trace capture is to be suspended
- an End of Excluded Trace Range to specify the location where trace capture is to continue
- searching for a function name.

10.8.2 Procedure

To exclude a function from being traced:

1. Remove the trace range you set previously:
   a. Select **Break/Tracepoints** from the **View** menu to display the Break/Tracepoints view. Figure 10-34 shows an example:



Break/Tracepoints

Type	Value
Trace In…	\TRACE\#71:4

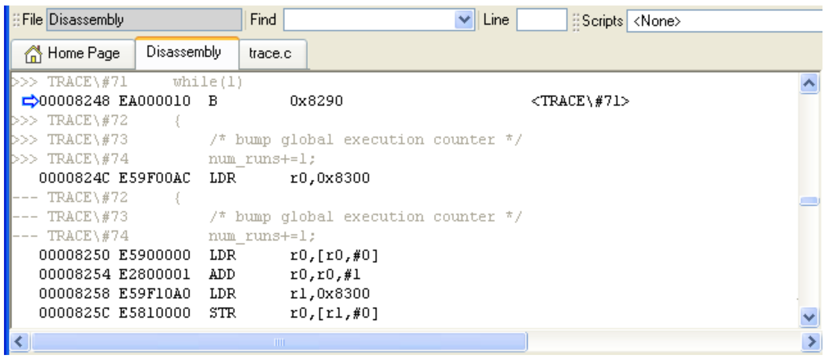
ARM926EJ-S_0@RVT-AP-926

Figure 10-34 Break/Tracepoints view showing a tracepoint

   b. Right-click on the tracepoint to display the context menu.
   c. Select **Delete** from the context menu. The tracepoint is deleted.
   d. Close the Break/Tracepoints view.

2. Reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Reload** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`).

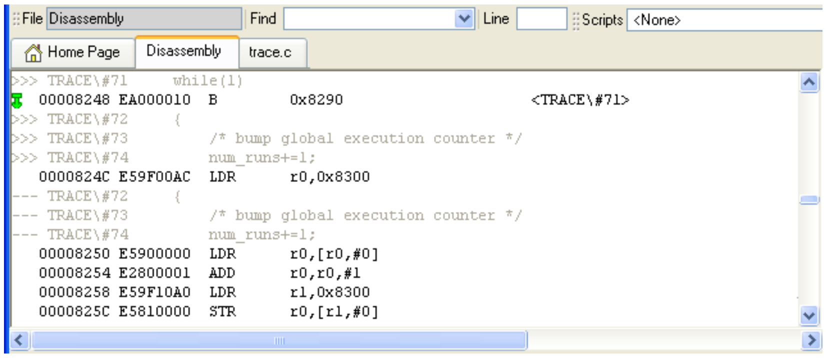
3. Set a Trace Start Point at the address corresponding to `while(1)`:
   a. In the Code window, click the **trace.c** tab.
   b. Right-click on line 71 in the **trace.c** tab to display the context menu.
   c. Select **Locate Disassembly** from the context menu to display the Prompt dialog box.
   d. The **Disassembly** tab is displayed, and the corresponding address is indicated by a hollow blue arrow. Figure 10-28 on page 10-29 shows an example:



```
>>> TRACE\#71      while(1)
 00008248 EA000010  B        0x8290                    <TRACE\#71>
>>> TRACE\#72          {
>>> TRACE\#73          /* bump global execution counter */
>>> TRACE\#74          num_runs+=1;
    0000824C E59F00AC  LDR      r0,0x8300
--- TRACE\#72          {
--- TRACE\#73          /* bump global execution counter */
--- TRACE\#74          num_runs+=1;
    00008250 E5900000  LDR      r0,[r0,#0]
    00008254 E2800001  ADD      r0,r0,#1
    00008258 E59F10A0  LDR      r1,0x8300
    0000825C E5810000  STR      r0,[r1,#0]
```

Figure 10-35 Disassembly view

   e. Right-click in the margin at the address corresponding to `while(1)` to display the context menu.
   f. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.
   g. Select **Trace Start Point** from the list of tracepoint types.
   h. Click **OK**. The dialog box closes, and the Trace Start Point is set as indicated by the green arrow. Figure 10-36 shows an example:



```
>>> TRACE\#71      while(1)
    00008248 EA000010  B        0x8290                    <TRACE\#71>
>>> TRACE\#72          {
>>> TRACE\#73          /* bump global execution counter */
>>> TRACE\#74          num_runs+=1;
    0000824C E59F00AC  LDR      r0,0x8300
--- TRACE\#72          {
--- TRACE\#73          /* bump global execution counter */
--- TRACE\#74          num_runs+=1;
    00008250 E5900000  LDR      r0,[r0,#0]
    00008254 E2800001  ADD      r0,r0,#1
    00008258 E59F10A0  LDR      r1,0x8300
    0000825C E5810000  STR      r0,[r1,#0]
```

Figure 10-36 Trace Start Point set

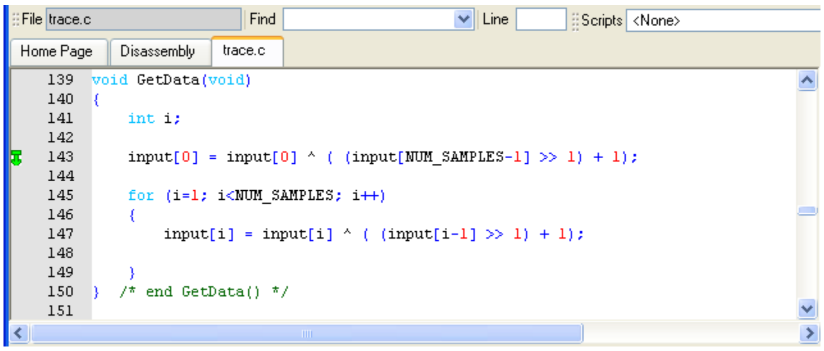
   RealView Debugger generates a `TRACE` CLI command, which is also displayed in the **Cmd** tab of the Output view, for example:

   ```
   trace,prompt 0x00008248
   ```

4. Set a Start of Excluded Trace Range for instructions and data at line 143 in the source:
   a. In the Code window, click the **trace.c** tab in the Code window.
   b. Scroll down until line 143 is visible.
   c. Right-click in the margin at line 143 to display the context menu.
   d. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

e. Select **Start of Excluded Trace Range (Instruction and Data)** from the list of tracepoint types.

f. Click **OK**. The dialog box closes, and a Start of Excluded Trace Range tracepoint is set as indicated by the green arrow. Figure 10-37 shows an example:



```
139 void GetData(void)
140 {
141     int i;
142
143     input[0] = input[0] ^ ( (input[NUM_SAMPLES-1] >> 1) + 1);
144
145     for (i=1; i<NUM_SAMPLES; i++)
146     {
147         input[i] = input[i] ^ ( (input[i-1] >> 1) + 1);
148
149     }
150 }  /* end GetData() */
151
```

Figure 10-37 Start of Excluded Trace Range set at line 142

5. Set an End of Excluded Trace Range for instructions and data at line 150 in the source:

   a. Scroll down until line 150 is visible.

   b. Right-click in the margin at line 150 to display the context menu.

   c. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

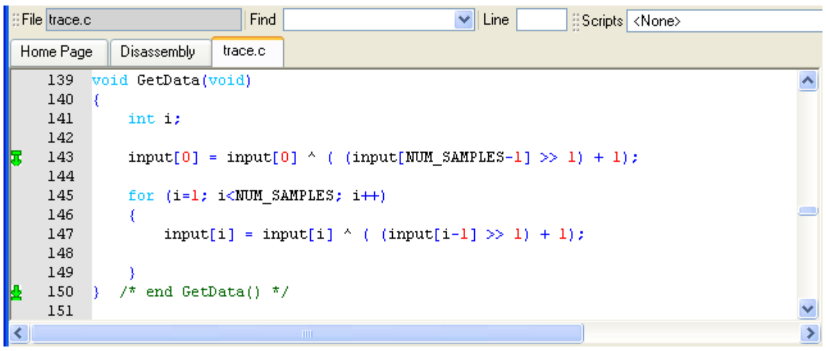
   d. Select **End of Excluded Trace Range (Instruction and Data)** from the list of tracepoint types.

   Note

   This is the only trace range option available, because you must complete the current trace range before another trace range can be set.

   ---

   e. Click **OK**. The dialog box closes, and the End of Excluded Trace Range tracepoint is set as indicated by the green arrow. Figure 10-38 shows an example:

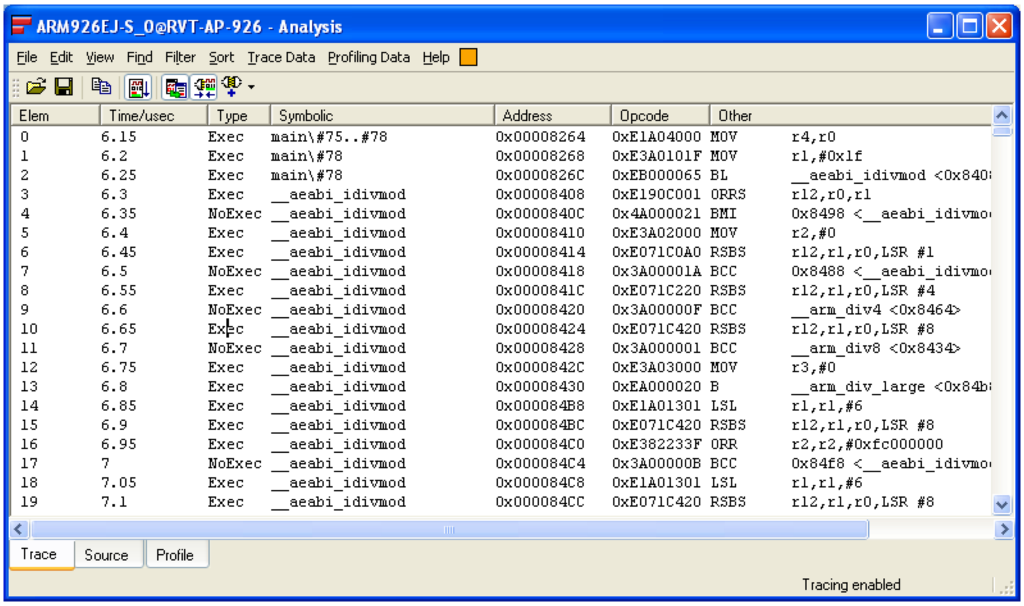


```
139 void GetData(void)
140 {
141     int i;
142
143     input[0] = input[0] ^ ( (input[NUM_SAMPLES-1] >> 1) + 1);
144
145     for (i=1; i<NUM_SAMPLES; i++)
146     {
147         input[i] = input[i] ^ ( (input[i-1] >> 1) + 1);
148
149     }
150 }  /* end GetData() */
151
```

Figure 10-38 End of Excluded Trace Range set at line 151

6. Click **Run** on the Debug toolbar to start execution.

7. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-39 on page 10-37 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Elem	Time/usec	Type	Symbolic	Address	Opcode	Other
0	6.15	Exec	main\#75..#78	0x00008264	0xE1A04000 MOV	r4,r0
1	6.2	Exec	main\#78	0x00008268	0xE3A0101F MOV	r1,#0x1f
2	6.25	Exec	main\#78	0x0000826C	0xEB000065 BL	__aeabi_idivmod <0x840
3	6.3	Exec	__aeabi_idivmod	0x00008408	0xE190C001 ORRS	r12,r0,r1
4	6.35	NoExec	__aeabi_idivmod	0x0000840C	0x4A000021 BMI	0x8498 <__aeabi_idivmo
5	6.4	Exec	__aeabi_idivmod	0x00008410	0xE3A02000 MOV	r2,#0
6	6.45	Exec	__aeabi_idivmod	0x00008414	0xE071C0A0 RSBS	r12,r1,r0,LSR #1
7	6.5	NoExec	__aeabi_idivmod	0x00008418	0x3A00001A BCC	0x8488 <__aeabi_idivmo
8	6.55	Exec	__aeabi_idivmod	0x0000841C	0xE071C220 RSBS	r12,r1,r0,LSR #4
9	6.6	NoExec	__aeabi_idivmod	0x00008420	0x3A00000F BCC	__arm_div4 <0x8464>
10	6.65	Exec	__aeabi_idivmod	0x00008424	0xE071C420 RSBS	r12,r1,r0,LSR #8
11	6.7	NoExec	__aeabi_idivmod	0x00008428	0x3A000001 BCC	__arm_div8 <0x8434>
12	6.75	Exec	__aeabi_idivmod	0x0000842C	0xE3A03000 MOV	r3,#0
13	6.8	Exec	__aeabi_idivmod	0x00008430	0xEA000020 B	__arm_div_large <0x84b
14	6.85	Exec	__aeabi_idivmod	0x000084B8	0xE1A01301 LSL	r1,r1,#6
15	6.9	Exec	__aeabi_idivmod	0x000084BC	0xE071C420 RSBS	r12,r1,r0,LSR #8
16	6.95	Exec	__aeabi_idivmod	0x000084C0	0xE382233F ORR	r2,r2,#0xfc000000
17	7	NoExec	__aeabi_idivmod	0x000084C4	0x3A00000B BCC	0x84f8 <__aeabi_idivmo
18	7.05	Exec	__aeabi_idivmod	0x000084C8	0xE1A01301 LSL	r1,r1,#6
19	7.1	Exec	__aeabi_idivmod	0x000084CC	0xE071C420 RSBS	r12,r1,r0,LSR #8

Figure 10-39 Captured trace in Analysis window

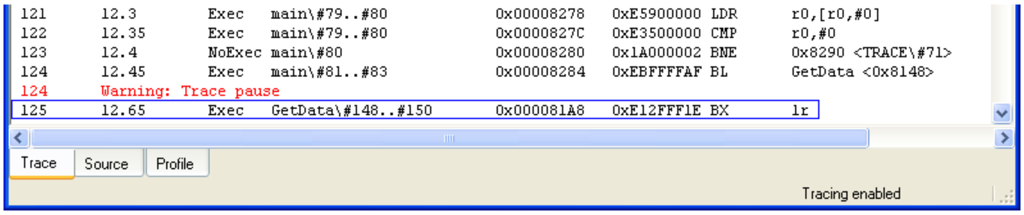
See also

- *Displaying captured trace on a stop condition* on page 2-21
- *Trace Start Point and Trace End Point* on page 5-4
- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Finding information in captured trace* on page 9-16
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `DISASSEMBLE` and `TRACE` commands.

10.8.3 Examining the captured trace

Examine the captured trace in the **Trace** tab:

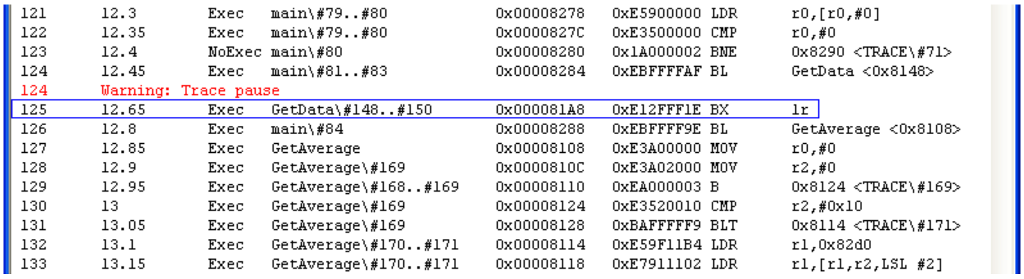
- All program execution from `while(1)` onwards is traced, except for the body of `GetData()`.
- Search for the first instance of `GetData()` in the captured trace:
  1. Click the **Trace** tab to display the captured trace.
  2. Select **Find Symbol Name...** from the **Find** menu to display the Enter Value dialog box.
  3. Enter **GetData** (without the parentheses).
  4. Click **Find**. The first instance of `GetData` in the captured trace is selected. Figure 10-40 on page 10-38 shows an example:



```
121    12.3     Exec    main\#79..#80       0x00008278  0xE5900000 LDR    r0,[r0,#0]
122    12.35    Exec    main\#79..#80       0x0000827C  0xE3500000 CMP    r0,#0
123    12.4     NoExec  main\#80            0x00008280  0x1A000002 BNE    0x8290 <TRACE\#71>
124    12.45    Exec    main\#81..#83       0x00008284  0xEBFFFFAF BL     GetData <0x8148>
124    Warning: Trace pause
125    12.65    Exec    GetData\#148..#150  0x000081A8  0xE12FFF1E BX     lr
```

Figure 10-40 Searched function in Trace view

5. Scroll through the captured trace. Observe that only the function call to `GetData()` and the instructions to return to `main()` are traced. Figure 10-41 shows an example:

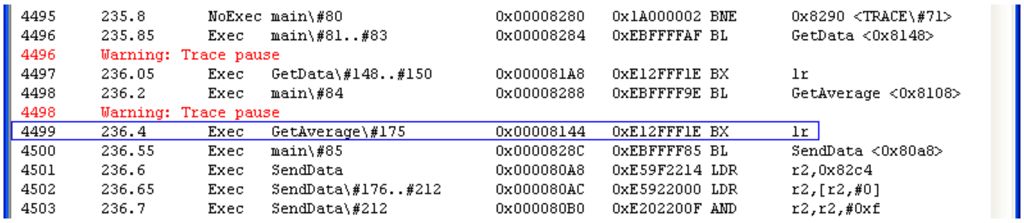


```
121    12.3     Exec    main\#79..#80          0x00008278  0xE5900000 LDR    r0,[r0,#0]
122    12.35    Exec    main\#79..#80          0x0000827C  0xE3500000 CMP    r0,#0
123    12.4     NoExec  main\#80               0x00008280  0x1A000002 BNE    0x8290 <TRACE\#71>
124    12.45    Exec    main\#81..#83          0x00008284  0xEBFFFFAF BL     GetData <0x8148>
124    Warning: Trace pause
125    12.65    Exec    GetData\#148..#150     0x000081A8  0xE12FFF1E BX     lr
126    12.8     Exec    main\#84               0x00008288  0xEBFFFF9E BL     GetAverage <0x8108>
127    12.85    Exec    GetAverage             0x00008108  0xE3A00000 MOV    r0,#0
128    12.9     Exec    GetAverage\#169        0x0000810C  0xE3A02000 MOV    r2,#0
129    12.95    Exec    GetAverage\#168..#169  0x00008110  0xEA000003 B      0x8124 <TRACE\#169>
130    13       Exec    GetAverage\#169        0x00008124  0xE3520010 CMP    r2,#0x10
131    13.05    Exec    GetAverage\#169        0x00008128  0xBAFFFFF9 BLT    0x8114 <TRACE\#171>
132    13.1     Exec    GetAverage\#170..#171  0x00008114  0xE59F11B4 LDR    r1,0x82d0
133    13.15    Exec    GetAverage\#170..#171  0x00008118  0xE7911102 LDR    r1,[r1,r2,LSL #2]
```

Figure 10-41 Captured trace near the GetData() function

- Set another exclude range to exclude tracing of `GetAverage()`:
  1. Click the **trace.c** tab in the Code window.
  2. Scroll down until the start of `GetAverage()` is visible.
  3. Right-click in the gray margin at line 167 to display the context menu.
  4. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.
  5. Select **Start of Excluded Trace Range (Instruction and Data)** from the list of tracepoint types.
  6. Click **OK**. The Start of Excluded Trace Range tracepoint is set.
  7. Scroll down until the end of `GetAverage()` is visible.
  8. Right-click in the gray margin at line 175 to display the context menu.
  9. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.
  10. Select **End of Excluded Trace Range (Instruction and Data)** from the list of tracepoint types.
  11. Click **OK**. The End of Excluded Trace Range tracepoint is set.
  12. Reload the image.
  13. Click **Run** on the Debug toolbar to start execution.
  14. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window.
  15. Search for the first instance of `GetAverage()` in the captured trace.
      a. Click the **Trace** tab to display the captured trace.
      b. Select **Find Symbol Name...** from the **Find** menu to display the Enter Value dialog box.
      c. Enter **GetAverage** (without the parentheses).
      d. Click **Find**. The first instance of `GetAverage` in the captured trace is selected.

16. Scroll a short way through the captured trace. Observe that only the function calls to `GetData()` and `GetAverage()`, and the instructions to return to `main()` are traced. Figure 10-42 shows an example:



```
4495    235.8     NoExec main\#80               0x00008280   0x1A000002 BNE     0x8290 <TRACE\#71>
4496    235.85    Exec   main\#81..#83          0x00008284   0xEBFFFFAF BL      GetData <0x8148>
4496    Warning: Trace pause
4497    236.05    Exec   GetData\#148..#150     0x000081A8   0xE12FFF1E BX      lr
4498    236.2     Exec   main\#84               0x00008288   0xEBFFFF9E BL      GetAverage <0x8108>
4498    Warning: Trace pause
4499    236.4     Exec   GetAverage\#175        0x00008144   0xE12FFF1E BX      lr
4500    236.55    Exec   main\#85               0x0000828C   0xEBFFFF85 BL      SendData <0x80a8>
4501    236.6     Exec   SendData               0x000080A8   0xE59F2214 LDR     r2,0x82c4
4502    236.65    Exec   SendData\#176..#212    0x000080AC   0xE5922000 LDR     r2,[r2,#0]
4503    236.7     Exec   SendData\#212          0x000080B0   0xE202200F AND     r2,r2,#0xf
```

Figure 10-42 Captured trace near the GetAverage() function

- Restart and stop execution again:
  1. Click **Run** on the Debug toolbar to restart execution.
  2. After a short time, click **Stop** on the Debug toolbar to stop execution. No trace is captured in this case. Because there is a Trace Start Point at line 71, tracing is disabled until the Trace Start Point is passed. Execution must pass the Trace Start Point to cause tracing to be enabled again. Therefore, to capture trace again, you must reload the image before restarting execution.

See also

- *Displaying captured trace on a stop condition* on page 2-21
- *Trace Start Point and Trace End Point* on page 5-4
- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Finding information in captured trace* on page 9-16
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `DISASSEMBLE` and `TRACE` commands.

10.9 Tracing data with auto trace

Some problems might be due to unexpected data values being used in your program. Therefore, you must be able to see the data accesses that are being preformed by your program.

See also:

- *Tracing features used*
- *Procedure*
- *Examining the captured trace* on page 10-42
- *Displaying instruction boundaries* on page 10-44
- *Displaying function boundaries* on page 10-46
- *Displaying inferred registers* on page 10-47.

10.9.1 Tracing features used

The following tracing features are used in this procedure:

- automatic tracing of instructions and data (that is, without tracepoints)
- tracing of data and address (the default)
- 16-bit data width to minimize FIFO overflow
- searching for an address
- improving the readability of the captured trace by viewing:
  - instruction boundaries
  - function boundaries
  - inferred registers.

10.9.2 Procedure

To automatically trace instructions and data:

1. Remove the tracepoints you set previously:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The tracepoints are removed.
2. Select **Edit** → **Automatic Tracing Mode** → **Instructions and Data** from the Analysis window menu. RealView Debugger generates an `ANALYZER` CLI command, which is also displayed in the **Cmd** tab of the Output view:

   ```
   analyzer,auto_both
   ```

3. Capture of data accesses (reads and writes) and addresses is performed by default. To check the data tracing mode, select **Data Tracing Mode** from the **Edit** menu in the Analysis window. The **Data Tracing Mode** submenu shows that **Data and Address** is currently selected.
4. Set up the ETM:
   a. Select **Configure Analyzer Properties...** from the **Edit** menu in the Analysis window to display the Configure ETM dialog box.
   b. In the Trace data width group, make sure that **16 bit** is selected.

c. Click **OK** to close the dialog box.

5. Add watches for `&average` and `average`:

   a. Select **Watch** from the **View** menu of the Code window to display the Watch view.

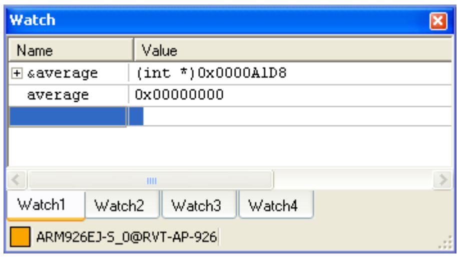
   b. Enter the variable name **&average**. Make sure you press Enter to finish the entry.

   c. Click the **trace.c** tab in the Code window.

   d. Scroll to the beginning of the source file until the definition of `average` is visible at line 23.

   e. Double click on `average`, to select the variable.

   f. Drag and drop the selected variable into the Watch view. Figure 10-43 shows an example:



```
Name          Value
+ &average    (int *)0x0000A1D8
  average     0x00000000
```

Figure 10-43 Watch view

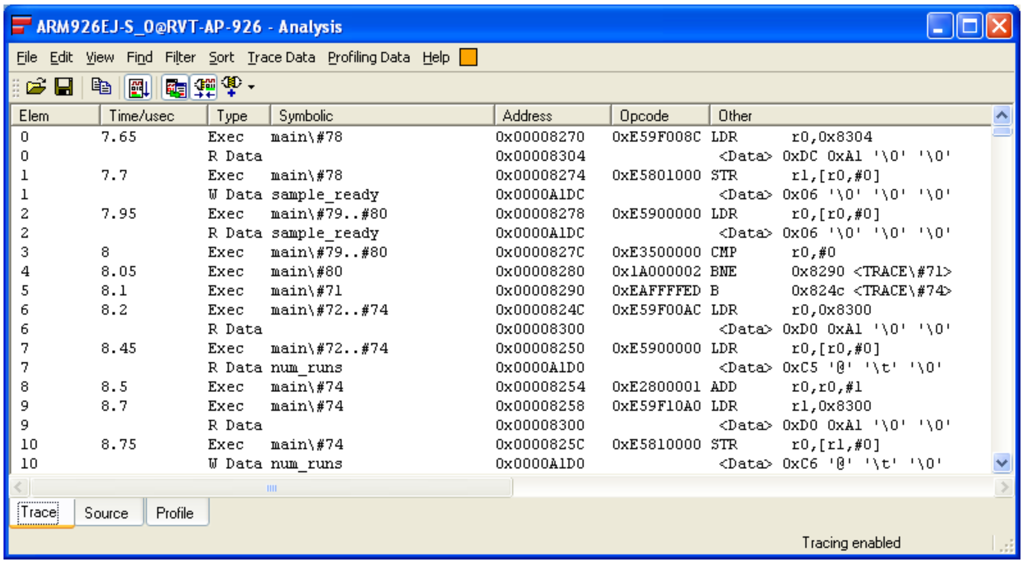
6. Click **Run** on the Debug toolbar to start execution.

7. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window.

8. Scroll down until the start of main is visible. Figure 10-44 shows an example.

   Note

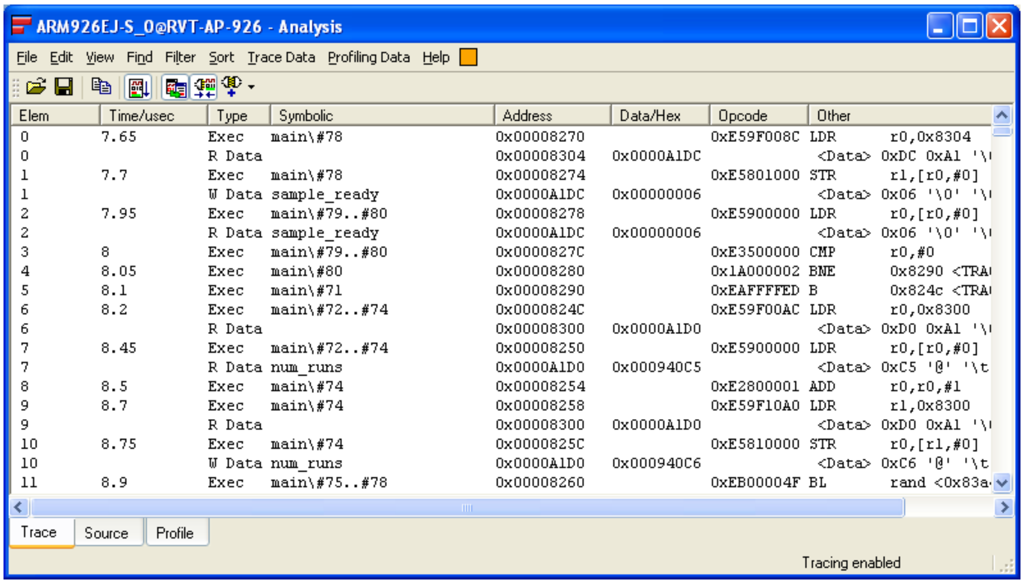
   The captured trace depends on the length of execution. Therefore, the element numbers shown in your captured trace might be different to that shown in this example.



```
Elem  Time/usec  Type    Symbolic         Address     Opcode          Other
0     7.65       Exec    main\#78         0x00008270  0xE59F008C LDR      r0,0x8304
0                R Data                   0x00008304            <Data> 0xDC 0xA1 '\0' '\0'
1     7.7        Exec    main\#78         0x00008274  0xE5801000 STR      r1,[r0,#0]
1                W Data sample_ready      0x0000A1DC            <Data> 0x06 '\0' '\0' '\0'
2     7.95       Exec    main\#79..#80    0x00008278  0xE5900000 LDR      r0,[r0,#0]
2                R Data sample_ready      0x0000A1DC            <Data> 0x06 '\0' '\0' '\0'
3     8          Exec    main\#79..#80    0x0000827C  0xE3500000 CMP      r0,#0
4     8.05       Exec    main\#80         0x00008280  0x1A000002 BNE      0x8290 <TRACE\#71>
5     8.1        Exec    main\#71         0x00008290  0xEAFFFFED B        0x824c <TRACE\#74>
6     8.2        Exec    main\#72..#74    0x0000824C  0xE59F00AC LDR      r0,0x8300
6                R Data                   0x00008300            <Data> 0xD0 0xA1 '\0' '\0'
7     8.45       Exec    main\#72..#74    0x00008250  0xE5900000 LDR      r0,[r0,#0]
7                R Data num_runs          0x0000A1D0            <Data> 0xC5 '@' '\t' '\0'
8     8.5        Exec    main\#74         0x00008254  0xE2800001 ADD      r0,r0,#1
9     8.7        Exec    main\#74         0x00008258  0xE59F10A0 LDR      r1,0x8300
9                R Data                   0x00008300            <Data> 0xD0 0xA1 '\0' '\0'
10    8.75       Exec    main\#74         0x0000825C  0xE5810000 STR      r0,[r1,#0]
10               W Data num_runs          0x0000A1D0            <Data> 0xC6 '@' '\t' '\0'
```

Figure 10-44 Captured trace in Analysis window

9. Select **Data Value in Hex** from the **Trace Data** menu in the Analysis window to display the `Data/Hex` column. You might have to scroll down again to see the start of main. Figure 10-45 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem  Time/usec  Type    Symbolic          Address     Data/Hex    Opcode          Other
0     7.65       Exec    main\#78          0x00008270              0xE59F008C LDR  r0,0x8304
0                R Data                    0x00008304  0x0000A1DC           <Data> 0xDC 0xA1 '\
1     7.7        Exec    main\#78          0x00008274              0xE5801000 STR  r1,[r0,#0]
1                W Data  sample_ready      0x0000A1DC  0x00000006           <Data> 0x06 '\0' '\
2     7.95       Exec    main\#79..#80     0x00008278              0xE5900000 LDR  r0,[r0,#0]
2                R Data  sample_ready      0x0000A1DC  0x00000006           <Data> 0x06 '\0' '\
3     8          Exec    main\#79..#80     0x0000827C              0xE3500000 CMP  r0,#0
4     8.05       Exec    main\#80          0x00008280              0x1A000002 BNE  0x8290 <TRA
5     8.1        Exec    main\#71          0x00008290              0xEAFFFFED B    0x824c <TRA
6     8.2        Exec    main\#72..#74     0x0000824C              0xE59F00AC LDR  r0,0x8300
6                R Data                    0x00008300  0x0000A1D0           <Data> 0xD0 0xA1 '\
7     8.45       Exec    main\#72..#74     0x00008250              0xE5900000 LDR  r0,[r0,#0]
7                R Data  num_runs          0x0000A1D0  0x000940C5           <Data> 0xC5 '@' '\t
8     8.5        Exec    main\#74          0x00008254              0xE2800001 ADD  r0,r0,#1
9     8.7        Exec    main\#74          0x00008258              0xE59F10A0 LDR  r1,0x8300
9                R Data                    0x00008300  0x0000A1D0           <Data> 0xD0 0xA1 '\
10    8.75       Exec    main\#74          0x0000825C              0xE5810000 STR  r0,[r1,#0]
10               W Data  num_runs          0x0000A1D0  0x000940C6           <Data> 0xC6 '@' '\t
11    8.9        Exec    main\#75..#78     0x00008260              0xEB00004F BL   rand <0x83a
```

Figure 10-45 Data/Hex column displayed in the Analysis window

See also

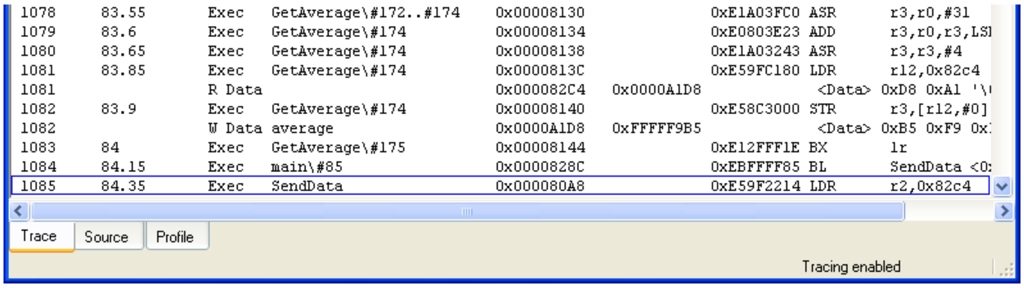
- *Capturing instructions and data* on page 6-3
- *Displaying instruction boundaries* on page 9-7
- *Displaying function boundaries* on page 9-7
- *Displaying inferred registers* on page 9-8
- *Finding information in captured trace* on page 9-16
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `ANALYZER`, `ETM_CONFIG`, and `TRACEBUFFER` commands.

10.9.3 Examining the captured trace

Examine the captured trace in the **Trace** tab:

- All the data access are included in the captured trace.
- Search for the first instance of `SendData`:
  1. Click the **Trace** tab to display the captured trace.
  2. Select **Find Symbol Name...** from the **Find** menu to display the Enter Value dialog box.
  3. Enter **SendData**.
  4. Click **Find**. The dialog box closes, and the first instance of `SendData` in the captured trace is selected. Figure 10-46 on page 10-43 shows an example. RealView Debugger generates a `TRACEBUFFER` CLI command, which is also displayed in the **Cmd** tab of the Output view, for example:

     ```
     tracebuffer,gui,find_addr=0x000080A8..0x00008107
     ```



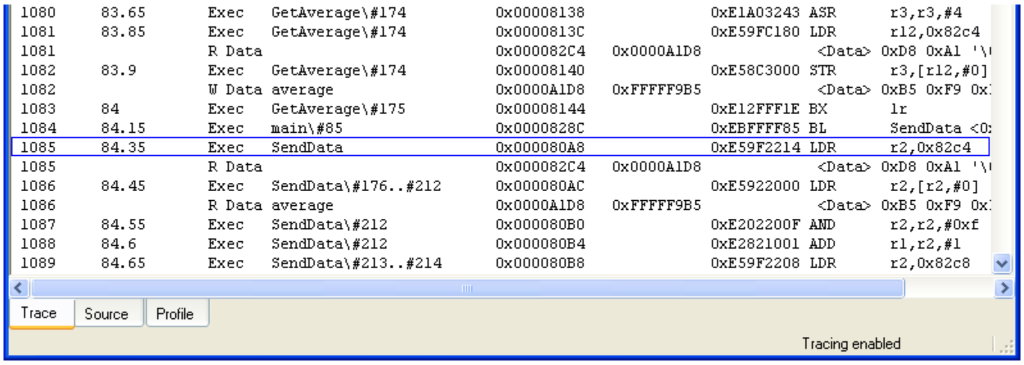
```
1078   83.55   Exec    GetAverage\#172..#174   0x00008130                0xE1A03FC0 ASR    r3,r0,#31
1079   83.6    Exec    GetAverage\#174         0x00008134                0xE0803E23 ADD    r3,r0,r3,LS
1080   83.65   Exec    GetAverage\#174         0x00008138                0xE1A03243 ASR    r3,r3,#4
1081   83.85   Exec    GetAverage\#174         0x0000813C                0xE59FC180 LDR    r12,0x82c4
1081           R Data                          0x000082C4   0x0000A1D8              <Data> 0xD8 0xA1 '\
1082   83.9    Exec    GetAverage\#174         0x00008140                0xE58C3000 STR    r3,[r12,#0]
1082           W Data  average                 0x0000A1D8   0xFFFFF9B5              <Data> 0xB5 0xF9 0x
1083   84      Exec    GetAverage\#175         0x00008144                0xE12FFF1E BX     lr
1084   84.15   Exec    main\#85                0x0000828C                0xEBFFFF85 BL     SendData <0
1085   84.35   Exec    SendData                0x000080A8                0xE59F2214 LDR    r2,0x82c4
```

Trace | Source | Profile

Tracing enabled

Figure 10-46 Searched function in Trace view

5. Scroll through the captured trace to view the next two data trace samples. Figure 10-47 shows an example. In this example, the read accesses from elements 1085 and 1086 are used to compute `num_xmit` (line 212 in `trace.c`).



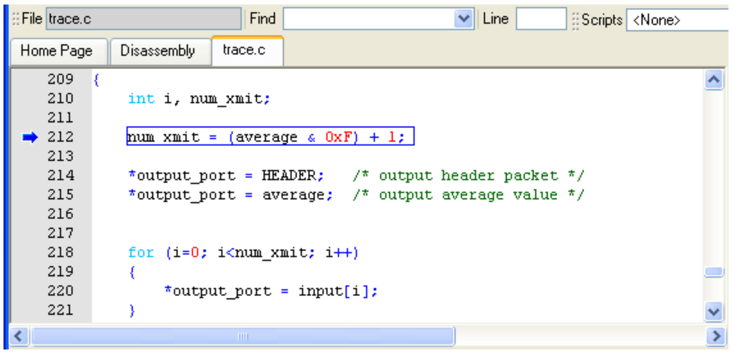
```
1080   83.65   Exec    GetAverage\#174         0x00008138                0xE1A03243 ASR    r3,r3,#4
1081   83.85   Exec    GetAverage\#174         0x0000813C                0xE59FC180 LDR    r12,0x82c4
1081           R Data                          0x000082C4   0x0000A1D8              <Data> 0xD8 0xA1 '\
1082   83.9    Exec    GetAverage\#174         0x00008140                0xE58C3000 STR    r3,[r12,#0]
1082           W Data  average                 0x0000A1D8   0xFFFFF9B5              <Data> 0xB5 0xF9 0x
1083   84      Exec    GetAverage\#175         0x00008144                0xE12FFF1E BX     lr
1084   84.15   Exec    main\#85                0x0000828C                0xEBFFFF85 BL     SendData <0
1085   84.35   Exec    SendData                0x000080A8                0xE59F2214 LDR    r2,0x82c4
1085           R Data                          0x000082C4   0x0000A1D8              <Data> 0xD8 0xA1 '\
1086   84.45   Exec    SendData\#176..#212     0x000080AC                0xE5922000 LDR    r2,[r2,#0]
1086           R Data  average                 0x0000A1D8   0xFFFFF9B5              <Data> 0xB5 0xF9 0x
1087   84.55   Exec    SendData\#212           0x000080B0                0xE202200F AND    r2,r2,#0xf
1088   84.6    Exec    SendData\#212           0x000080B4                0xE2821001 ADD    r1,r2,#1
1089   84.65   Exec    SendData\#213..#214     0x000080B8                0xE59F2208 LDR    r2,0x82c8
```

Trace | Source | Profile

Tracing enabled

Figure 10-47 Data trace samples for SendData()

6. Click the **trace.c** tab in the Code window to see the corresponding line of source. Figure 10-48 shows an example:



File trace.c | Find | Line | Scripts <None>

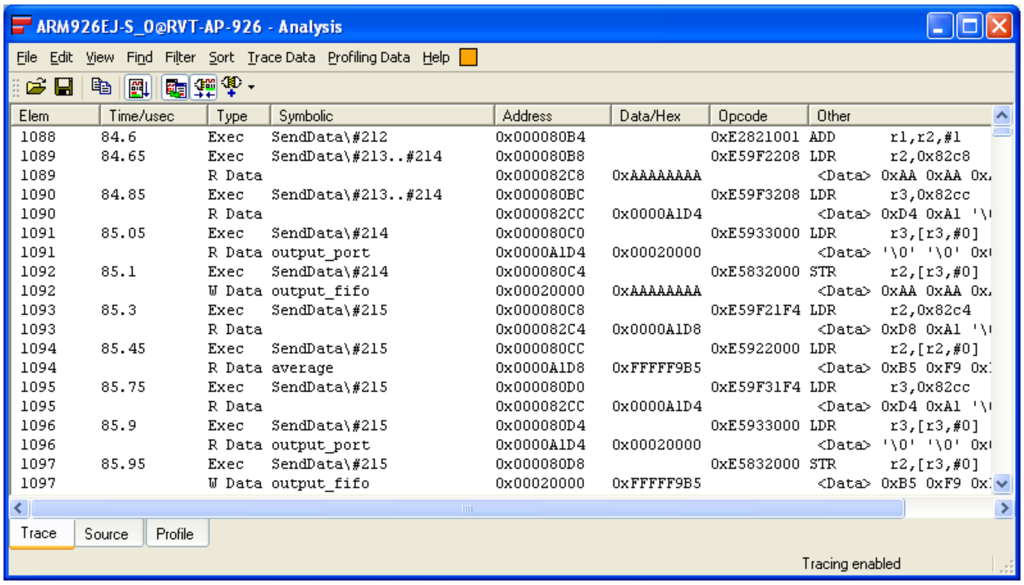
Home Page | Disassembly | trace.c

```
209  {
210      int i, num_xmit;
211
212      num_xmit = (average & 0xF) + 1;
213
214      *output_port = HEADER;    /* output header packet */
215      *output_port = average;   /* output average value */
216
217
218      for (i=0; i<num_xmit; i++)
219      {
220          *output_port = input[i];
221      }
```

Figure 10-48 Corresponding source view

- Scroll down the trace until elements 10346 to 10373 are visible:
  - — Elements 1089 to 1092 (`W Data`) show how the header value (`0xAAAAAAAA`) is written to `*output_fifo` (`0x20000`).
  - — Elements 1093 (`Exec`) to 1097 (`W Data`) show how the value of `average` (`0xFFFFF9B5`) is written to `*output_fifo` (`0x20000`).
  - — The variable `average` is stored at `0xA1D4`, shown in the sample at element 1095.

  Figure 10-49 on page 10-44 shows an example:



ARM926EJ-S_0@RVT-AP-926 - Analysis

Elem	Time/usec	Type	Symbolic	Address	Data/Hex	Opcode	Other
1088	84.6	Exec	SendData\#212	0x000080B4		0xE2821001	ADD r1,r2,#1
1089	84.65	Exec	SendData\#213..#214	0x000080B8		0xE59F2208	LDR r2,0x82c8
1089		R Data		0x000082C8	0xAAAAAAAA		<Data> 0xAA 0xAA 0x
1090	84.85	Exec	SendData\#213..#214	0x000080BC		0xE59F3208	LDR r3,0x82cc
1090		R Data		0x000082CC	0x0000A1D4		<Data> 0xD4 0xA1 '\
1091	85.05	Exec	SendData\#214	0x000080C0		0xE5933000	LDR r3,[r3,#0]
1091		R Data	output_port	0x0000A1D4	0x00020000		<Data> '\0' '\0' 0x
1092	85.1	Exec	SendData\#214	0x000080C4		0xE5832000	STR r2,[r3,#0]
1092		W Data	output_fifo	0x00020000	0xAAAAAAAA		<Data> 0xAA 0xAA 0x
1093	85.3	Exec	SendData\#215	0x000080C8		0xE59F21F4	LDR r2,0x82c4
1093		R Data		0x000082C4	0x0000A1D8		<Data> 0xD8 0xA1 '\
1094	85.45	Exec	SendData\#215	0x000080CC		0xE5922000	LDR r2,[r2,#0]
1094		R Data	average	0x0000A1D8	0xFFFFF9B5		<Data> 0xB5 0xF9 0x
1095	85.75	Exec	SendData\#215	0x000080D0		0xE59F31F4	LDR r3,0x82cc
1095		R Data		0x000082CC	0x0000A1D4		<Data> 0xD4 0xA1 '\
1096	85.9	Exec	SendData\#215	0x000080D4		0xE5933000	LDR r3,[r3,#0]
1096		R Data	output_port	0x0000A1D4	0x00020000		<Data> '\0' '\0' 0x
1097	85.95	Exec	SendData\#215	0x000080D8		0xE5832000	STR r2,[r3,#0]
1097		W Data	output_fifo	0x00020000	0xFFFFF9B5		<Data> 0xB5 0xF9 0x

File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Figure 10-49 Write of average to output_port

- Search for elements of interest:
  1. Make a note of the address of `average`, given by `&average` in the Watch view.
  2. Select **Find Address Expression...** from the **Find** menu in the Analysis window to display the Enter Value dialog box.
  3. Enter the address of `average`, **0xA1D8** in this example.
  4. Click **Find**. The dialog box closes, and the first element with the address `0xA1D8` is selected.
  5. Scroll up through the trace until you see the start of the `GetAverage()` function.
  6. The traced data accesses for `GetAverage()` show all the reads from the `input[]` array, ending in a write to `average`.

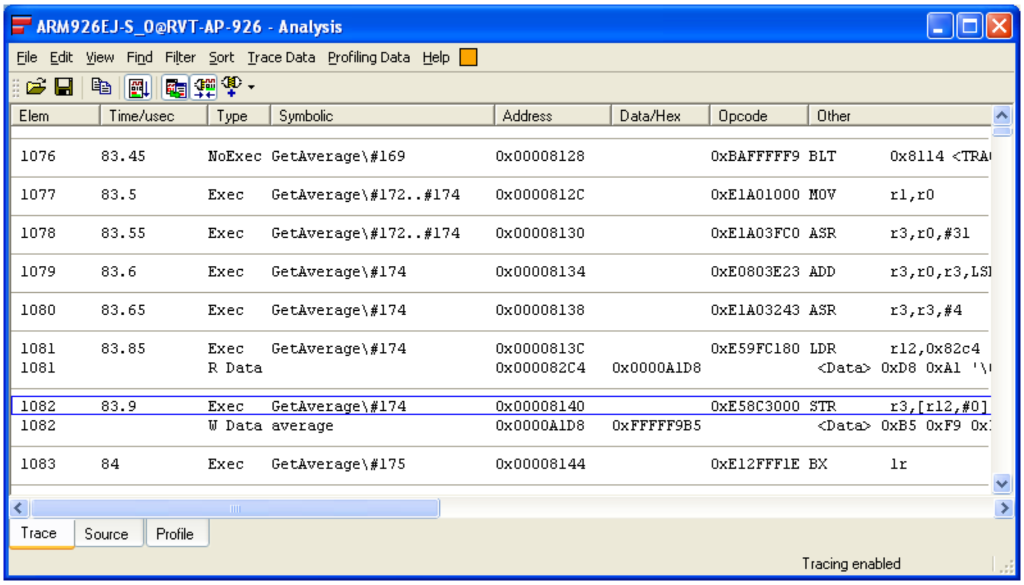
See also

- *Capturing instructions and data* on page 6-3
- *Finding information in captured trace* on page 9-16
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — Chapter 2 *RealView Debugger Commands* for details of the `ANALYZER`, `ETM_CONFIG`, and `TRACEBUFFER` commands.

10.9.4 Displaying instruction boundaries

You can display a graphical view of the instruction boundaries, which makes the captured trace easier to read. You can now see which instruction made the data access. Although you can display instruction boundaries for trace that captures only instructions, this feature is more useful when tracing data accesses.

To display the instruction boundaries graphically, select **Instruction Boundaries** from the **Trace Data** menu in the Analysis window. Figure 10-50 on page 10-45 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Elem	Time/usec	Type	Symbolic	Address	Data/Hex	Opcode	Other
1076	83.45	NoExec	GetAverage\#169	0x00008128		0xBAFFFFF9 BLT	0x8114 <TRA
1077	83.5	Exec	GetAverage\#172..#174	0x0000812C		0xE1A01000 MOV	r1,r0
1078	83.55	Exec	GetAverage\#172..#174	0x00008130		0xE1A03FC0 ASR	r3,r0,#31
1079	83.6	Exec	GetAverage\#174	0x00008134		0xE0803E23 ADD	r3,r0,r3,LS
1080	83.65	Exec	GetAverage\#174	0x00008138		0xE1A03243 ASR	r3,r3,#4
1081	83.85	Exec	GetAverage\#174	0x0000813C		0xE59FC180 LDR	r12,0x82c4
1081		R Data		0x000082C4	0x0000A1D8		<Data> 0xD8 0xA1 '\
1082	83.9	Exec	GetAverage\#174	0x00008140		0xE58C3000 STR	r3,[r12,#0]
1082		W Data	average	0x0000A1D8	0xFFFFF9B5		<Data> 0xB5 0xF9 0x
1083	84	Exec	GetAverage\#175	0x00008144		0xE12FFF1E BX	lr

Figure 10-50 Instruction boundaries in the trace view

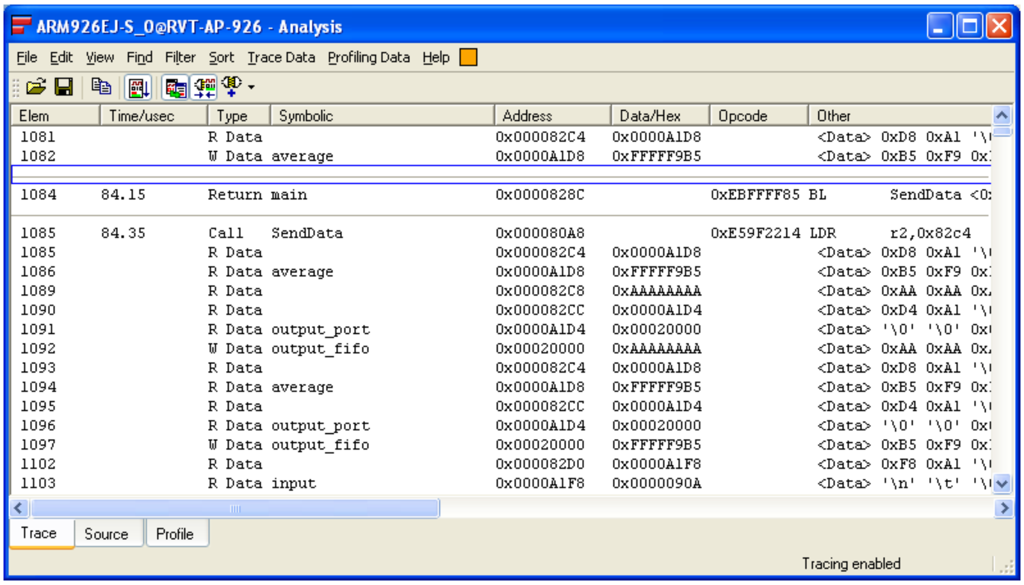
See also

- *Capturing instructions and data* on page 6-3
- *Displaying instruction boundaries* on page 9-7
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `ANALYZER`, `ETM_CONFIG`, and `TRACEBUFFER` commands.

10.9.5 Displaying function boundaries

You can display a graphical view of the function boundaries in the Trace view, to make the data accesses more readable:

1. Select **Instruction Boundaries** from the **Trace Data** menu to hide the instruction boundaries.
2. Select **Instructions** from the **Trace Data** menu to hide the traced instructions. Only the traced data accesses are displayed.
3. Select **Function Boundaries** from the **Trace Data** menu to view the function boundaries. Figure 10-51 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem    Time/usec  Type        Symbolic        Address     Data/Hex    Opcode          Other
1081               R Data                      0x000082C4  0x0000A1D8                  <Data> 0xD8 0xA1 '\
1082               W Data average              0x0000A1D8  0xFFFFF9B5                  <Data> 0xB5 0xF9 0x

1084    84.15      Return main                 0x0000828C              0xEBFFFF85 BL       SendData <0

1085    84.35      Call   SendData             0x000080A8              0xE59F2214 LDR      r2,0x82c4
1085               R Data                      0x000082C4  0x0000A1D8                  <Data> 0xD8 0xA1 '\
1086               R Data average              0x0000A1D8  0xFFFFF9B5                  <Data> 0xB5 0xF9 0x
1089               R Data                      0x000082C8  0xAAAAAAAA                  <Data> 0xAA 0xAA 0x
1090               R Data                      0x000082CC  0x0000A1D4                  <Data> 0xD4 0xA1 '\
1091               R Data output_port          0x0000A1D4  0x00020000                  <Data> '\0' '\0' 0x
1092               W Data output_fifo          0x00020000  0xAAAAAAAA                  <Data> 0xAA 0xAA 0x
1093               R Data                      0x000082C4  0x0000A1D8                  <Data> 0xD8 0xA1 '\
1094               R Data average              0x0000A1D8  0xFFFFF9B5                  <Data> 0xB5 0xF9 0x
1095               R Data                      0x000082CC  0x0000A1D4                  <Data> 0xD4 0xA1 '\
1096               R Data output_port          0x0000A1D4  0x00020000                  <Data> '\0' '\0' 0x
1097               W Data output_fifo          0x00020000  0xFFFFF9B5                  <Data> 0xB5 0xF9 0x
1102               R Data                      0x000082D0  0x0000A1F8                  <Data> 0xF8 0xA1 '\
1103               R Data input                0x0000A1F8  0x0000090A                  <Data> '\n' '\t' '\
```

Figure 10-51 Function boundaries in the trace view

See also

- *Capturing instructions and data* on page 6-3
- *Displaying function boundaries* on page 9-7
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — Chapter 2 *RealView Debugger Commands* for details of the `ANALYZER`, `ETM_CONFIG`, and `TRACEBUFFER` commands.

10.9.6 Displaying inferred registers

RealView Debugger enables you to view the values that the registers held at each instruction in the trace output. The values are derived from each trace sample, so not all register values can be inferred. The registers are interleaved with the trace output, and the instruction boundaries are also displayed.

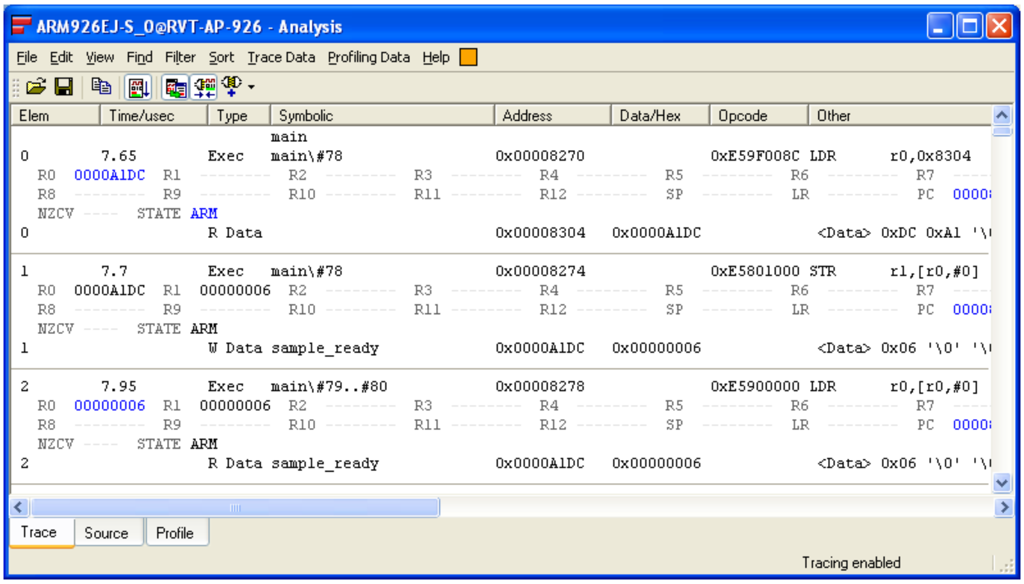
To display inferred registers:

1. Select **Instructions** from the **Trace Data** menu to view the traced instructions.

   Note

   Traced instructions must be visible to view inferred registers. When traced instructions are hidden, the traced instructions can also be displayed if you view the instruction boundaries.

2. Select **Inferred Registers** from the **Trace Data** menu. The inferred registers and instruction boundaries are displayed. Figure 10-52 shows an example


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem      Time/usec     Type     Symbolic                     Address        Data/Hex      Opcode        Other
                                 main
0         7.65          Exec     main\#78                     0x00008270                   0xE59F008C LDR      r0,0x8304
   R0   0000A1DC  R1   --------  R2   --------  R3   --------  R4   --------  R5   --------  R6   --------  R7   ----
   R8   --------  R9   --------  R10  --------  R11  --------  R12  --------  SP   --------  LR   --------  PC   0000
   NZCV ----   STATE ARM
0                       R Data                                0x00008304     0x0000A1DC                 <Data> 0xDC 0xA1 '\
1         7.7           Exec     main\#78                     0x00008274                   0xE5801000 STR      r1,[r0,#0]
   R0   0000A1DC  R1   00000006  R2   --------  R3   --------  R4   --------  R5   --------  R6   --------  R7   ----
   R8   --------  R9   --------  R10  --------  R11  --------  R12  --------  SP   --------  LR   --------  PC   0000
   NZCV ----   STATE ARM
1                       W Data sample_ready                   0x0000A1DC     0x00000006                 <Data> 0x06 '\0' '\
2         7.95          Exec     main\#79..#80                0x00008278                   0xE5900000 LDR      r0,[r0,#0]
   R0   00000006  R1   00000006  R2   --------  R3   --------  R4   --------  R5   --------  R6   --------  R7   ----
   R8   --------  R9   --------  R10  --------  R11  --------  R12  --------  SP   --------  LR   --------  PC   0000
   NZCV ----   STATE ARM
2                       R Data sample_ready                   0x0000A1DC     0x00000006                 <Data> 0x06 '\0' '\
```

Figure 10-52 Inferred registers in the Trace tab

3. The trace sample with the element number 0 shows:
   - the traced instruction
   - the register values derived from the trace
   - the associated data accesses.

See also

- *Capturing instructions and data* on page 6-3
- *Displaying inferred registers* on page 9-8
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — Chapter 2 *RealView Debugger Commands* for details of the ANALYZER, ETM_CONFIG, and TRACEBUFFER commands.

10.10 Tracing data with trace ranges

If you suspect that a certain part of program is accessing incorrect data, then you can focus the tracing of data accesses to that part of your program. To do this, you can use one or more trace ranges to capture the data accesses.

See also:

- *Tracing features used*
- *Procedure*
- *Examining the captured trace* on page 10-50.

10.10.1 Tracing features used

The following tracing features are used in this procedure:

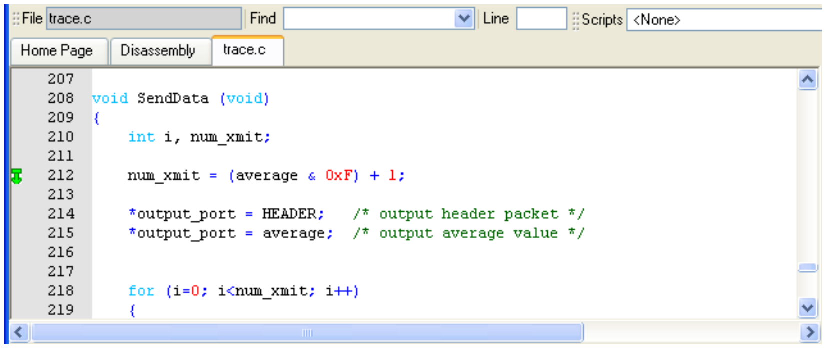
- a Start of Trace Range to specify the location where trace capture is to begin
- an End of Trace Range to specify the location where trace capture is to stop
- capture of data accesses and addresses (the default)
- 16-bit data width to minimize FIFO overflow
- filtering for a specific address
- improve the readability of the captured trace by viewing:
  - instruction boundaries
  - function boundaries.

Discontinuities in the captured trace are identified by the message `Warning: Trace Pause`. This message shows where instruction executions and possible data accesses occur, but are not captured because of the current trace conditions.

10.10.2 Procedure

To capture data trace with a trace range:

1. Reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Reload** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`).
2. Set a Start of Trace Range for instructions and data at line 210 in the source:
   a. In the Code window, click the **trace.c** tab in the Code window.
   b. Scroll down until line 210 is visible.
   c. Right-click in the gray margin at line 210 to display the context menu.
   d. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.
   e. Select **Start of Trace Range (Instruction and Data)** from the list of tracepoint types.
   f. Click **OK**. The dialog box closes, and a Start of Trace Range tracepoint is set as indicated by the green arrow. Figure 10-53 on page 10-49 shows an example. The tracepoint is set at line 212, because this is where the code for `SendData()` starts.



```
207
208 void SendData (void)
209 {
210     int i, num_xmit;
211
212     num_xmit = (average & 0xF) + 1;
213
214     *output_port = HEADER;   /* output header packet */
215     *output_port = average;  /* output average value */
216
217
218     for (i=0; i<num_xmit; i++)
219     {
```

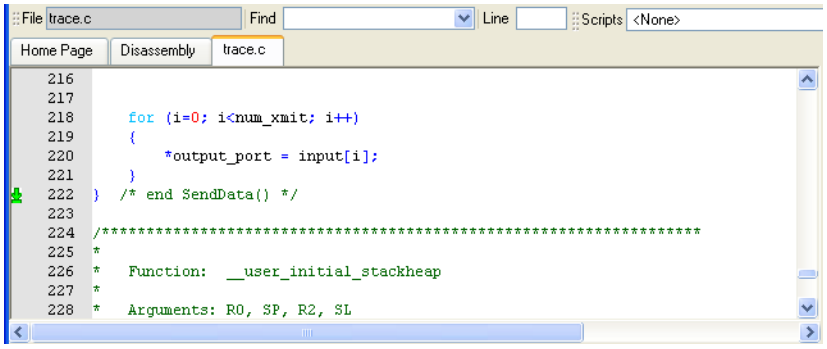
Figure 10-53 Start of Trace Range set

3. Set an End of Trace Range for instructions and data at line 222 in the source:
    a. In the Code window, click the **trace.c** tab in the Code window.
    b. Scroll down until line 222 is visible.
    c. Right-click in the gray margin at line 222 to display the context menu.
    d. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.
    e. Select **End of Trace Range (Instruction and Data)** from the list of tracepoint types.

    Note

    This is the only trace range option available, because you must complete the current trace range before another trace range can be set.

    f. Click **OK**. The dialog box closes, and the End of Trace Range tracepoint is set as indicated by the green arrow. Figure 10-54 shows an example:



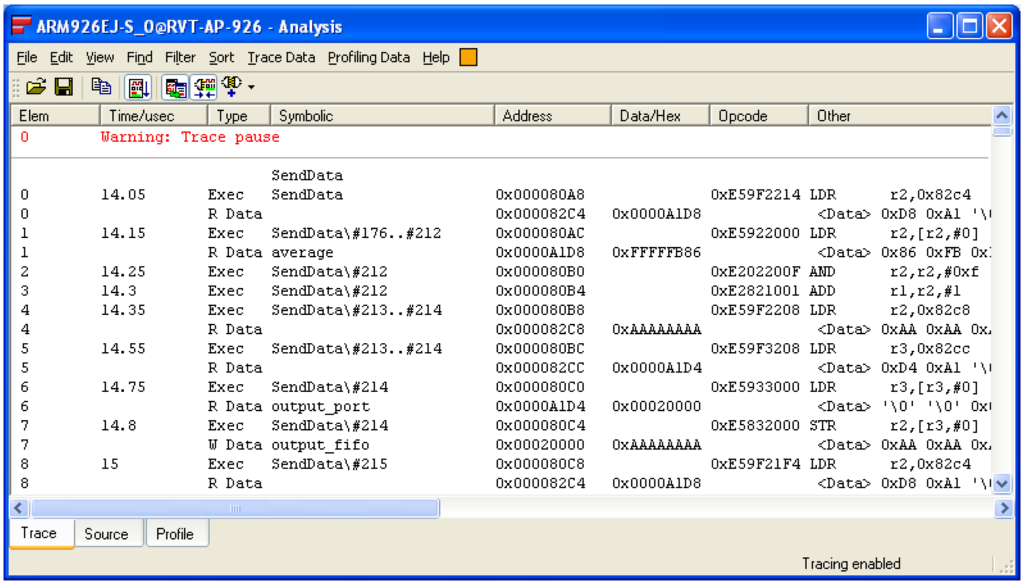
```
216
217
218     for (i=0; i<num_xmit; i++)
219     {
220         *output_port = input[i];
221     }
222 }  /* end SendData() */
223
224 /*********************************************************************
225 *
226 *   Function:   __user_initial_stackheap
227 *
228 *   Arguments: R0, SP, R2, SL
```

Figure 10-54 End of Trace Range set

4. Click **Run** on the Debug toolbar to start execution.

5. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window.

6. Select **Inferred Registers** from the **Trace Data** menu to hide the inferred register view. Figure 10-55 on page 10-50 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis

```
Elem     Time/usec  Type    Symbolic              Address      Data/Hex     Opcode           Other
0        Warning: Trace pause

                            SendData
0        14.05      Exec    SendData              0x000080A8                0xE59F2214 LDR     r2,0x82c4
0                   R Data                        0x000082C4   0x0000A1D8              <Data> 0xD8 0xA1 '\
1        14.15      Exec    SendData\#176..#212   0x000080AC                0xE5922000 LDR     r2,[r2,#0]
1                   R Data  average               0x0000A1D8   0xFFFFFB86              <Data> 0x86 0xFB 0x
2        14.25      Exec    SendData\#212         0x000080B0                0xE202200F AND     r2,r2,#0xf
3        14.3       Exec    SendData\#212         0x000080B4                0xE2821001 ADD     r1,r2,#1
4        14.35      Exec    SendData\#213..#214   0x000080B8                0xE59F2208 LDR     r2,0x82c8
4                   R Data                        0x000082C8   0xAAAAAAAA              <Data> 0xAA 0xAA 0x
5        14.55      Exec    SendData\#213..#214   0x000080BC                0xE59F3208 LDR     r3,0x82cc
5                   R Data                        0x000082CC   0x0000A1D4              <Data> 0xD4 0xA1 '\
6        14.75      Exec    SendData\#214         0x000080C0                0xE5933000 LDR     r3,[r3,#0]
6                   R Data  output_port           0x0000A1D4   0x00020000              <Data> '\0' '\0' 0x
7        14.8       Exec    SendData\#214         0x000080C4                0xE5832000 STR     r2,[r3,#0]
7                   W Data  output_fifo           0x00020000   0xAAAAAAAA              <Data> 0xAA 0xAA 0x
8        15         Exec    SendData\#215         0x000080C8                0xE59F21F4 LDR     r2,0x82c4
8                   R Data                        0x000082C4   0x0000A1D8              <Data> 0xD8 0xA1 '\
```

File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Figure 10-55 Captured trace in Analysis window

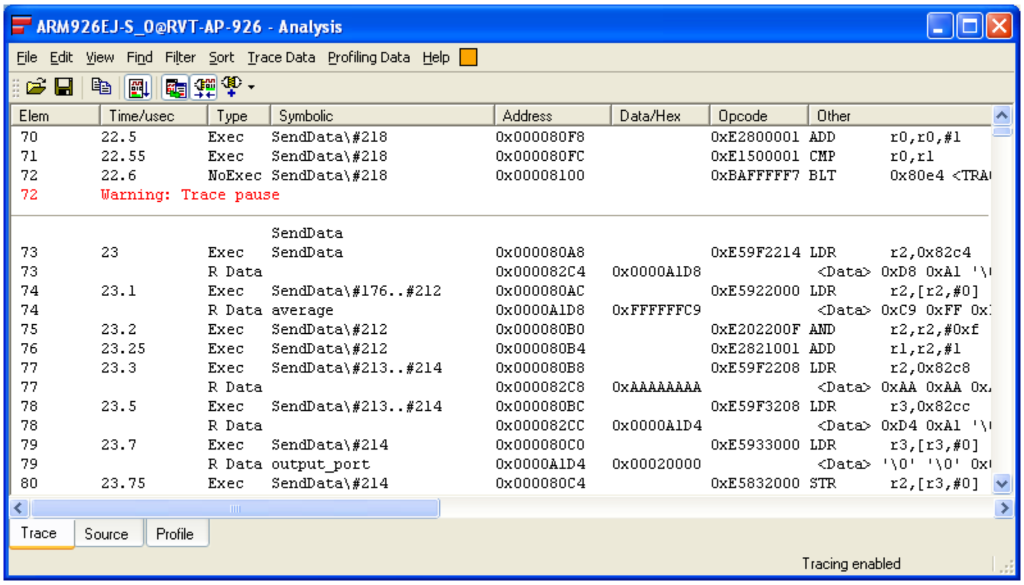
See also

- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Capturing instructions and data* on page 6-3
- *Filtering information in captured trace* on page 9-22
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `TRACE` and `TRACEBUFFER` commands.

10.10.3 Examining the captured trace

Examine the captured trace in the **Trace** tab:

- Scroll down though the captured trace until the next function boundary is visible. Figure 10-56 on page 10-51 shows an example. The trace shows the discontinuity between different runs of `SendData()`, identified by the red `Warning: Trace Pause` message.

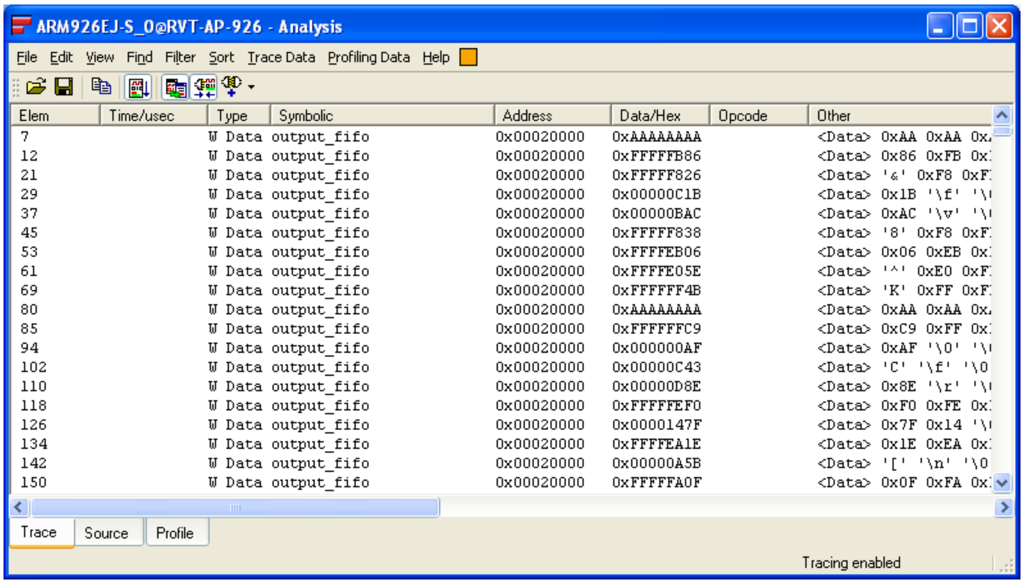

ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem    Time/usec   Type   Symbolic                 Address      Data/Hex     Opcode          Other
70      22.5        Exec   SendData\#218            0x000080F8                0xE2800001 ADD      r0,r0,#1
71      22.55       Exec   SendData\#218            0x000080FC                0xE1500001 CMP      r0,r1
72      22.6        NoExec SendData\#218            0x00008100                0xBAFFFFF7 BLT      0x80e4 <TRA
72      Warning: Trace pause

                           SendData
73      23          Exec   SendData                 0x000080A8                0xE59F2214 LDR      r2,0x82c4
73                  R Data                          0x000082C4   0x0000A1D8              <Data> 0xD8 0xA1 '\
74      23.1        Exec   SendData\#176..#212      0x000080AC                0xE5922000 LDR      r2,[r2,#0]
74                  R Data average                  0x0000A1D8   0xFFFFFFC9              <Data> 0xC9 0xFF 0x
75      23.2        Exec   SendData\#212            0x000080B0                0xE202200F AND      r2,r2,#0xf
76      23.25       Exec   SendData\#212            0x000080B4                0xE2821001 ADD      r1,r2,#1
77      23.3        Exec   SendData\#213..#214      0x000080B8                0xE59F2208 LDR      r2,0x82c8
77                  R Data                          0x000082C8   0xAAAAAAAA              <Data> 0xAA 0xAA 0x
78      23.5        Exec   SendData\#213..#214      0x000080BC                0xE59F3208 LDR      r3,0x82cc
78                  R Data                          0x000082CC   0x0000A1D4              <Data> 0xD4 0xA1 '\
79      23.7        Exec   SendData\#214            0x000080C0                0xE5933000 LDR      r3,[r3,#0]
79                  R Data output_port              0x0000A1D4   0x00020000              <Data> '\0' '\0' 0x
80      23.75       Exec   SendData\#214            0x000080C4                0xE5832000 STR      r2,[r3,#0]
```

Figure 10-56 Captured trace showing discontinuity between runs of SendData()

- Examine different runs of `SendData()`:
  1. Select **Instructions** from the **Trace Data** menu to hide the traced instructions.
  2. Scroll though the captured trace. The number of samples output for each run is pseudo randomly generated, as indicated by the different number of `input+`*`value`* elements in the trace.
- Filter the data for accesses to `*output_port` (address `0x20000`):
  1. Select **Filter on Address Expression...** from the **Filter** menu to display the Enter Value dialog box.
  2. Enter the address **0x20000**.
  3. Click **Filter**. The dialog box closes, and the filter is applied. That is, only those elements with the address `0x20000` are displayed. Figure 10-57 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem    Time/usec   Type   Symbolic                 Address      Data/Hex     Opcode          Other
7                   W Data output_fifo              0x00020000   0xAAAAAAAA              <Data> 0xAA 0xAA 0x
12                  W Data output_fifo              0x00020000   0xFFFFFB86              <Data> 0x86 0xFB 0x
21                  W Data output_fifo              0x00020000   0xFFFFF826              <Data> '&' 0xF8 0xF
29                  W Data output_fifo              0x00020000   0x00000C1B              <Data> 0x1B '\f' '\
37                  W Data output_fifo              0x00020000   0x00000BAC              <Data> 0xAC '\v' '\
45                  W Data output_fifo              0x00020000   0xFFFFF838              <Data> '8' 0xF8 0xF
53                  W Data output_fifo              0x00020000   0xFFFFEB06              <Data> 0x06 0xEB 0x
61                  W Data output_fifo              0x00020000   0xFFFFE05E              <Data> '^' 0xE0 0xF
69                  W Data output_fifo              0x00020000   0xFFFFFF4B              <Data> 'K' 0xFF 0xF
80                  W Data output_fifo              0x00020000   0xAAAAAAAA              <Data> 0xAA 0xAA 0x
85                  W Data output_fifo              0x00020000   0xFFFFFFC9              <Data> 0xC9 0xFF 0x
94                  W Data output_fifo              0x00020000   0x000000AF              <Data> 0xAF '\0' '\
102                 W Data output_fifo              0x00020000   0x00000C43              <Data> 'C' '\f' '\0
110                 W Data output_fifo              0x00020000   0x00000D8E              <Data> 0x8E '\r' '\
118                 W Data output_fifo              0x00020000   0xFFFFFEF0              <Data> 0xF0 0xFE 0x
126                 W Data output_fifo              0x00020000   0x0000147F              <Data> 0x7F 0x14 '\
134                 W Data output_fifo              0x00020000   0xFFFFEA1E              <Data> 0x1E 0xEA 0x
142                 W Data output_fifo              0x00020000   0x00000A5B              <Data> '[' '\n' '\0
150                 W Data output_fifo              0x00020000   0xFFFFFA0F              <Data> 0x0F 0xFA 0x
```

Figure 10-57 Filtered address in the Trace view

——— **Note** ———

The symbol for the data accesses is `output_fifo`. This is because the variable `output_port` used in the program is assigned to `output_fifo` at line 28 in `trace.c`.

———

RealView Debugger generates a `TRACEBUFFER` CLI command, which is also displayed in the **Cmd** tab of the Output view:

```
tracebuffer,gui,addr=0x00020000
```

4. Examine each element that has a Data/Hex value of `0xAAAAAAAA`. The number of trace samples between each of these elements is not constant.

See also

- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Capturing instructions and data* on page 6-3
- *Filtering information in captured trace* on page 9-22
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `TRACE` and `TRACEBUFFER` commands.

10.11 Tracing data accesses to a specific address

You might want to trace all data accesses to a specific address, where the access can be from anywhere in your source code.

See also:

- *Tracing features used*
- *Procedure*
- *Examining the captured trace* on page 10-55.

10.11.1 Tracing features used

The following tracing features are used in this procedure:

- setting data access tracepoints using the Create Tracepoint dialog box, which generates the following tracepoint commands:
  - `TRACEDATAACCESS`
  - `TRACEDATAREAD`
  - `TRACEDATAWRITE`.
- timestamping enabled
- filtering for a specific data value.

Discontinuities in the captured trace are identified by the message `Warning: Trace Pause`. This message shows where instruction executions and possible data accesses occur, but are not captured because of the current trace conditions.

10.11.2 Procedure

To trace all data accesses to a specific address:

1. Remove the tracepoints you set previously:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The tracepoints are removed.
2. Enable timestamping:
   a. Select **Configure Analyzer Properties...** from the **Edit** menu of the Analysis window to display the Configure ETM dialog box.
   b. Select **Enable timestamping**.
   c. Click **OK**. The Configure ETM dialog box closes. RealView Debugger generates a `ETM_CONFIG` CLI command, which is also displayed in the **Cmd** tab of the Output view:

      ```
      etm_config,syncfreq:0,packauto,time
      ```
3. Set a tracepoint to trace data accesses to `*output_port` (address `0x20000`):
   a. Select **Debug** → **Tracepoints** → **Create Tracepoint...** from the Code window main menu to display the Create Tracepoint dialog box. Figure 10-58 on page 10-54 shows an example:

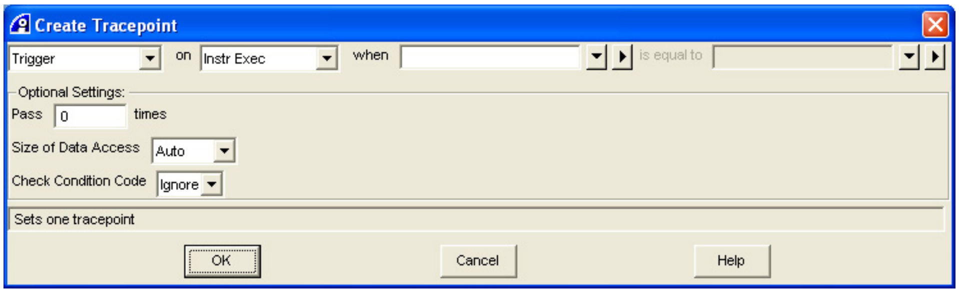


Figure 10-58 Create Tracepoint dialog box

b. Select **Trace Instr and Data** from the tracepoint type drop-down list.

c. Select **Data Access** from the tracepoint comparison type drop-down list (that is, the `on` field).

d. Enter the address to be traced (that is, the `when` field). In this case, the address to be traced corresponds to the symbol `output_port`. Therefore, enter **@trace\\output_port**.

Note

Because this is a pointer, you do not have to prefix the variable name with `&`.

e. Click **OK**. The Create Tracepoint dialog box closes, and the tracepoint is set. RealView Debugger generates a `TRACEDATAACCESS` CLI command, which is also displayed in the **Cmd** tab of the Output view:

```
trcdaccess,hw_out:"Tracepoint Type=Trace Instr and Data" @trace\\output_port
```

4. Click **Run** on the Debug toolbar to start execution.

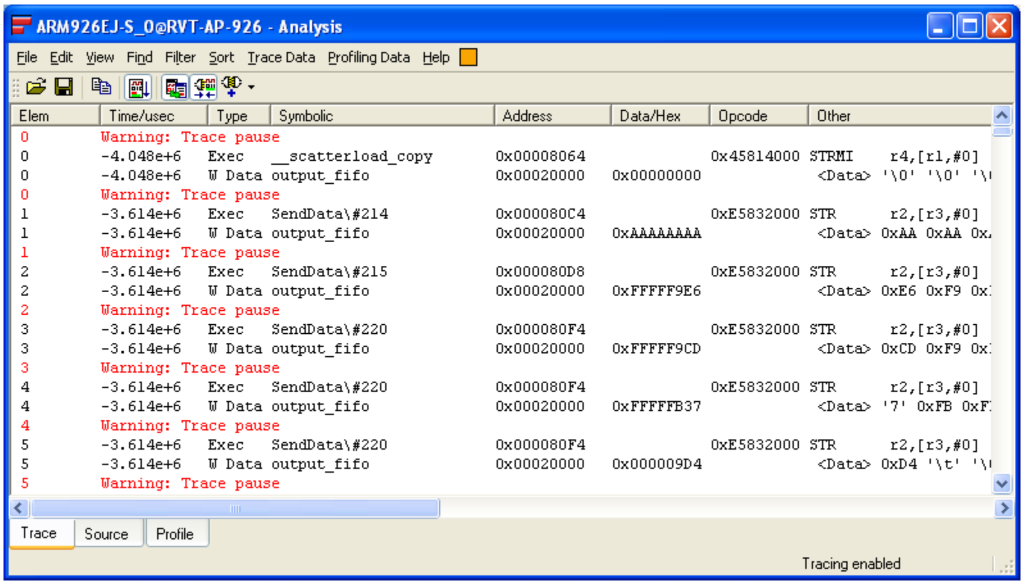
5. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window.

6. Select **Clear Filtering** from the **Filter** menu.

7. Select **Function Boundaries** from the **Trace Data** menu to hide the function boundaries. Figure 10-59 on page 10-55 shows an example.

Note

The symbol for the data accesses is `output_fifo`. This is because the variable `output_port` used in the program is assigned to `output_fifo` at line 28 in `trace.c`.


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem    Time/usec   Type    Symbolic                Address      Data/Hex     Opcode          Other
0       Warning: Trace pause
0       -4.048e+6   Exec    __scatterload_copy      0x00008064                0x45814000 STRMI    r4,[r1,#0]
0       -4.048e+6   W Data  output_fifo             0x00020000   0x00000000             <Data> '\0' '\0' '\
0       Warning: Trace pause
1       -3.614e+6   Exec    SendData\#214           0x000080C4                0xE5832000 STR      r2,[r3,#0]
1       -3.614e+6   W Data  output_fifo             0x00020000   0xAAAAAAAA             <Data> 0xAA 0xAA 0x
1       Warning: Trace pause
2       -3.614e+6   Exec    SendData\#215           0x000080D8                0xE5832000 STR      r2,[r3,#0]
2       -3.614e+6   W Data  output_fifo             0x00020000   0xFFFFF9E6             <Data> 0xE6 0xF9 0x
2       Warning: Trace pause
3       -3.614e+6   Exec    SendData\#220           0x000080F4                0xE5832000 STR      r2,[r3,#0]
3       -3.614e+6   W Data  output_fifo             0x00020000   0xFFFFF9CD             <Data> 0xCD 0xF9 0x
3       Warning: Trace pause
4       -3.614e+6   Exec    SendData\#220           0x000080F4                0xE5832000 STR      r2,[r3,#0]
4       -3.614e+6   W Data  output_fifo             0x00020000   0xFFFFFB37             <Data> '7' 0xFB 0xF
4       Warning: Trace pause
5       -3.614e+6   Exec    SendData\#220           0x000080F4                0xE5832000 STR      r2,[r3,#0]
5       -3.614e+6   W Data  output_fifo             0x00020000   0x000009D4             <Data> 0xD4 '\t' '\
5       Warning: Trace pause
```

Figure 10-59 Captured trace in Analysis window

See also

- *Capturing instructions and data* on page 6-3
- *Calculating the time between two points* on page 9-12
- *Filtering information in captured trace* on page 9-22
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `ETM_CONFIG`, `TRACE`, `TRACEBUFFER`, `TRACEDATAACCESS`, `TRACEDATAREAD`, and `TRACEDATAWRITTE` commands.

10.11.3 Examining the captured trace

Examine the captured trace in the **Trace** tab:

- Identify the sources of the data written to `output_port`:
  1. Select the trace sample with the Symbolic name `SendData\#214`. This is associated with the traced data value `0xAAAAAAAA`.
  2. Click the **trace.c** tab in the Code window. The corresponding line of source is highlighted in the source. That is:

     ```
     *output_port = HEADER;
     ```
  3. Select the next trace sample with the Symbolic name `SendData\#215`. This is associated with the traced data value `0xFFFFF9E6`. The corresponding line of source is highlighted in the **trace.c** tab. That is:

     ```
     *output_port = average;
     ```
  4. Count the number of trace samples associated with the Symbolic name `SendData\#220` until the next sample with Symbolic name of `SendData\#214` is reached.
  5. Repeat the previous step for different trace samples to see that the total number of input words associated with the Symbolic name `SendData\#220` varies.

- Calculate the time difference between output messages:
  1. Select **Instructions** from the **Trace Data** menu to hide the traced instructions.
  2. Select **Filter on Data Value...** from the **Filter** menu to display the Enter Value dialog box.
  3. Enter the value **0xAAAAAAAA**.
  4. Click **Filter**. The dialog box closes, and the filter is applied.
  5. Select the first trace sample.
  6. Right-click on the next trace sample to display the context menu.
  7. Select **Time Measure from Selected...** from the context menu to display the Prompt dialog box. This shows the time difference between the selected samples.
  8. Click **OK** to close the Prompt dialog box.
  9. Repeat these steps for subsequent trace samples, to see that the time difference is not constant.
- You can capture the same trace produced in this example with a Data Write tracepoint. You can modify the existing tracepoint to change it from a Data Access to a Data Write tracepoint:
  1. Select **Break/Tracepoints** from the **View** menu of the Code window to display the Break/Tracepoints view.
  2. Right-click on the tracepoint in the Break/Tracepoints view to display the context menu.
  3. Select **Edit...** from the context menu to display the Create Tracepoint dialog box.
  4. Select **Data Write** from the tracepoint comparison type drop-down list (that is, the `on` field).
  5. Click **OK**. The Create Tracepoint dialog box closes, and the tracepoint is modified. RealView Debugger generates a `TRACEDATAWRITE` CLI command, which is also displayed in the **Cmd** tab of the Output view:
     ```
     trcdwrite,hw_out:"Tracepoint Type=Trace Instr and Data",modify:1
     @trace\\output_port
     ```
  6. Select **Reload Image to Target** from the **Target** menu of the Code window to reload the image.
  7. Click **Run** on the Debug toolbar to start execution.
  8. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window. Compare the captured trace with that shown in Figure 10-59 on page 10-55.
- You can capture trace with a Data Read tracepoint. Repeat the previous procedure and select **Data Read** instead of **Data Write**. No trace is captured because there are no read accesses from `output_port`.
- Capture data reads from `&average`:
  1. Select **Break/Tracepoints** from the **View** menu of the Code window to display the Break/Tracepoints view.
  2. Right-click on the tracepoint in the Break/Tracepoints view to display the context menu.
  3. Select **Edit...** from the context menu to display the Create Tracepoint dialog box.
  4. Select **Data Read** from the tracepoint comparison type drop-down list (that is, the `on` field).
  5. Change the address to be traced (that is, the `when` field) to **&average**.

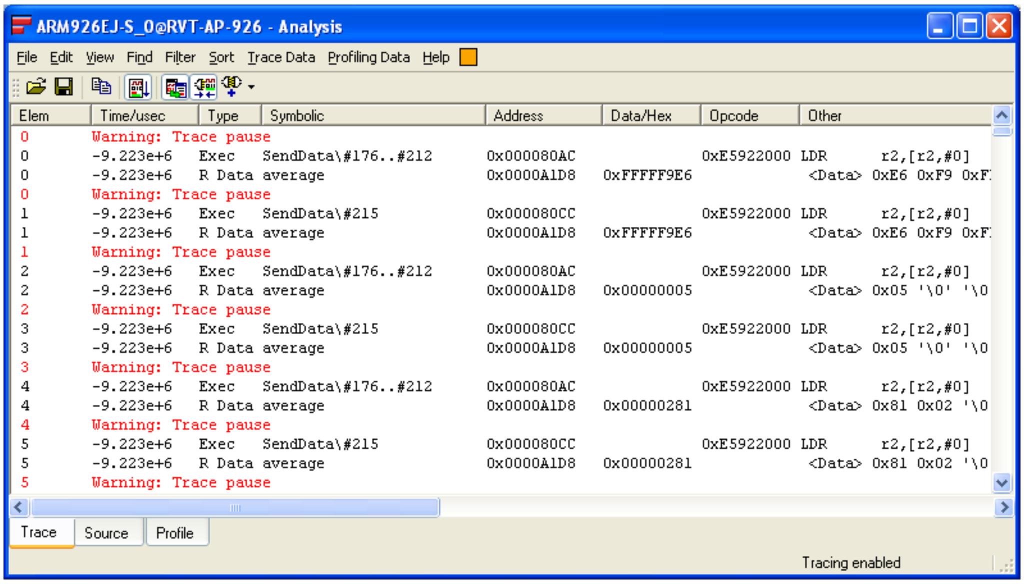
Note

Because `average` is not a pointer, you must prefix the variable name with `&`.

6. Click **OK**. The Create Tracepoint dialog box closes, and the tracepoint is modified. RealView Debugger generates a `TRACEDATAREAD` CLI command, which is also displayed in the **Cmd** tab of the Output view:

```
trcdread,hw_out:"Tracepoint Type=Trace Instr and Data",modify:1 &average
```

7. Select **Reload Image to Target** from the **Target** menu of the Code window to reload the image.
8. Click **Run** on the Debug toolbar to start execution.
9. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window.
10. Select **Clear Filtering** from the **Filter** menu.
11. Select **Instructions** from the **Trace Data** menu to show the traced instructions. Figure 10-60 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem    Time/usec    Type     Symbolic                 Address       Data/Hex      Opcode          Other
0       Warning: Trace pause
0       -9.223e+6    Exec     SendData\#176..#212      0x000080AC                  0xE5922000 LDR      r2,[r2,#0]
0       -9.223e+6    R Data average                    0x0000A1D8    0xFFFFF9E6                    <Data> 0xE6 0xF9 0xF
0       Warning: Trace pause
1       -9.223e+6    Exec     SendData\#215            0x000080CC                  0xE5922000 LDR      r2,[r2,#0]
1       -9.223e+6    R Data average                    0x0000A1D8    0xFFFFF9E6                    <Data> 0xE6 0xF9 0xF
1       Warning: Trace pause
2       -9.223e+6    Exec     SendData\#176..#212      0x000080AC                  0xE5922000 LDR      r2,[r2,#0]
2       -9.223e+6    R Data average                    0x0000A1D8    0x00000005                    <Data> 0x05 '\0' '\0
2       Warning: Trace pause
3       -9.223e+6    Exec     SendData\#215            0x000080CC                  0xE5922000 LDR      r2,[r2,#0]
3       -9.223e+6    R Data average                    0x0000A1D8    0x00000005                    <Data> 0x05 '\0' '\0
3       Warning: Trace pause
4       -9.223e+6    Exec     SendData\#176..#212      0x000080AC                  0xE5922000 LDR      r2,[r2,#0]
4       -9.223e+6    R Data average                    0x0000A1D8    0x00000281                    <Data> 0x81 0x02 '\0
4       Warning: Trace pause
5       -9.223e+6    Exec     SendData\#215            0x000080CC                  0xE5922000 LDR      r2,[r2,#0]
5       -9.223e+6    R Data average                    0x0000A1D8    0x00000281                    <Data> 0x81 0x02 '\0
5       Warning: Trace pause
```

Figure 10-60 Captured trace of data reads from average

Only the two instructions corresponding to lines 212 and 215 in `SendData()` are captured. These are the instructions that read `&average`.

Each pair of instruction and data read samples is separated by a trace discontinuity. These discontinuities show where instruction executions and data writes occur, but are not captured because of the current trace conditions.

See also

- *Capturing instructions and data* on page 6-3
- *Calculating the time between two points* on page 9-12
- *Filtering information in captured trace* on page 9-22
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `ETM_CONFIG`, `TRACE`, `TRACEBUFFER`, `TRACEDATAACCESS`, `TRACEDATAREAD`, and `TRACEDATAWRITTE` commands.

10.12 Tracing data accesses to addresses in an address range

You might want to check data accesses over a range of addresses, for example, the address range occupied by an array. RealView Debugger enables you to do this in a number of ways. Although the method described here is not the quickest method, it introduces you to additional features that are available.

See also:

- *Tracing features used*
- *Procedure*
- *Examining the captured trace* on page 10-60.

10.12.1 Tracing features used

The following tracing features are used in this procedure:

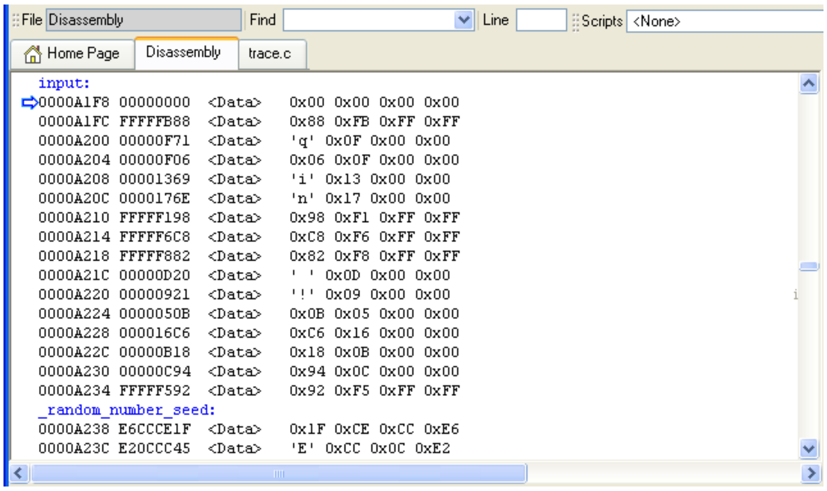
- setting a trace range for a selected area of disassembly
- editing tracepoints using the Create Tracepoint dialog box
- viewing the source corresponding to the captured trace

Discontinuities in the captured trace are identified by the message `Warning: Trace Pause`. This message shows where instruction executions and possible data accesses occur, but are not captured because of the current trace conditions.

10.12.2 Procedure

To trace data accesses to addresses in an address range:

1. Remove the tracepoints you set previously:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The tracepoints are removed.
2. Locate the address of `input[]` in the disassembly view:
   a. Click the **Disassembly** tab in the Code window to display the disassembly view.
   b. In the **Cmd** tab of the Output view, enter the command:
      ```
      printsym input
      ```
      The details of the symbol `input` are displayed, which includes the address range for the array, for example:
      ```
       @trace\\input       : Global int[16].
                             Address = 0x0000A1F8 to 0x0000A237
      ```
      The addresses might be different depending on the target you are using.
   c. Right-click on the white background of the **Disassembly** tab to display the context menu.
   d. Select **Locate Address...** from the context menu to display the Prompt dialog box.
   e. Enter the start address of the `input[]` array, **0xA1F8**.
   f. Click **OK**. The dialog box closes, and the disassembly view changes to show the disassembly from the address `0xA1F8`. Figure 10-61 on page 10-59 shows an example:



```
input:
0000A1F8 00000000  <Data>   0x00 0x00 0x00 0x00
0000A1FC FFFFFB88  <Data>   0x88 0xFB 0xFF 0xFF
0000A200 00000F71  <Data>   'q' 0x0F 0x00 0x00
0000A204 00000F06  <Data>   0x06 0x0F 0x00 0x00
0000A208 00001369  <Data>   'i' 0x13 0x00 0x00
0000A20C 0000176E  <Data>   'n' 0x17 0x00 0x00
0000A210 FFFFF198  <Data>   0x98 0xF1 0xFF 0xFF
0000A214 FFFFF6C8  <Data>   0xC8 0xF6 0xFF 0xFF
0000A218 FFFFF882  <Data>   0x82 0xF8 0xFF 0xFF
0000A21C 00000D20  <Data>   ' ' 0x0D 0x00 0x00
0000A220 00000921  <Data>   '!' 0x09 0x00 0x00
0000A224 0000050B  <Data>   0x0B 0x05 0x00 0x00
0000A228 000016C6  <Data>   0xC6 0x16 0x00 0x00
0000A22C 00000B18  <Data>   0x18 0x0B 0x00 0x00
0000A230 00000C94  <Data>   0x94 0x0C 0x00 0x00
0000A234 FFFFF592  <Data>   0x92 0xF5 0xFF 0xFF
_random_number_seed:
0000A238 E6CCCE1F  <Data>   0x1F 0xCE 0xCC 0xE6
0000A23C E20CCC45  <Data>   'E' 0xCC 0x0C 0xE2
```

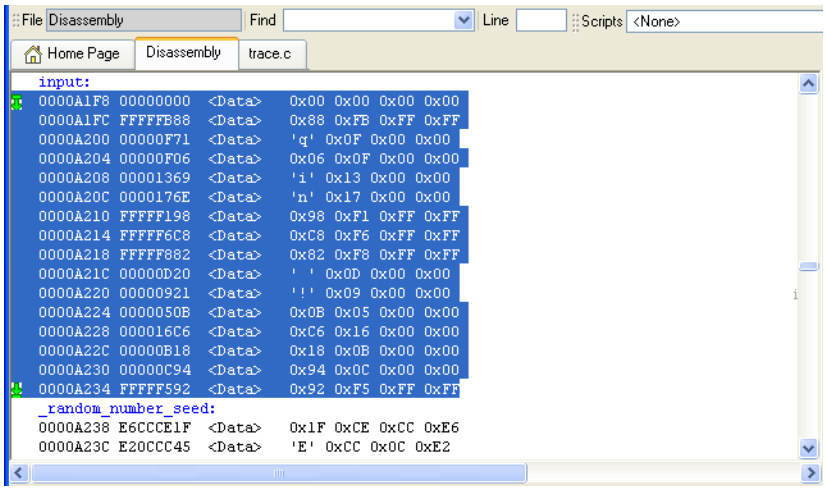
Figure 10-61 Disassembly view of the input[] array

3. Set a trace range for the range of addresses occupied by the `input[]` array in the disassembly view:

   a. Select the data values between addresses `0xA1F8` and `0xA237`. All the disassembly between the two addresses is highlighted.

   b. Right-click on the margin of the selected area to display the context menu.

   c. Select **Set Trace Range** from the context menu. An instruction execution trace range is set between addresses `0xA0F0` and `0xA0BF`. Figure 10-62 shows an example:



```
input:
0000A1F8 00000000  <Data>   0x00 0x00 0x00 0x00
0000A1FC FFFFFB88  <Data>   0x88 0xFB 0xFF 0xFF
0000A200 00000F71  <Data>   'q' 0x0F 0x00 0x00
0000A204 00000F06  <Data>   0x06 0x0F 0x00 0x00
0000A208 00001369  <Data>   'i' 0x13 0x00 0x00
0000A20C 0000176E  <Data>   'n' 0x17 0x00 0x00
0000A210 FFFFF198  <Data>   0x98 0xF1 0xFF 0xFF
0000A214 FFFFF6C8  <Data>   0xC8 0xF6 0xFF 0xFF
0000A218 FFFFF882  <Data>   0x82 0xF8 0xFF 0xFF
0000A21C 00000D20  <Data>   ' ' 0x0D 0x00 0x00
0000A220 00000921  <Data>   '!' 0x09 0x00 0x00
0000A224 0000050B  <Data>   0x0B 0x05 0x00 0x00
0000A228 000016C6  <Data>   0xC6 0x16 0x00 0x00
0000A22C 00000B18  <Data>   0x18 0x0B 0x00 0x00
0000A230 00000C94  <Data>   0x94 0x0C 0x00 0x00
0000A234 FFFFF592  <Data>   0x92 0xF5 0xFF 0xFF
_random_number_seed:
0000A238 E6CCCE1F  <Data>   0x1F 0xCE 0xCC 0xE6
0000A23C E20CCC45  <Data>   'E' 0xCC 0x0C 0xE2
```

Figure 10-62 Trace range set in the disassembly view

   RealView Debugger generates a `TRACE` CLI command, which is also displayed in the **Cmd** tab of the Output view, for example:

   ```
   trace,range 0x0000A1F8..0x0000A234
   ```

4. Modify the instruction execution tracepoint to be a data read tracepoint:

   a. Select **Break/Tracepoints** from the **View** menu of the Code window to display the Break/Tracepoints view.

   b. Right-click on the tracepoint in the Break/Tracepoints view to display the context menu.

   c. Select **Edit...** from the context menu to display the Create Tracepoint dialog box.

   d. Select **Trace Instr and Data** from the tracepoint type drop-down list.

e. Select **Data Read** from the tracepoint comparison type drop-down list (that is, the `on` field).

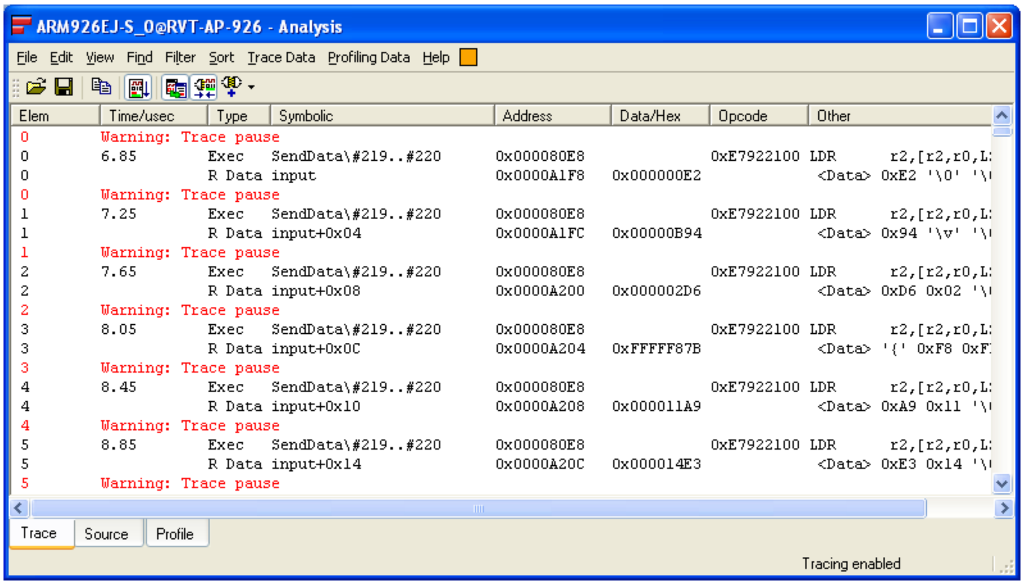
f. Change the end of range address to **0x0000A237**.

g. Click **OK**. The Create Tracepoint dialog box closes, and the tracepoint is modified. RealView Debugger generates a `TRACEDATAREAD` CLI command, which is also displayed in the **Cmd** tab of the Output view:

```
trcdread,hw_out:"Tracepoint Type=Trace Instr and Data",hw_in:"Size of Data
Access=Byte",modify:1 0x0000a1f8..0x0000a237
```

5. Click **Run** on the Debug toolbar to start execution.

6. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-63 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem    Time/usec   Type    Symbolic               Address      Data/Hex     Opcode      Other
0       Warning: Trace pause
0       6.85        Exec    SendData\#219..#220    0x000080E8                0xE7922100 LDR      r2,[r2,r0,L
0                   R Data input                   0x0000A1F8   0x000000E2               <Data> 0xE2 '\0' '\
0       Warning: Trace pause
1       7.25        Exec    SendData\#219..#220    0x000080E8                0xE7922100 LDR      r2,[r2,r0,L
1                   R Data input+0x04              0x0000A1FC   0x00000B94               <Data> 0x94 '\v' '\
1       Warning: Trace pause
2       7.65        Exec    SendData\#219..#220    0x000080E8                0xE7922100 LDR      r2,[r2,r0,L
2                   R Data input+0x08              0x0000A200   0x000002D6               <Data> 0xD6 0x02 '\
2       Warning: Trace pause
3       8.05        Exec    SendData\#219..#220    0x000080E8                0xE7922100 LDR      r2,[r2,r0,L
3                   R Data input+0x0C              0x0000A204   0xFFFFF87B               <Data> '{' 0xF8 0xF
3       Warning: Trace pause
4       8.45        Exec    SendData\#219..#220    0x000080E8                0xE7922100 LDR      r2,[r2,r0,L
4                   R Data input+0x10              0x0000A208   0x000011A9               <Data> 0xA9 0x11 '\
4       Warning: Trace pause
5       8.85        Exec    SendData\#219..#220    0x000080E8                0xE7922100 LDR      r2,[r2,r0,L
5                   R Data input+0x14              0x0000A20C   0x000014E3               <Data> 0xE3 0x14 '\
5       Warning: Trace pause
```

Figure 10-63 Captured trace of data reads from the input[] array

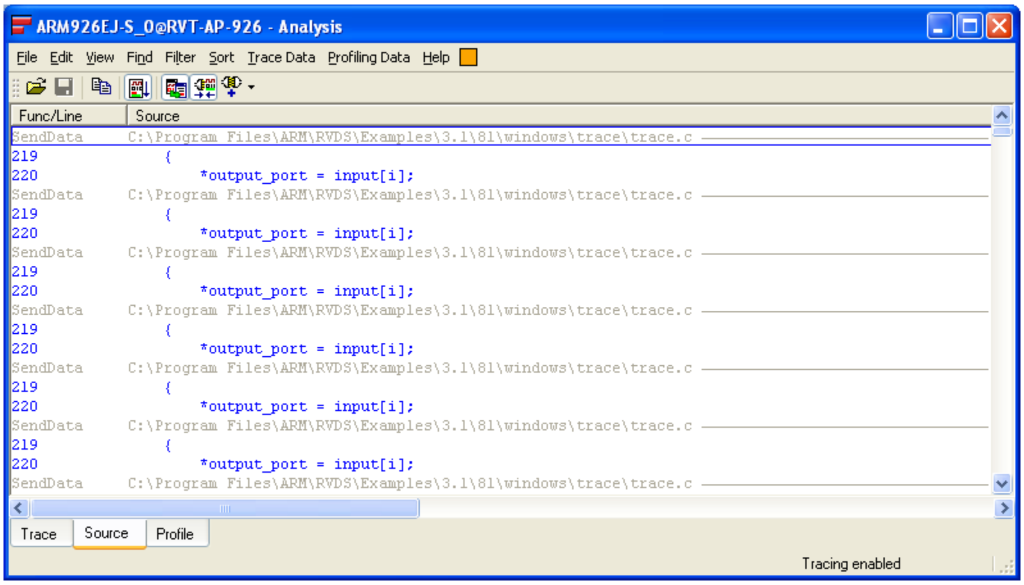
See also

- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Capturing instructions and data* on page 6-3
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `PRINTSYMBOLS`, `TRACE`, and `TRACEDATAREAD` commands.

10.12.3 Examining the captured trace

Examine the captured trace in the **Trace** tab:

- Only read accesses to the `input[]` array are traced, which occur at lines 143, 147, 171 and 220 in the source code `trace.c`.
- Click the **Source** tab in the Analysis window. Figure 10-64 on page 10-61 shows an example. Here you can see the program flow corresponding to the captured trace.


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Func/Line
Source
Trace
Source
Profile
Tracing enabled

```
SendData    C:\Program Files\ARM\RVDS\Examples\3.1\81\windows\trace\trace.c
219             {
220                 *output_port = input[i];
SendData    C:\Program Files\ARM\RVDS\Examples\3.1\81\windows\trace\trace.c
219             {
220                 *output_port = input[i];
SendData    C:\Program Files\ARM\RVDS\Examples\3.1\81\windows\trace\trace.c
219             {
220                 *output_port = input[i];
SendData    C:\Program Files\ARM\RVDS\Examples\3.1\81\windows\trace\trace.c
219             {
220                 *output_port = input[i];
SendData    C:\Program Files\ARM\RVDS\Examples\3.1\81\windows\trace\trace.c
219             {
220                 *output_port = input[i];
SendData    C:\Program Files\ARM\RVDS\Examples\3.1\81\windows\trace\trace.c
219             {
220                 *output_port = input[i];
SendData    C:\Program Files\ARM\RVDS\Examples\3.1\81\windows\trace\trace.c
```

Figure 10-64 Source code corresponding to captured trace

See also

- *Trace Range* on page 5-5
- *Setting trace ranges in conjunction with trace start and end points* on page 5-7
- *Capturing instructions and data* on page 6-3
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `PRINTSYMBOLS`, `TRACE`, and `TRACEDATAREAD` commands.

10.13 Collecting trace on a running target

Sometimes, you might want to capture trace on a target that is already running. For example, a target that is running an OS.

See also:

- *Tracing features used*
- *Capturing trace by disconnecting and connecting the analyzer*
- *Capturing trace by disabling tracing* on page 10-63
- *Displaying capturing trace with a trigger* on page 10-64
- *Examining trace captured with a trigger* on page 10-66.

10.13.1 Tracing features used

The following tracing features are used in this procedure:

- clearing the trace buffer
- disconnecting and connecting the analyzer
- disabling and enabling tracing
- setting a trigger tracepoint
- finding the sample corresponding to the trigger in the captured trace.

10.13.2 Capturing trace by disconnecting and connecting the analyzer

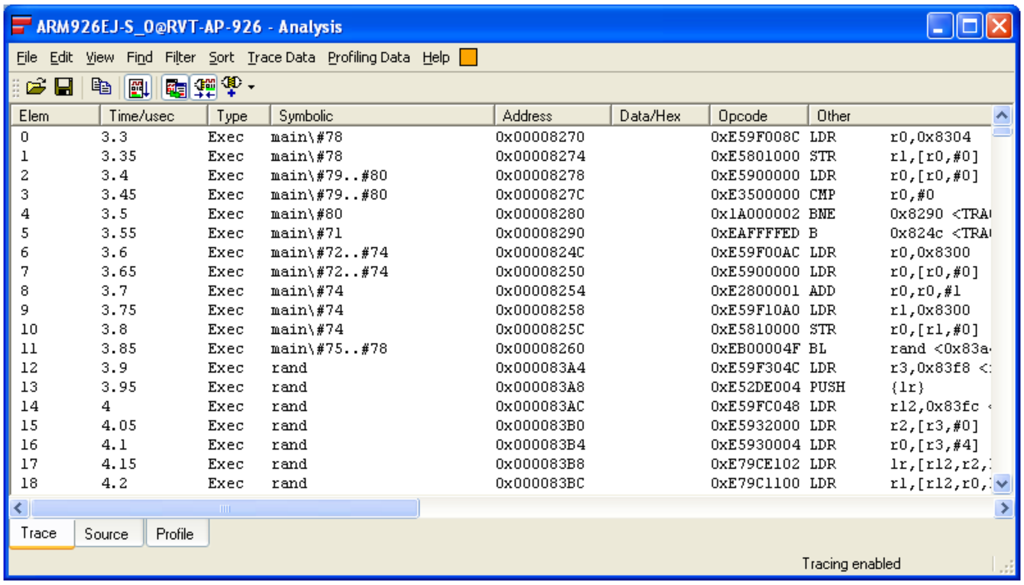
To capture trace from a running target:

1. Click **Toggle Analyzer** on the Analysis window toolbar to disconnect from the analyzer.
2. Unload and reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The tracepoint is removed.
3. Click **Run** on the Debug toolbar to start execution.
4. Click **Toggle Analyzer** on the Analysis window toolbar to connect to the analyzer. This action does not have an impact on the performance of the program.

   Note

   You cannot use the `ANALYZER,connect` CLI command to connect to the analyzer when the target is running.

5. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-65 on page 10-63 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Elem	Time/usec	Type	Symbolic	Address	Data/Hex	Opcode	Other
0	3.3	Exec	main\#78	0x00008270		0xE59F008C LDR	r0,0x8304
1	3.35	Exec	main\#78	0x00008274		0xE5801000 STR	r1,[r0,#0]
2	3.4	Exec	main\#79..#80	0x00008278		0xE5900000 LDR	r0,[r0,#0]
3	3.45	Exec	main\#79..#80	0x0000827C		0xE3500000 CMP	r0,#0
4	3.5	Exec	main\#80	0x00008280		0x1A000002 BNE	0x8290 <TRA
5	3.55	Exec	main\#71	0x00008290		0xEAFFFFED B	0x824c <TRA
6	3.6	Exec	main\#72..#74	0x0000824C		0xE59F00AC LDR	r0,0x8300
7	3.65	Exec	main\#72..#74	0x00008250		0xE5900000 LDR	r0,[r0,#0]
8	3.7	Exec	main\#74	0x00008254		0xE2800001 ADD	r0,r0,#1
9	3.75	Exec	main\#74	0x00008258		0xE59F10A0 LDR	r1,0x8300
10	3.8	Exec	main\#74	0x0000825C		0xE5810000 STR	r0,[r1,#0]
11	3.85	Exec	main\#75..#78	0x00008260		0xEB00004F BL	rand <0x83a
12	3.9	Exec	rand	0x000083A4		0xE59F304C LDR	r3,0x83f8 <
13	3.95	Exec	rand	0x000083A8		0xE52DE004 PUSH	{lr}
14	4	Exec	rand	0x000083AC		0xE59FC048 LDR	r12,0x83fc
15	4.05	Exec	rand	0x000083B0		0xE5932000 LDR	r2,[r3,#0]
16	4.1	Exec	rand	0x000083B4		0xE5930004 LDR	r0,[r3,#4]
17	4.15	Exec	rand	0x000083B8		0xE79CE102 LDR	lr,[r12,r2,
18	4.2	Exec	rand	0x000083BC		0xE79C1100 LDR	r1,[r12,r0,

Figure 10-65 Captured trace from a running target

——— **Note** ———

The captured trace depends on the location that execution has reached when you connect the analyzer and the point where you stop execution.

———

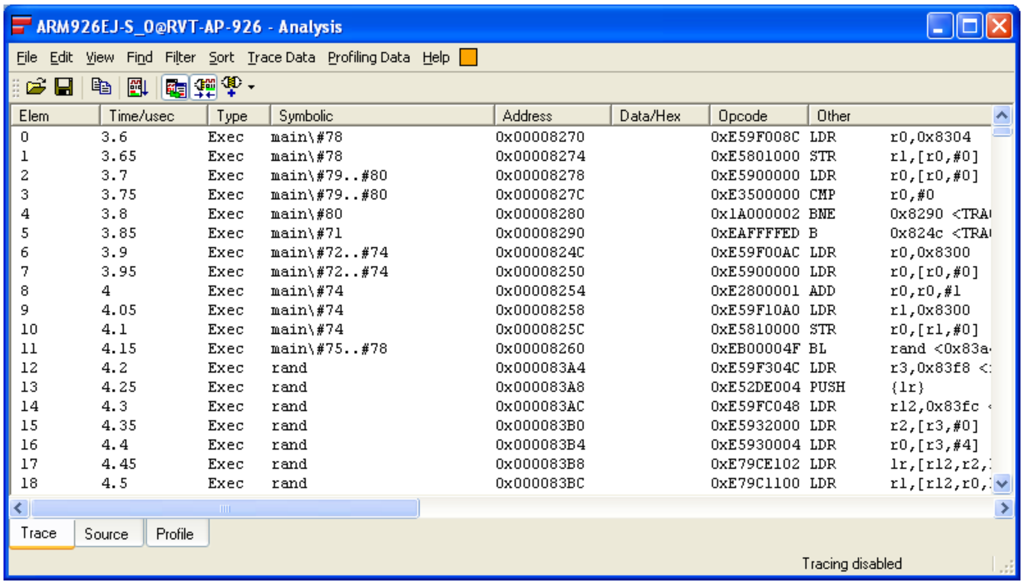
See also

- *Connecting to an analyzer* on page 2-17
- *Disconnecting from the Analyzer* on page 2-18
- *Enabling and disabling tracing* on page 2-19
- *Setting up new trace conditions on a running processor* on page 2-20
- *Trigger* on page 5-3
- *Clearing the trace buffer* on page 9-15.

10.13.3 Capturing trace by disabling tracing

To capture trace by disabling tracing:

1. Select **Clear Trace Buffer** from the **View** menu of the Analysis window. The captured trace is cleared.
2. Click **Run** on the Debug toolbar to restart execution.
3. Select **Tracing Enabled** from the **Edit** menu of the Analysis window. Trace capture is disabled, as shown in the status line at the bottom of the Analysis window. Do not stop the target running at the moment. When you disable tracing, the captured trace is displayed. Figure 10-66 on page 10-64 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile

```
Elem   Time/usec  Type  Symbolic          Address      Data/Hex   Opcode         Other
0      3.6        Exec  main\#78          0x00008270              0xE59F008C LDR    r0,0x8304
1      3.65       Exec  main\#78          0x00008274              0xE5801000 STR    r1,[r0,#0]
2      3.7        Exec  main\#79..#80     0x00008278              0xE5900000 LDR    r0,[r0,#0]
3      3.75       Exec  main\#79..#80     0x0000827C              0xE3500000 CMP    r0,#0
4      3.8        Exec  main\#80          0x00008280              0x1A000002 BNE    0x8290 <TRA
5      3.85       Exec  main\#71          0x00008290              0xEAFFFFED B      0x824c <TRA
6      3.9        Exec  main\#72..#74     0x0000824C              0xE59F00AC LDR    r0,0x8300
7      3.95       Exec  main\#72..#74     0x00008250              0xE5900000 LDR    r0,[r0,#0]
8      4          Exec  main\#74          0x00008254              0xE2800001 ADD    r0,r0,#1
9      4.05       Exec  main\#74          0x00008258              0xE59F10A0 LDR    r1,0x8300
10     4.1        Exec  main\#74          0x0000825C              0xE5810000 STR    r0,[r1,#0]
11     4.15       Exec  main\#75..#78     0x00008260              0xEB00004F BL     rand <0x83a
12     4.2        Exec  rand              0x000083A4              0xE59F304C LDR    r3,0x83f8 <
13     4.25       Exec  rand              0x000083A8              0xE52DE004 PUSH   {lr}
14     4.3        Exec  rand              0x000083AC              0xE59FC048 LDR    r12,0x83fc
15     4.35       Exec  rand              0x000083B0              0xE5932000 LDR    r2,[r3,#0]
16     4.4        Exec  rand              0x000083B4              0xE5930004 LDR    r0,[r3,#4]
17     4.45       Exec  rand              0x000083B8              0xE79CE102 LDR    lr,[r12,r2,
18     4.5        Exec  rand              0x000083BC              0xE79C1100 LDR    r1,[r12,r0,
```

Tracing disabled

Figure 10-66 Captured trace after disabling trace for a running target

The captured trace might be different to that shown here, because it depends on what has been captured at the moment tracing is disabled.

The target continues running, and you can analyze the captured trace.

You can update the captured trace again if required. To do this:

a. Select **Tracing Enabled** from the **Edit** menu of the Analysis window to enable tracing again.

b. Select **Tracing Enabled** from the **Edit** menu of the Analysis window to disable tracing. The captured trace is updated.

4. After a short time, click **Stop** on the Debug toolbar to stop execution. The captured trace remains intact.

See also

- *Connecting to an analyzer* on page 2-17
- *Disconnecting from the Analyzer* on page 2-18
- *Enabling and disabling tracing* on page 2-19
- *Setting up new trace conditions on a running processor* on page 2-20
- *Trigger* on page 5-3
- *Clearing the trace buffer* on page 9-15.

10.13.4 Displaying capturing trace with a trigger

To display the captured trace on a running target using a trigger:

1. Unload and reload the image:

   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.

   b. Select **Unload** from the context menu. The image is unloaded.

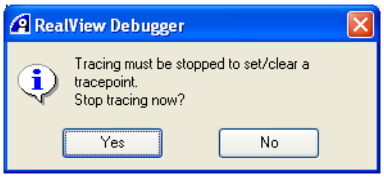
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.

   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The previous tracepoint is removed.

2. If tracing is disabled, select **Tracing Enabled** from the **Edit** menu of the Analysis window.

3. Click **Run** on the Debug toolbar to start execution.

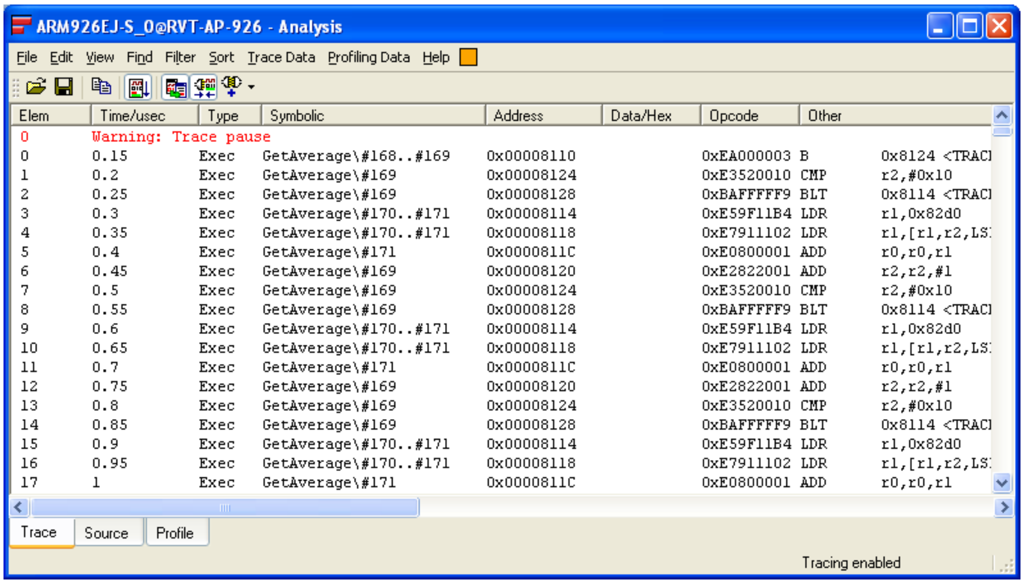
4. Set a Trigger tracepoint at line 74:
   a. Click the **trace.c** tab to display the source code.
   b. Scroll down until line 74 is visible.
   c. Right-click in the margin at line 74 to display the context menu.
   d. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.
   e. Select **Set Trigger**.
   f. Click **OK**. The New Tracepoint dialog box closes and the Prompt dialog box, is displayed. Figure 10-67 shows an example. To set a tracepoint on a running target, tracing must be disabled.



Tracing must be stopped to set/clear a tracepoint.
Stop tracing now?

Figure 10-67 Prompt to stop tracing when setting a tracepoint

5. Click **Yes** to disable tracing. The following actions occur:
   - The trigger tracepoint is set.
   - All trace captured up to the point where tracing is disabled is displayed.
   - Enable tracing again. When the trigger is activated, the captured trace is updated in the **Trace** tab of the Analysis window. Figure 10-68 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem  Time/usec  Type  Symbolic                 Address     Data/Hex  Opcode          Other
0     Warning: Trace pause
0     0.15       Exec  GetAverage\#168..#169    0x00008110            0xEA000003 B    0x8124 <TRACI
1     0.2        Exec  GetAverage\#169          0x00008124            0xE3520010 CMP  r2,#0x10
2     0.25       Exec  GetAverage\#169          0x00008128            0xBAFFFFF9 BLT  0x8114 <TRACI
3     0.3        Exec  GetAverage\#170..#171    0x00008114            0xE59F11B4 LDR  r1,0x82d0
4     0.35       Exec  GetAverage\#170..#171    0x00008118            0xE7911102 LDR  r1,[r1,r2,LS
5     0.4        Exec  GetAverage\#171          0x0000811C            0xE0800001 ADD  r0,r0,r1
6     0.45       Exec  GetAverage\#169          0x00008120            0xE2822001 ADD  r2,r2,#1
7     0.5        Exec  GetAverage\#169          0x00008124            0xE3520010 CMP  r2,#0x10
8     0.55       Exec  GetAverage\#169          0x00008128            0xBAFFFFF9 BLT  0x8114 <TRACI
9     0.6        Exec  GetAverage\#170..#171    0x00008114            0xE59F11B4 LDR  r1,0x82d0
10    0.65       Exec  GetAverage\#170..#171    0x00008118            0xE7911102 LDR  r1,[r1,r2,LS
11    0.7        Exec  GetAverage\#171          0x0000811C            0xE0800001 ADD  r0,r0,r1
12    0.75       Exec  GetAverage\#169          0x00008120            0xE2822001 ADD  r2,r2,#1
13    0.8        Exec  GetAverage\#169          0x00008124            0xE3520010 CMP  r2,#0x10
14    0.85       Exec  GetAverage\#169          0x00008128            0xBAFFFFF9 BLT  0x8114 <TRACI
15    0.9        Exec  GetAverage\#170..#171    0x00008114            0xE59F11B4 LDR  r1,0x82d0
16    0.95       Exec  GetAverage\#170..#171    0x00008118            0xE7911102 LDR  r1,[r1,r2,LS
17    1          Exec  GetAverage\#171          0x0000811C            0xE0800001 ADD  r0,r0,r1
```

Figure 10-68 Trace captured with a trigger

   - The target continues running.

See also

- *Connecting to an analyzer* on page 2-17

- *Disconnecting from the Analyzer* on page 2-18
- *Enabling and disabling tracing* on page 2-19
- *Setting up new trace conditions on a running processor* on page 2-20
- *Trigger* on page 5-3
- *Clearing the trace buffer* on page 9-15.

10.13.5 Examining trace captured with a trigger

Examine the captured trace in the **Trace** tab:

- Select **Find Trigger** from the **Find** menu in the Analysis window to locate the trigger in the captured trace:
  - — The trace message corresponding to the trigger is highlighted with a red box.
  - — The captured trace appears before the trigger, which is near the end of the trace buffer. However, there might be a small amount of trace captured after the trigger.
  - — Click the **trace.c** tab in the Code window. The corresponding line in the source code is also selected.
  - — The line number of the trigger in the captured trace might not correspond to the line number where you set the trigger. This is because there is a certain amount of skid involved between the trigger being activated and the trace being captured.
- To set a tracepoint on a running target without being prompted to stop tracing:
  1. Select **Tracing Enabled** from the **Edit** menu in the Analysis window to disable tracing. Captured trace might be displayed depending on the current trace conditions.
  2. Set the required tracepoint.
  3. Select **Tracing Enabled** from the **Edit** menu to enable tracing again.
     Captured trace might be displayed automatically depending on the trace conditions. If not, you might have to stop the target.
  4. After a short time, click **Stop** on the Debug toolbar to stop execution.

See also

- *Connecting to an analyzer* on page 2-17
- *Disconnecting from the Analyzer* on page 2-18
- *Enabling and disabling tracing* on page 2-19
- *Setting up new trace conditions on a running processor* on page 2-20
- *Trigger* on page 5-3
- *Clearing the trace buffer* on page 9-15.

10.14 Using triggers to collect trace around a specific instruction

You can collect trace when execution reaches a specific instruction, such as a call to an error handler routine or interrupt service routine. You can display the trace captured:

- before the trigger point, which is the default action
- after the trigger point
- around the trigger point.

——— **Note** ———

You can also choose to stop execution when the trigger is hit.

———————————

See also:

- *Tracing features used*
- *Capturing trace before a trigger*
- *Capturing trace around a trigger* on page 10-69
- *Capturing trace after a trigger* on page 10-71
- *Examining the captured trace* on page 10-72.

10.14.1 Tracing features used

The following tracing features are used in this procedure:

- clearing the trace buffer
- setting a trigger tracepoint
- finding the sample corresponding to the trigger in the captured trace
- setting a conditional trigger tracepoint that activates after a specific number of passes
- specifying the trigger mode:
  - Collect Trace Before Trigger (this is the default)
  - Collect Trace Around Trigger
  - Collect Trace After Trigger.

10.14.2 Capturing trace before a trigger

To display the capture trace before a trigger:

1. Unload and reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The previous tracepoint is removed.
2. Set a trigger tracepoint at line 212:
   a. Click the **trace.c** tab to display the source code.
   b. Scroll down until line 212 is visible.
   c. Right-click in the margin at line 212 to display the context menu.

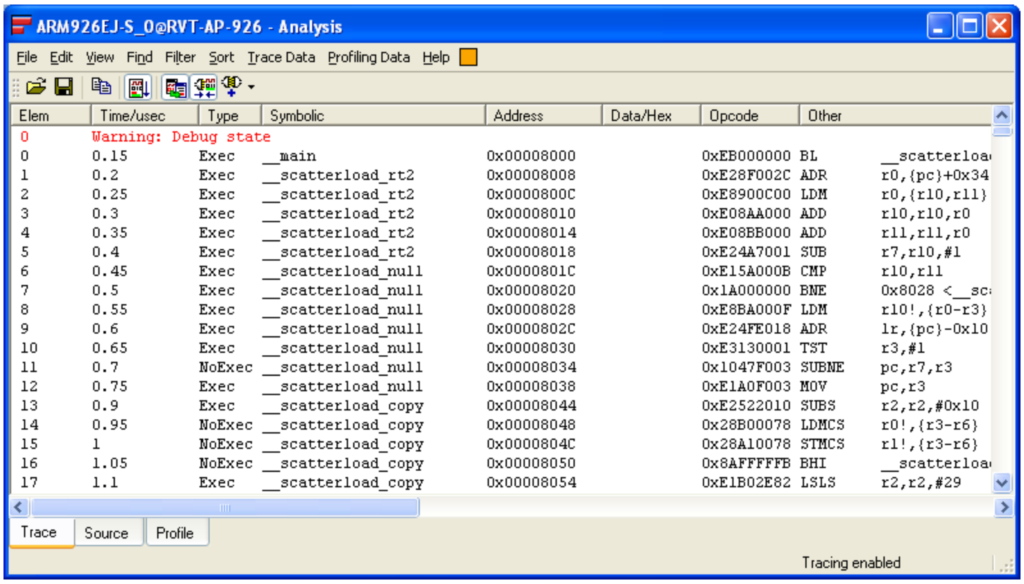
d. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

e. Select **Set Trigger**.

f. Click **OK**. The New Tracepoint dialog box closes and the tracepoint is set.

3. Click **Run** on the Debug toolbar to start execution.

4. When the trigger is activated, the captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-69 shows an example:

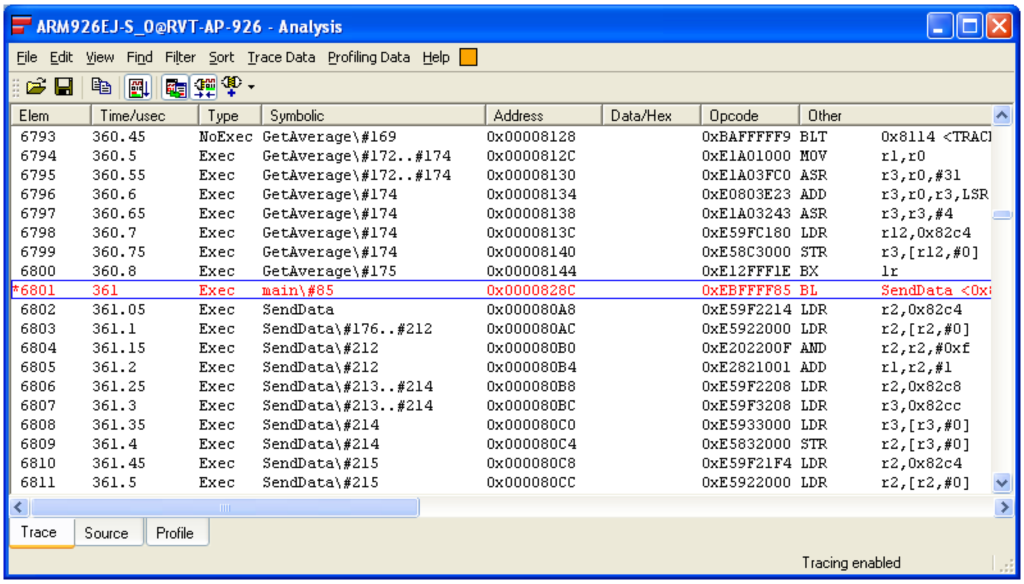

ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem  Time/usec  Type    Symbolic               Address     Data/Hex  Opcode           Other
0     Warning: Debug state
0     0.15       Exec    __main                 0x00008000            0xEB000000 BL     __scatterloa
1     0.2        Exec    __scatterload_rt2      0x00008008            0xE28F002C ADR    r0,{pc}+0x34
2     0.25       Exec    __scatterload_rt2      0x0000800C            0xE8900C00 LDM    r0,{r10,r11}
3     0.3        Exec    __scatterload_rt2      0x00008010            0xE08AA000 ADD    r10,r10,r0
4     0.35       Exec    __scatterload_rt2      0x00008014            0xE08BB000 ADD    r11,r11,r0
5     0.4        Exec    __scatterload_rt2      0x00008018            0xE24A7001 SUB    r7,r10,#1
6     0.45       Exec    __scatterload_null     0x0000801C            0xE15A000B CMP    r10,r11
7     0.5        Exec    __scatterload_null     0x00008020            0x1A000000 BNE    0x8028 <__sc
8     0.55       Exec    __scatterload_null     0x00008028            0xE8BA000F LDM    r10!,{r0-r3}
9     0.6        Exec    __scatterload_null     0x0000802C            0xE24FE018 ADR    lr,{pc}-0x10
10    0.65       Exec    __scatterload_null     0x00008030            0xE3130001 TST    r3,#1
11    0.7        NoExec  __scatterload_null     0x00008034            0x1047F003 SUBNE  pc,r7,r3
12    0.75       Exec    __scatterload_null     0x00008038            0xE1A0F003 MOV    pc,r3
13    0.9        Exec    __scatterload_copy     0x00008044            0xE2522010 SUBS   r2,r2,#0x10
14    0.95       NoExec  __scatterload_copy     0x00008048            0x28B00078 LDMCS  r0!,{r3-r6}
15    1          NoExec  __scatterload_copy     0x0000804C            0x28A10078 STMCS  r1!,{r3-r6}
16    1.05       NoExec  __scatterload_copy     0x00008050            0x8AFFFFFB BHI    __scatterloa
17    1.1        Exec    __scatterload_copy     0x00008054            0xE1B02E82 LSLS   r2,r2,#29
```

Figure 10-69 Trace captured before a trigger

5. Select **Find Trigger** from the Analysis window **Find** menu. RealView Debugger searches the trace output, and selects the line of trace corresponding to the trigger point.

6. Scroll down the trace output to see the captured trace after the trigger point. Figure 10-70 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem   Time/usec  Type    Symbolic                 Address     Data/Hex  Opcode           Other
6793   360.45     NoExec  GetAverage\#169          0x00008128            0xBAFFFFF9 BLT    0x8114 <TRACI
6794   360.5      Exec    GetAverage\#172..#174    0x0000812C            0xE1A01000 MOV    r1,r0
6795   360.55     Exec    GetAverage\#172..#174    0x00008130            0xE1A03FC0 ASR    r3,r0,#31
6796   360.6      Exec    GetAverage\#174          0x00008134            0xE0803E23 ADD    r3,r0,r3,LSR
6797   360.65     Exec    GetAverage\#174          0x00008138            0xE1A03243 ASR    r3,r3,#4
6798   360.7      Exec    GetAverage\#174          0x0000813C            0xE59FC180 LDR    r12,0x82c4
6799   360.75     Exec    GetAverage\#174          0x00008140            0xE58C3000 STR    r3,[r12,#0]
6800   360.8      Exec    GetAverage\#175          0x00008144            0xE12FFF1E BX     lr
*6801  361        Exec    main\#85                 0x0000828C            0xEBFFFF85 BL     SendData <0x
6802   361.05     Exec    SendData                 0x000080A8            0xE59F2214 LDR    r2,0x82c4
6803   361.1      Exec    SendData\#176..#212      0x000080AC            0xE5922000 LDR    r2,[r2,#0]
6804   361.15     Exec    SendData\#212            0x000080B0            0xE202200F AND    r2,r2,#0xf
6805   361.2      Exec    SendData\#212            0x000080B4            0xE2821001 ADD    r1,r2,#1
6806   361.25     Exec    SendData\#213..#214      0x000080B8            0xE59F2208 LDR    r2,0x82c8
6807   361.3      Exec    SendData\#213..#214      0x000080BC            0xE59F3208 LDR    r3,0x82cc
6808   361.35     Exec    SendData\#214            0x000080C0            0xE5933000 LDR    r3,[r3,#0]
6809   361.4      Exec    SendData\#214            0x000080C4            0xE5832000 STR    r2,[r3,#0]
6810   361.45     Exec    SendData\#215            0x000080C8            0xE59F21F4 LDR    r2,0x82c4
6811   361.5      Exec    SendData\#215            0x000080CC            0xE5922000 LDR    r2,[r2,#0]
```

Figure 10-70 Captured trace before the trigger point

7. Click **Stop** on the Debug toolbar to stop execution.

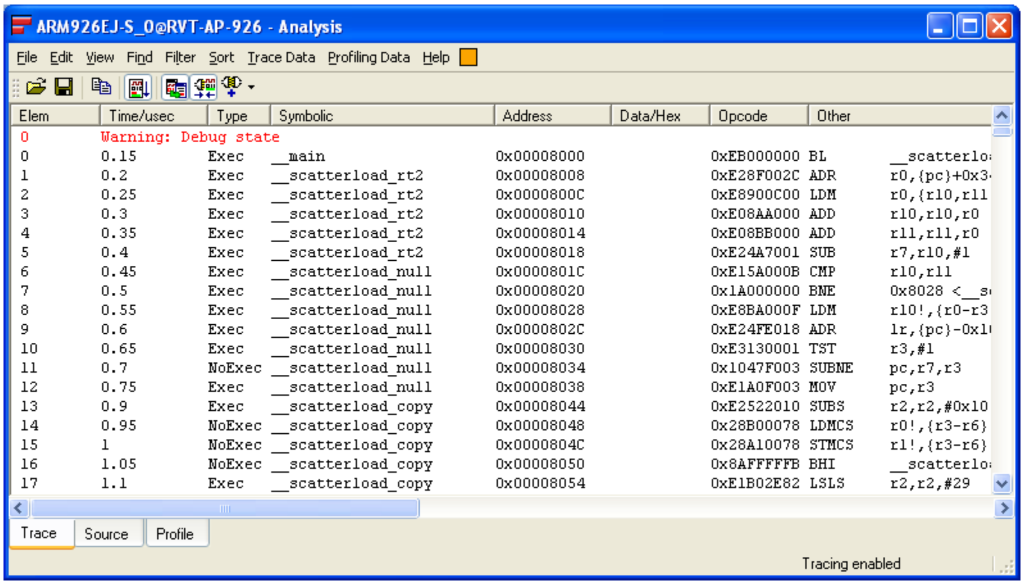
See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Trigger* on page 5-3
- *Setting a trigger point* on page 6-4
- *Setting a tracepoint that activates after a number of passes* on page 7-13
- *Clearing the trace buffer* on page 9-15.

10.14.3 Capturing trace around a trigger

To display the capture trace around a trigger:

1. Unload and reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The previous tracepoint is removed.
2. Set a trigger tracepoint at line 212:
   a. Click the **trace.c** tab to display the source code.
   b. Scroll down until line 212 is visible.
   c. Right-click in the margin at line 212 to display the context menu.
   d. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.
   e. Select **Set Trigger**.
   f. Click **OK**. The New Tracepoint dialog box closes and the tracepoint is set.
3. Select **Edit** → **Trigger Mode** → **Collect Trace Around Trigger** from the Analysis window menu.
4. Click **Run** on the Debug toolbar to start execution.
5. When the trigger is activated, the captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-71 on page 10-70 shows an example:

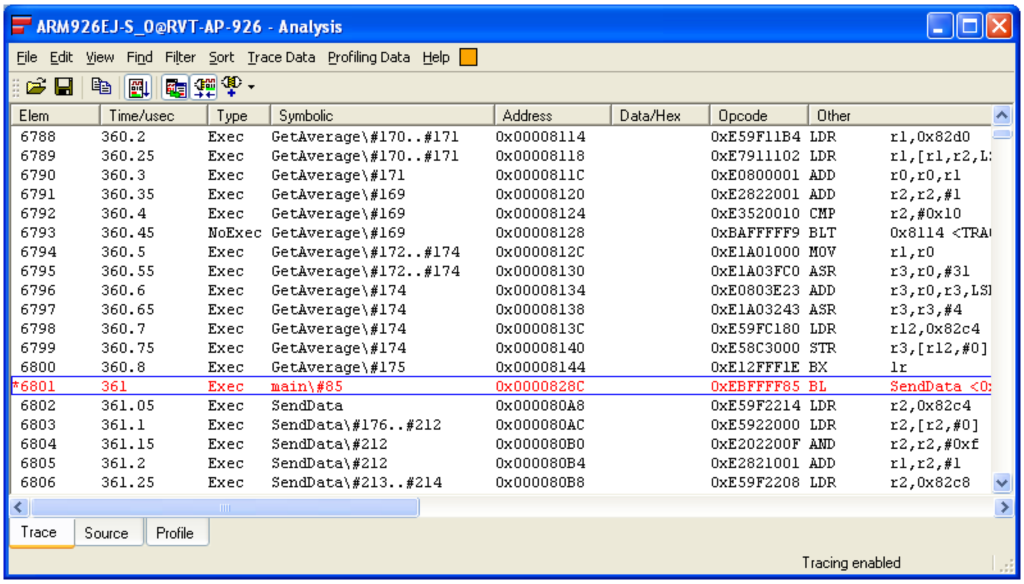

ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem    Time/usec  Type    Symbolic               Address     Data/Hex  Opcode            Other
0       Warning: Debug state
0       0.15       Exec    __main                 0x00008000            0xEB000000 BL     __scatterlo
1       0.2        Exec    __scatterload_rt2      0x00008008            0xE28F002C ADR    r0,{pc}+0x3
2       0.25       Exec    __scatterload_rt2      0x0000800C            0xE8900C00 LDM    r0,{r10,r11
3       0.3        Exec    __scatterload_rt2      0x00008010            0xE08AA000 ADD    r10,r10,r0
4       0.35       Exec    __scatterload_rt2      0x00008014            0xE08BB000 ADD    r11,r11,r0
5       0.4        Exec    __scatterload_rt2      0x00008018            0xE24A7001 SUB    r7,r10,#1
6       0.45       Exec    __scatterload_null     0x0000801C            0xE15A000B CMP    r10,r11
7       0.5        Exec    __scatterload_null     0x00008020            0x1A000000 BNE    0x8028 <__s
8       0.55       Exec    __scatterload_null     0x00008028            0xE8BA000F LDM    r10!,{r0-r3
9       0.6        Exec    __scatterload_null     0x0000802C            0xE24FE018 ADR    lr,{pc}-0x1
10      0.65       Exec    __scatterload_null     0x00008030            0xE3130001 TST    r3,#1
11      0.7        NoExec  __scatterload_null     0x00008034            0x1047F003 SUBNE  pc,r7,r3
12      0.75       Exec    __scatterload_null     0x00008038            0xE1A0F003 MOV    pc,r3
13      0.9        Exec    __scatterload_copy     0x00008044            0xE2522010 SUBS   r2,r2,#0x10
14      0.95       NoExec  __scatterload_copy     0x00008048            0x28B00078 LDMCS  r0!,{r3-r6}
15      1          NoExec  __scatterload_copy     0x0000804C            0x28A10078 STMCS  r1!,{r3-r6}
16      1.05       NoExec  __scatterload_copy     0x00008050            0x8AFFFFFB BHI    __scatterlo
17      1.1        Exec    __scatterload_copy     0x00008054            0xE1B02E82 LSLS   r2,r2,#29
```

Figure 10-71 Trace captured around a trigger

6. Select **Find Trigger** from the Analysis window **Find** menu. RealView Debugger searches the trace output, and selects the line of trace corresponding to the trigger point.

7. Scroll down the trace output to see the captured trace after the trigger point. Figure 10-72 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem    Time/usec  Type    Symbolic               Address     Data/Hex  Opcode            Other
6788    360.2      Exec    GetAverage\#170..#171  0x00008114            0xE59F11B4 LDR    r1,0x82d0
6789    360.25     Exec    GetAverage\#170..#171  0x00008118            0xE7911102 LDR    r1,[r1,r2,L
6790    360.3      Exec    GetAverage\#171        0x0000811C            0xE0800001 ADD    r0,r0,r1
6791    360.35     Exec    GetAverage\#169        0x00008120            0xE2822001 ADD    r2,r2,#1
6792    360.4      Exec    GetAverage\#169        0x00008124            0xE3520010 CMP    r2,#0x10
6793    360.45     NoExec  GetAverage\#169        0x00008128            0xBAFFFFF9 BLT    0x8114 <TRA
6794    360.5      Exec    GetAverage\#172..#174  0x0000812C            0xE1A01000 MOV    r1,r0
6795    360.55     Exec    GetAverage\#172..#174  0x00008130            0xE1A03FC0 ASR    r3,r0,#31
6796    360.6      Exec    GetAverage\#174        0x00008134            0xE0803E23 ADD    r3,r0,r3,LS
6797    360.65     Exec    GetAverage\#174        0x00008138            0xE1A03243 ASR    r3,r3,#4
6798    360.7      Exec    GetAverage\#174        0x0000813C            0xE59FC180 LDR    r12,0x82c4
6799    360.75     Exec    GetAverage\#174        0x00008140            0xE58C3000 STR    r3,[r12,#0]
6800    360.8      Exec    GetAverage\#175        0x00008144            0xE12FFF1E BX     lr
*6801   361        Exec    main\#85               0x0000828C            0xEBFFFF85 BL     SendData <0
6802    361.05     Exec    SendData               0x000080A8            0xE59F2214 LDR    r2,0x82c4
6803    361.1      Exec    SendData\#176..#212    0x000080AC            0xE5922000 LDR    r2,[r2,#0]
6804    361.15     Exec    SendData\#212          0x000080B0            0xE202200F AND    r2,r2,#0xf
6805    361.2      Exec    SendData\#212          0x000080B4            0xE2821001 ADD    r1,r2,#1
6806    361.25     Exec    SendData\#213..#214    0x000080B8            0xE59F2208 LDR    r2,0x82c8
```

Figure 10-72 Captured trace around the trigger point

8. Click **Stop** on the Debug toolbar to stop execution.

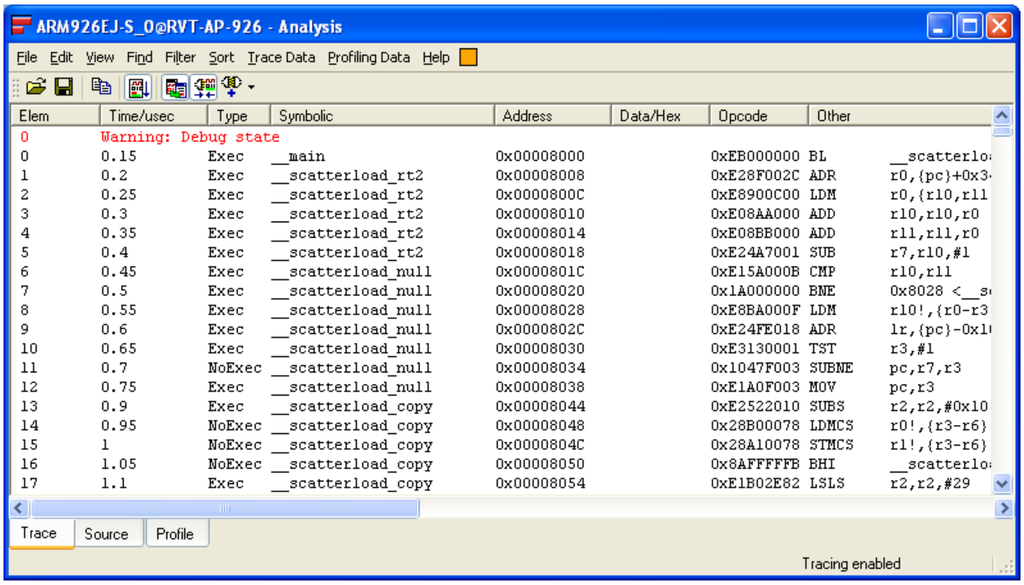
See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Trigger* on page 5-3
- *Setting a trigger point* on page 6-4
- *Setting a tracepoint that activates after a number of passes* on page 7-13
- *Clearing the trace buffer* on page 9-15.

10.14.4 Capturing trace after a trigger

To display the capture trace after a trigger:

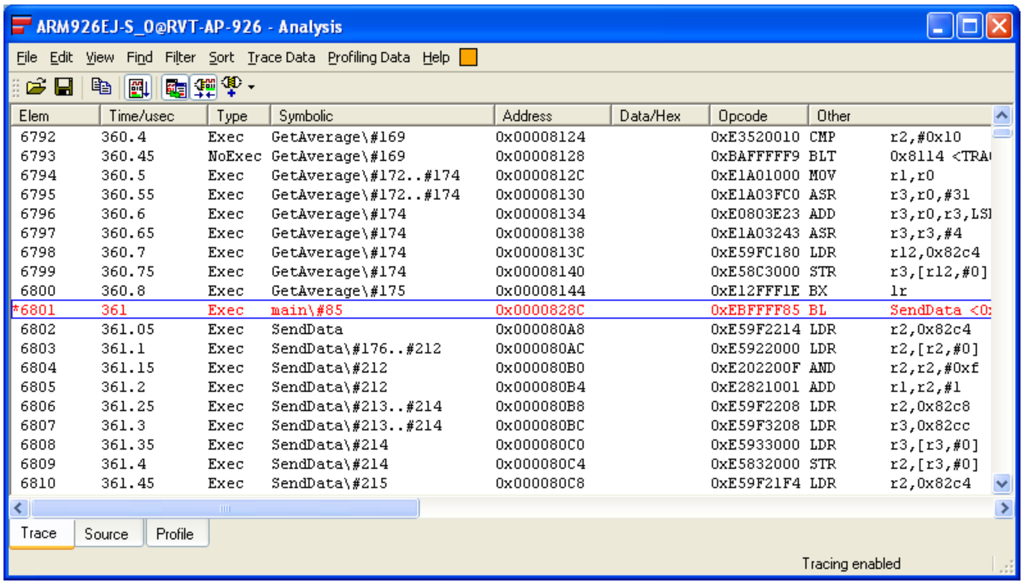
1. Unload and reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The previous tracepoint is removed.
2. Set a trigger tracepoint at line 212:
   a. Click the **trace.c** tab to display the source code.
   b. Scroll down until line 212 is visible.
   c. Right-click in the margin at line 212 to display the context menu.
   d. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.
   e. Select **Set Trigger**.
   f. Click **OK**. The New Tracepoint dialog box closes and the tracepoint is set.
3. Select **Edit** → **Trigger Mode** → **Collect Trace After Trigger** from the Analysis window menu.
4. Click **Run** on the Debug toolbar to start execution. When the trigger is activated, the captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-73 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Elem	Time/usec	Type	Symbolic	Address	Data/Hex	Opcode	Other
0	Warning: Debug state						
0	0.15	Exec	__main	0x00008000		0xEB000000 BL	__scatterlo
1	0.2	Exec	__scatterload_rt2	0x00008008		0xE28F002C ADR	r0,{pc}+0x3
2	0.25	Exec	__scatterload_rt2	0x0000800C		0xE8900C00 LDM	r0,{r10,r11
3	0.3	Exec	__scatterload_rt2	0x00008010		0xE08AA000 ADD	r10,r10,r0
4	0.35	Exec	__scatterload_rt2	0x00008014		0xE08BB000 ADD	r11,r11,r0
5	0.4	Exec	__scatterload_rt2	0x00008018		0xE24A7001 SUB	r7,r10,#1
6	0.45	Exec	__scatterload_null	0x0000801C		0xE15A000B CMP	r10,r11
7	0.5	Exec	__scatterload_null	0x00008020		0x1A000000 BNE	0x8028 <__s
8	0.55	Exec	__scatterload_null	0x00008028		0xE8BA000F LDM	r10!,{r0-r3
9	0.6	Exec	__scatterload_null	0x0000802C		0xE24FE018 ADR	lr,{pc}-0x1
10	0.65	Exec	__scatterload_null	0x00008030		0xE3130001 TST	r3,#1
11	0.7	NoExec	__scatterload_null	0x00008034		0x1047F003 SUBNE	pc,r7,r3
12	0.75	Exec	__scatterload_null	0x00008038		0xE1A0F003 MOV	pc,r3
13	0.9	Exec	__scatterload_copy	0x00008044		0xE2522010 SUBS	r2,r2,#0x10
14	0.95	NoExec	__scatterload_copy	0x00008048		0x28B00078 LDMCS	r0!,{r3-r6}
15	1	NoExec	__scatterload_copy	0x0000804C		0x28A10078 STMCS	r1!,{r3-r6}
16	1.05	NoExec	__scatterload_copy	0x00008050		0x8AFFFFFB BHI	__scatterlo
17	1.1	Exec	__scatterload_copy	0x00008054		0xE1B02E82 LSLS	r2,r2,#29

Figure 10-73 Trace captured after a trigger

5. Select **Find Trigger** from the Analysis window **Find** menu. RealView Debugger searches the trace output, and selects the line of trace corresponding to the trigger point.
6. Scroll down the trace output to see the captured trace after the trigger point. Figure 10-74 on page 10-72 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

Elem	Time/usec	Type	Symbolic	Address	Data/Hex	Opcode	Other
6792	360.4	Exec	GetAverage\#169	0x00008124		0xE3520010 CMP	r2,#0x10
6793	360.45	NoExec	GetAverage\#169	0x00008128		0xBAFFFFF9 BLT	0x8114 <TRA
6794	360.5	Exec	GetAverage\#172..#174	0x0000812C		0xE1A01000 MOV	r1,r0
6795	360.55	Exec	GetAverage\#172..#174	0x00008130		0xE1A03FC0 ASR	r3,r0,#31
6796	360.6	Exec	GetAverage\#174	0x00008134		0xE0803E23 ADD	r3,r0,r3,LSl
6797	360.65	Exec	GetAverage\#174	0x00008138		0xE1A03243 ASR	r3,r3,#4
6798	360.7	Exec	GetAverage\#174	0x0000813C		0xE59FC180 LDR	r12,0x82c4
6799	360.75	Exec	GetAverage\#174	0x00008140		0xE58C3000 STR	r3,[r12,#0]
6800	360.8	Exec	GetAverage\#175	0x00008144		0xE12FFF1E BX	lr
*6801	361	Exec	main\#85	0x0000828C		0xEBFFFF85 BL	SendData <0:
6802	361.05	Exec	SendData	0x000080A8		0xE59F2214 LDR	r2,0x82c4
6803	361.1	Exec	SendData\#176..#212	0x000080AC		0xE5922000 LDR	r2,[r2,#0]
6804	361.15	Exec	SendData\#212	0x000080B0		0xE202200F AND	r2,r2,#0xf
6805	361.2	Exec	SendData\#212	0x000080B4		0xE2821001 ADD	r1,r2,#1
6806	361.25	Exec	SendData\#213..#214	0x000080B8		0xE59F2208 LDR	r2,0x82c8
6807	361.3	Exec	SendData\#213..#214	0x000080BC		0xE59F3208 LDR	r3,0x82cc
6808	361.35	Exec	SendData\#214	0x000080C0		0xE5933000 LDR	r3,[r3,#0]
6809	361.4	Exec	SendData\#214	0x000080C4		0xE5832000 STR	r2,[r3,#0]
6810	361.45	Exec	SendData\#215	0x000080C8		0xE59F21F4 LDR	r2,0x82c4

Figure 10-74 Captured trace after the trigger point

7. Click **Stop** on the Debug toolbar to stop execution.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Trigger* on page 5-3
- *Setting a trigger point* on page 6-4
- *Setting a tracepoint that activates after a number of passes* on page 7-13
- *Clearing the trace buffer* on page 9-15.

10.14.5 Examining the captured trace

Examine the captured trace in the **Trace** tab:

1. When collecting trace before a trigger point, a small amount of trace is often included after the trigger point. The captured trace is displayed when the trigger is activated.

2. When collecting trace after a trigger point, a small amount of trace is often included before the trigger point.

   Because the trace is not captured until the trigger is activated, there is a delay in displaying the captured trace. This delay depends on your target processor, and the size of the trace buffer.

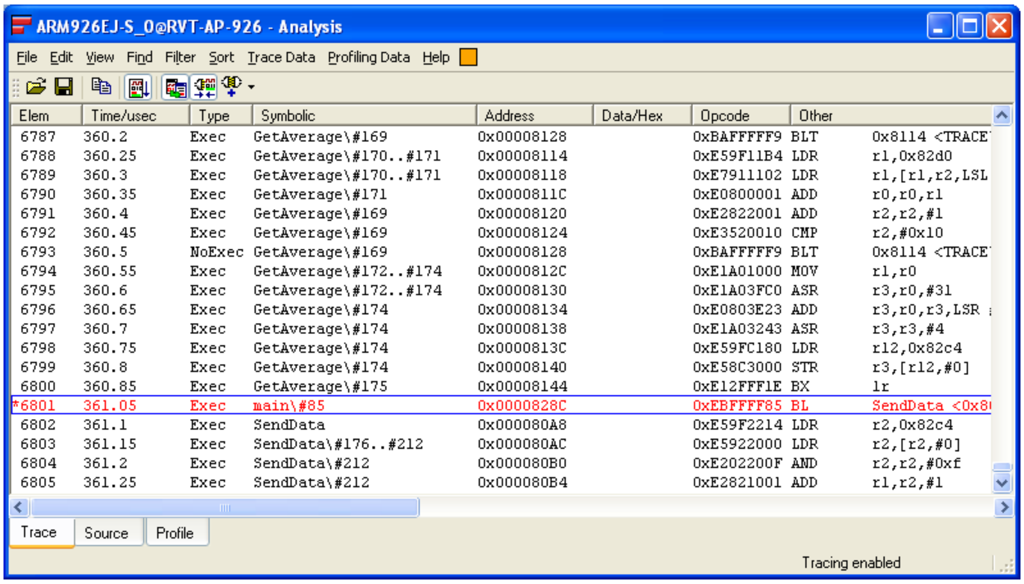
3. When collecting trace around a trigger point, the amount of trace captured before and after the trigger point depends on the position of the trigger point.

   You can alter the amount of trace information before and after the trigger by using a Trace Start Point to enable tracing before the trigger, and a Trace End Point to disable tracing after the trigger.

   Because the trace following the trigger is not captured until the trigger is activated, there is a delay in displaying the captured trace. This delay depends on your target processor, and the size of the trace buffer.

4. You can stop the processor when the trigger is activated. To do this, select **Edit** → **Trigger Mode** → **Stop Processor on Trigger** when you set the trigger mode. The trace information that is collected when tracing around or after the trigger depends on your

TPA. For example, with RealView Trace, trace information is captured before the trigger, but none after the trigger. However, some trace is captured after the trigger because of the short delay in stopping the processor. Figure 10-75 shows an example:



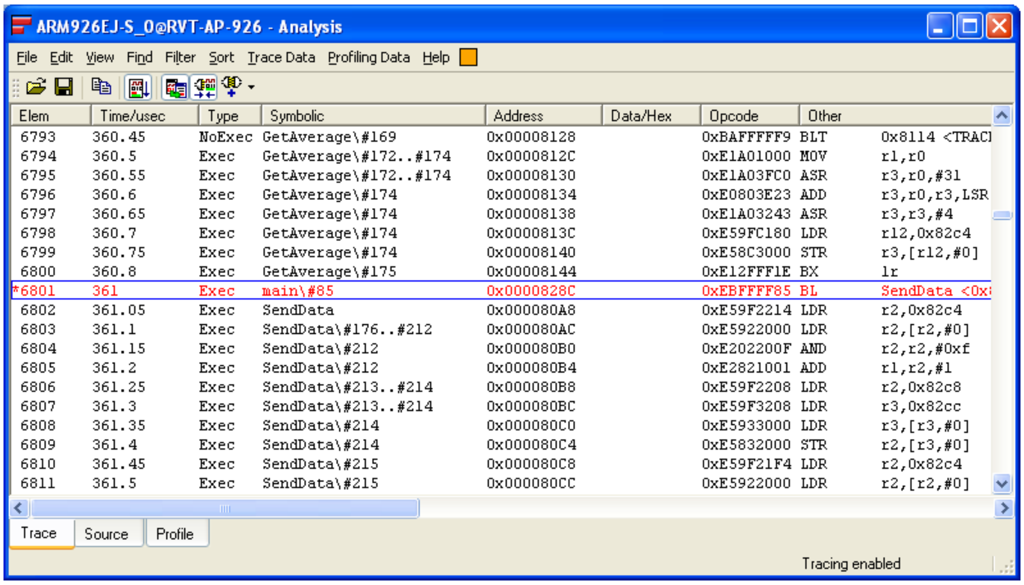
ARM926EJ-S_0@RVT-AP-926 - Analysis

Elem	Time/usec	Type	Symbolic	Address	Data/Hex	Opcode	Other
6787	360.2	Exec	GetAverage\#169	0x00008128		0xBAFFFFF9 BLT	0x8114 <TRACE
6788	360.25	Exec	GetAverage\#170..#171	0x00008114		0xE59F11B4 LDR	r1,0x82d0
6789	360.3	Exec	GetAverage\#170..#171	0x00008118		0xE7911102 LDR	r1,[r1,r2,LSL
6790	360.35	Exec	GetAverage\#171	0x0000811C		0xE0800001 ADD	r0,r0,r1
6791	360.4	Exec	GetAverage\#169	0x00008120		0xE2822001 ADD	r2,r2,#1
6792	360.45	Exec	GetAverage\#169	0x00008124		0xE3520010 CMP	r2,#0x10
6793	360.5	NoExec	GetAverage\#169	0x00008128		0xBAFFFFF9 BLT	0x8114 <TRACE
6794	360.55	Exec	GetAverage\#172..#174	0x0000812C		0xE1A01000 MOV	r1,r0
6795	360.6	Exec	GetAverage\#172..#174	0x00008130		0xE1A03FC0 ASR	r3,r0,#31
6796	360.65	Exec	GetAverage\#174	0x00008134		0xE0803E23 ADD	r3,r0,r3,LSR
6797	360.7	Exec	GetAverage\#174	0x00008138		0xE1A03243 ASR	r3,r3,#4
6798	360.75	Exec	GetAverage\#174	0x0000813C		0xE59FC180 LDR	r12,0x82c4
6799	360.8	Exec	GetAverage\#174	0x00008140		0xE58C3000 STR	r3,[r12,#0]
6800	360.85	Exec	GetAverage\#175	0x00008144		0xE12FFF1E BX	lr
*6801	361.05	Exec	main\#85	0x0000828C		0xEBFFFF85 BL	SendData <0x8
6802	361.1	Exec	SendData	0x000080A8		0xE59F2214 LDR	r2,0x82c4
6803	361.15	Exec	SendData\#176..#212	0x000080AC		0xE5922000 LDR	r2,[r2,#0]
6804	361.2	Exec	SendData\#212	0x000080B0		0xE202200F AND	r2,r2,#0xf
6805	361.25	Exec	SendData\#212	0x000080B4		0xE2821001 ADD	r1,r2,#1

File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Figure 10-75 Captured trace from RealView Trace when stopped at a trigger

5. Change the trigger to activate after 10 passes:
   a. Select **Edit** → **Trigger Mode** → **Stop Processor on Trigger** to disable this setting.
   b. Select **Break/Tracepoints** from the **View** menu of the Code window to display the Break/Tracepoints view.
   c. Right-click on the tracepoint entry to display the context menu.
   d. Select **Edit...** from the context menu to display the Create Tracepoint dialog box.
   e. Enter **10** in the **Pass times** field.
   f. Click **OK** to close the Create Tracepoint dialog box.
   g. Select **Reload Image to Target** from the **Target** menu of the Code window to reload the image.
   h. Click **Run** on the Debug toolbar to start execution. The trigger activates after 10 passes, and the captured trace is displayed in the Analysis window. Figure 10-76 on page 10-74 shows an example:



ARM926EJ-S_0@RVT-AP-926 - Analysis

Elem	Time/usec	Type	Symbolic	Address	Data/Hex	Opcode	Other
6793	360.45	NoExec	GetAverage\#169	0x00008128		0xBAFFFFF9 BLT	0x8114 <TRACI
6794	360.5	Exec	GetAverage\#172..#174	0x0000812C		0xE1A01000 MOV	r1,r0
6795	360.55	Exec	GetAverage\#172..#174	0x00008130		0xE1A03FC0 ASR	r3,r0,#31
6796	360.6	Exec	GetAverage\#174	0x00008134		0xE0803E23 ADD	r3,r0,r3,LSR
6797	360.65	Exec	GetAverage\#174	0x00008138		0xE1A03243 ASR	r3,r3,#4
6798	360.7	Exec	GetAverage\#174	0x0000813C		0xE59FC180 LDR	r12,0x82c4
6799	360.75	Exec	GetAverage\#174	0x00008140		0xE58C3000 STR	r3,[r12,#0]
6800	360.8	Exec	GetAverage\#175	0x00008144		0xE12FFF1E BX	lr
*6801	361	Exec	main\#85	0x0000828C		0xEBFFFF85 BL	SendData <0x
6802	361.05	Exec	SendData	0x000080A8		0xE59F2214 LDR	r2,0x82c4
6803	361.1	Exec	SendData\#176..#212	0x000080AC		0xE5922000 LDR	r2,[r2,#0]
6804	361.15	Exec	SendData\#212	0x000080B0		0xE202200F AND	r2,r2,#0xf
6805	361.2	Exec	SendData\#212	0x000080B4		0xE2821001 ADD	r1,r2,#1
6806	361.25	Exec	SendData\#213..#214	0x000080B8		0xE59F2208 LDR	r2,0x82c8
6807	361.3	Exec	SendData\#213..#214	0x000080BC		0xE59F3208 LDR	r3,0x82cc
6808	361.35	Exec	SendData\#214	0x000080C0		0xE5933000 LDR	r3,[r3,#0]
6809	361.4	Exec	SendData\#214	0x000080C4		0xE5832000 STR	r2,[r3,#0]
6810	361.45	Exec	SendData\#215	0x000080C8		0xE59F21F4 LDR	r2,0x82c4
6811	361.5	Exec	SendData\#215	0x000080CC		0xE5922000 LDR	r2,[r2,#0]

File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Figure 10-76 Captured trace after 10 passes

6. Click **Stop** on the Debug toolbar to stop execution.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Trigger* on page 5-3
- *Setting a trigger point* on page 6-4
- *Setting a tracepoint that activates after a number of passes* on page 7-13
- *Clearing the trace buffer* on page 9-15.

10.15 Using triggers to collect trace around a specific data access

You can collect trace when a specified data access occurs. You can display the trace captured:

- before the trigger point, which is the default action
- after the trigger point
- around the trigger point (distributed evenly around the trigger point by default).

Note

You can also choose to stop execution when the trigger is hit.

See also:

- *Tracing features used*
- *Capturing trace before a trigger*
- *Capturing trace around a trigger* on page 10-77
- *Capturing trace after a trigger* on page 10-80
- *Examining the captured trace* on page 10-82.

10.15.1 Tracing features used

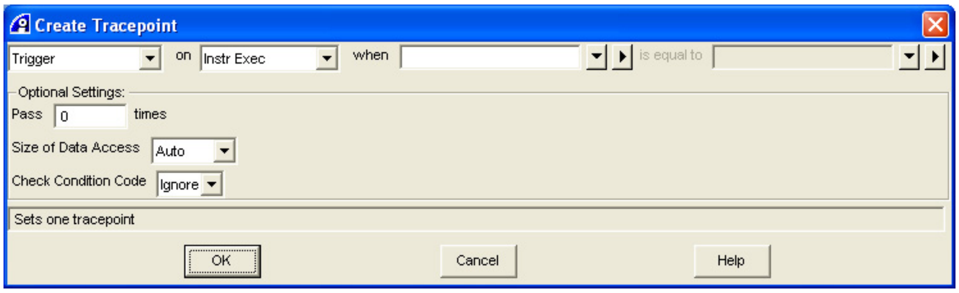
The following tracing features are used in this procedure:

- clearing the trace buffer
- setting a trigger tracepoint
- finding the sample corresponding to the trigger in the captured trace
- setting a conditional trigger tracepoint that activates after a specific number of passes
- specifying the trigger mode:
  - Collect Trace Before Trigger
  - Collect Trace Around Trigger
  - Collect Trace After Trigger.

10.15.2 Capturing trace before a trigger

To display the capture trace before a trigger:

1. Unload and reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The previous tracepoint is removed.
2. Set a trigger tracepoint on data writes to `output_port` that activates after 200 passes:
   a. Select **Debug** → **Tracepoints** → **Create Tracepoint...** from the Code window main menu to display the Create Tracepoint dialog box. Figure 10-77 on page 10-76 shows an example:



Create Tracepoint

Trigger on Instr Exec when is equal to

Optional Settings:

Pass 0 times

Size of Data Access Auto

Check Condition Code Ignore

Sets one tracepoint

OK Cancel Help

Figure 10-77 Create Tracepoint dialog box

b. Select **Trigger** from the tracepoint type drop-down list.

c. Select **Data Write** from the tracepoint comparison type drop-down list (that is, the `on` field).

d. Enter the address to be traced (that is, the `when` field). In this case, the address to be traced corresponds to the symbol `output_port`. Therefore, enter **@trace\\output_port**.

Note

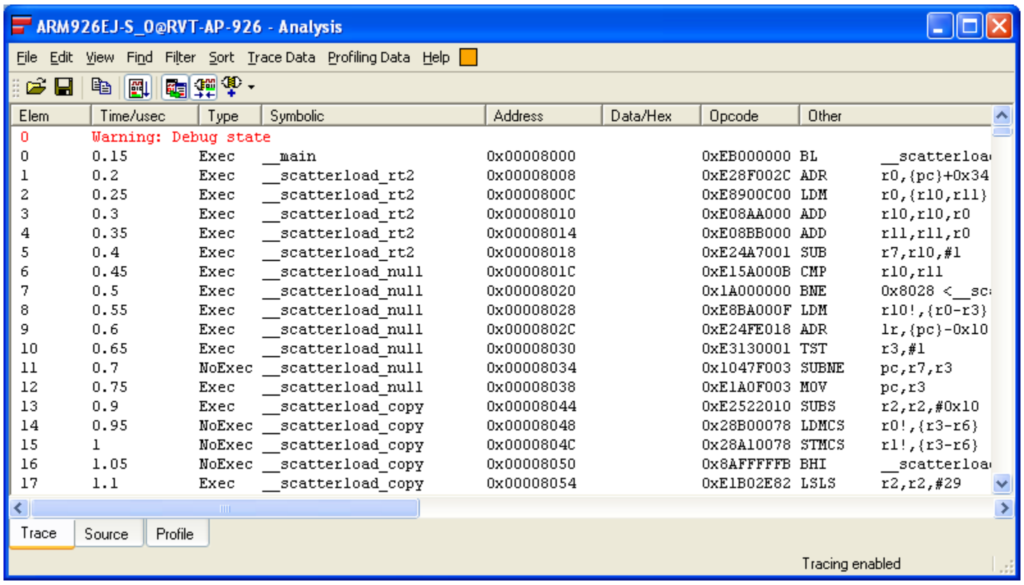
Because this is a pointer, you do not have to prefix the variable name with `&`.

e. In the `Optional Settings` group, enter **200** in the `Pass times` text box.

f. Click **OK**. The Create Tracepoint dialog box closes, and the tracepoint is set. RealView Debugger generates a `TRACEDATAWRITE` CLI command, which is also displayed in the **Cmd** tab of the Output view:

```
trcdwrite,hw_out:"Tracepoint Type=Trigger",hw_pass:200 @trace\\output_port
```

3. Click **Run** on the Debug toolbar to start execution. The trigger is activated after 200 writes to `output_port`, and the captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-78 shows an example:



ARM926EJ-S_0@RVT-AP-926 - Analysis

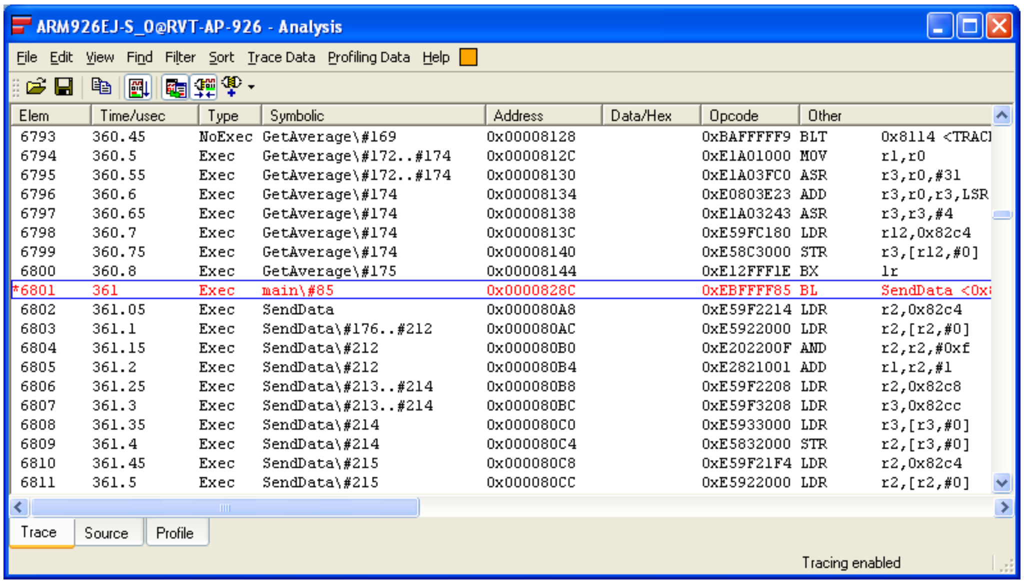
```
Elem  Time/usec  Type    Symbolic             Address     Data/Hex  Opcode           Other
0     Warning: Debug state
0     0.15       Exec    __main               0x00008000            0xEB000000 BL    __scatterloa
1     0.2        Exec    __scatterload_rt2    0x00008008            0xE28F002C ADR   r0,{pc}+0x34
2     0.25       Exec    __scatterload_rt2    0x0000800C            0xE8900C00 LDM   r0,{r10,r11}
3     0.3        Exec    __scatterload_rt2    0x00008010            0xE08AA000 ADD   r10,r10,r0
4     0.35       Exec    __scatterload_rt2    0x00008014            0xE08BB000 ADD   r11,r11,r0
5     0.4        Exec    __scatterload_rt2    0x00008018            0xE24A7001 SUB   r7,r10,#1
6     0.45       Exec    __scatterload_null   0x0000801C            0xE15A000B CMP   r10,r11
7     0.5        Exec    __scatterload_null   0x00008020            0x1A000000 BNE   0x8028 <__sc
8     0.55       Exec    __scatterload_null   0x00008028            0xE8BA000F LDM   r10!,{r0-r3}
9     0.6        Exec    __scatterload_null   0x0000802C            0xE24FE018 ADR   lr,{pc}-0x10
10    0.65       Exec    __scatterload_null   0x00008030            0xE3130001 TST   r3,#1
11    0.7        NoExec  __scatterload_null   0x00008034            0x1047F003 SUBNE pc,r7,r3
12    0.75       Exec    __scatterload_null   0x00008038            0xE1A0F003 MOV   pc,r3
13    0.9        Exec    __scatterload_copy   0x00008044            0xE2522010 SUBS  r2,r2,#0x10
14    0.95       NoExec  __scatterload_copy   0x00008048            0x28B00078 LDMCS r0!,{r3-r6}
15    1          NoExec  __scatterload_copy   0x0000804C            0x28A10078 STMCS r1!,{r3-r6}
16    1.05       NoExec  __scatterload_copy   0x00008050            0x8AFFFFFB BHI   __scatterloa
17    1.1        Exec    __scatterload_copy   0x00008054            0xE1B02E82 LSLS  r2,r2,#29
```

File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

Figure 10-78 Trace captured before a trigger

4. Select **Find Trigger** from the Analysis window **Find** menu. RealView Debugger searches the trace output and the line of trace corresponding to the trigger point is selected.

5. Scroll down the trace output to see the captured trace after the trigger point. Figure 10-79 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Elem	Time/usec	Type	Symbolic	Address	Data/Hex	Opcode	Other
6793	360.45	NoExec	GetAverage\#169	0x00008128		0xBAFFFFF9 BLT	0x8114 <TRACI
6794	360.5	Exec	GetAverage\#172..#174	0x0000812C		0xE1A01000 MOV	r1,r0
6795	360.55	Exec	GetAverage\#172..#174	0x00008130		0xE1A03FC0 ASR	r3,r0,#31
6796	360.6	Exec	GetAverage\#174	0x00008134		0xE0803E23 ADD	r3,r0,r3,LSR
6797	360.65	Exec	GetAverage\#174	0x00008138		0xE1A03243 ASR	r3,r3,#4
6798	360.7	Exec	GetAverage\#174	0x0000813C		0xE59FC180 LDR	r12,0x82c4
6799	360.75	Exec	GetAverage\#174	0x00008140		0xE58C3000 STR	r3,[r12,#0]
6800	360.8	Exec	GetAverage\#175	0x00008144		0xE12FFF1E BX	lr
*6801	361	Exec	main\#85	0x0000828C		0xEBFFFF85 BL	SendData <0x
6802	361.05	Exec	SendData	0x000080A8		0xE59F2214 LDR	r2,0x82c4
6803	361.1	Exec	SendData\#176..#212	0x000080AC		0xE5922000 LDR	r2,[r2,#0]
6804	361.15	Exec	SendData\#212	0x000080B0		0xE202200F AND	r2,r2,#0xf
6805	361.2	Exec	SendData\#212	0x000080B4		0xE2821001 ADD	r1,r2,#1
6806	361.25	Exec	SendData\#213..#214	0x000080B8		0xE59F2208 LDR	r2,0x82c8
6807	361.3	Exec	SendData\#213..#214	0x000080BC		0xE59F3208 LDR	r3,0x82cc
6808	361.35	Exec	SendData\#214	0x000080C0		0xE5933000 LDR	r3,[r3,#0]
6809	361.4	Exec	SendData\#214	0x000080C4		0xE5832000 STR	r2,[r3,#0]
6810	361.45	Exec	SendData\#215	0x000080C8		0xE59F21F4 LDR	r2,0x82c4
6811	361.5	Exec	SendData\#215	0x000080CC		0xE5922000 LDR	r2,[r2,#0]

Figure 10-79 Captured trace before the trigger

6. Click **Stop** on the Debug toolbar to stop execution.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Trigger* on page 5-3
- *Setting a trigger point* on page 6-4
- Chapter 7 *Setting Conditional Tracepoints*
- the following in the *RealView Debugger Command Line Reference Guide*:
  — Chapter 2 *RealView Debugger Commands* for details of the `TRACEDATAWRITE` command.

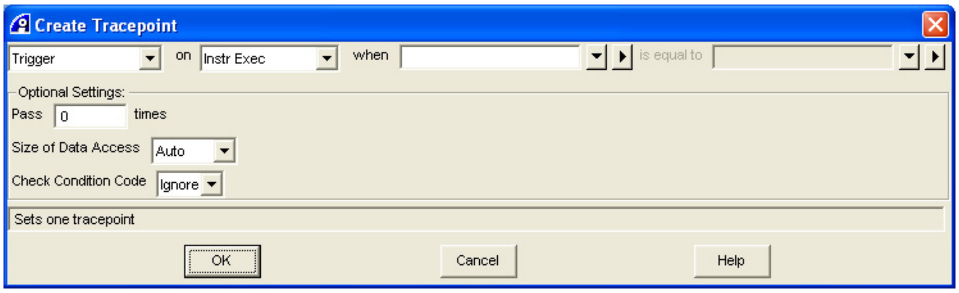
10.15.3 Capturing trace around a trigger

To display the capture trace around a trigger:

1. Unload and reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The previous tracepoint is removed.

2. Set a trigger tracepoint on data writes to `output_port` that activates after 200 passes:

   a. Select **Debug → Tracepoints → Create Tracepoint...** from the Code window main menu to display the Create Tracepoint dialog box. Figure 10-80 shows an example:



Create Tracepoint

Trigger on Instr Exec when is equal to

Optional Settings:

Pass 0 times

Size of Data Access Auto

Check Condition Code Ignore

Sets one tracepoint

Figure 10-80 Create Tracepoint dialog box

   b. Select **Trigger** from the tracepoint type drop-down list.

   c. Select **Data Write** from the tracepoint comparison type drop-down list (that is, the `on` field).

   d. Enter the address to be traced (that is, the `when` field). In this case, the address to be traced corresponds to the symbol `output_port`. Therefore, enter **@trace\\output_port**.

   ——— **Note** ———

   Because this is a pointer, you do not have to prefix the variable name with `&`.

   ————————————

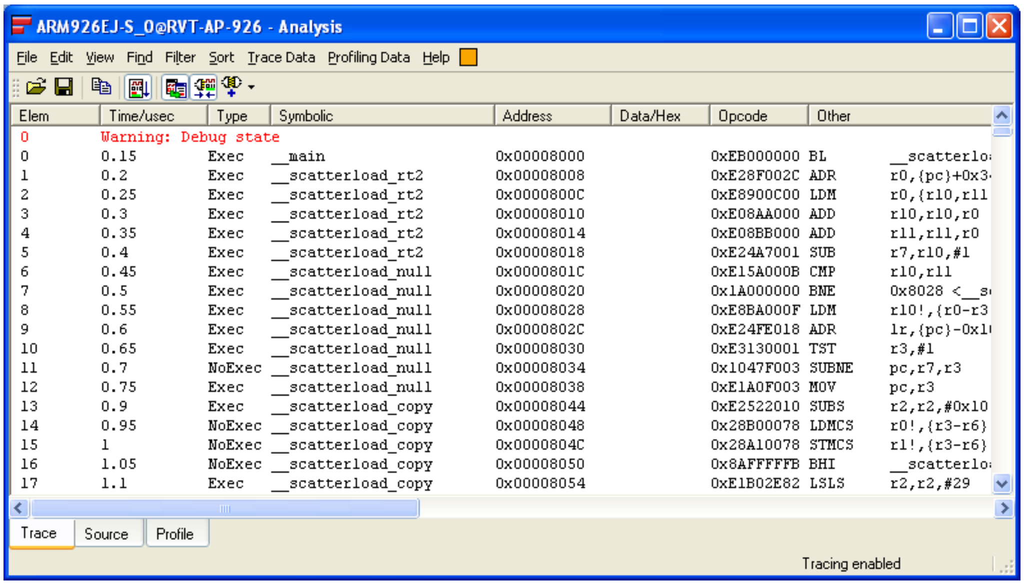
   e. In the `Optional Settings` group, enter **200** in the `Pass times` text box.

   f. Click **OK**. The Create Tracepoint dialog box closes, and the tracepoint is set. RealView Debugger generates a `TRACEDATAWRITE` CLI command, which is also displayed in the **Cmd** tab of the Output view:

```
trcdwrite,hw_out:"Tracepoint Type=Trigger",hw_pass:200 @trace\\output_port
```

3. Select **Edit → Trigger Mode → Collect Trace Around Trigger** from the Analysis window menu.

4. Click **Run** on the Debug toolbar to start execution. The trigger is activated after 200 writes to `output_port`, and the captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-81 on page 10-79 shows an example:

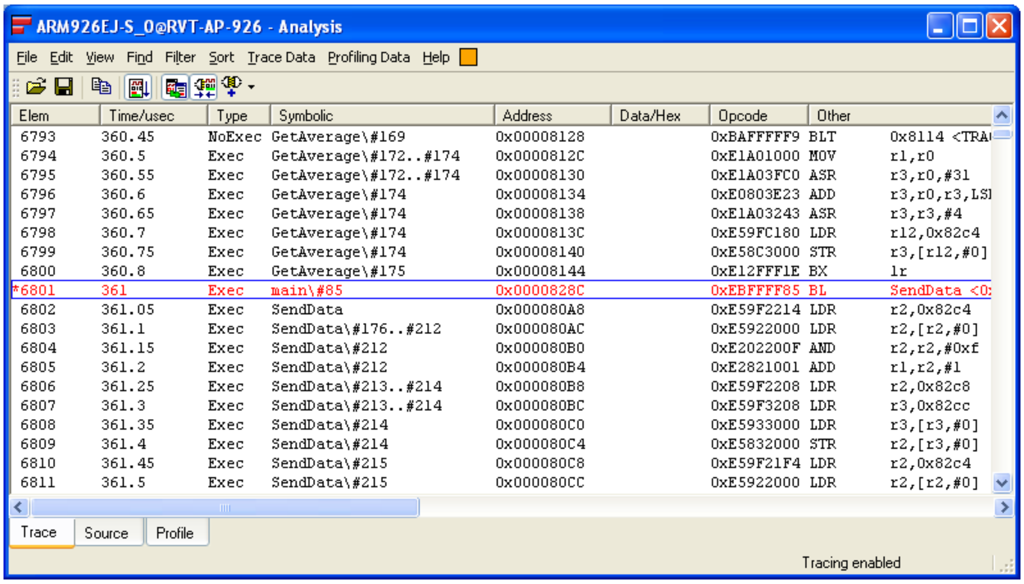

ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem  Time/usec  Type    Symbolic            Address     Data/Hex  Opcode          Other
0     Warning: Debug state
0     0.15       Exec    __main              0x00008000            0xEB000000 BL     __scatterlo
1     0.2        Exec    __scatterload_rt2   0x00008008            0xE28F002C ADR    r0,{pc}+0x3
2     0.25       Exec    __scatterload_rt2   0x0000800C            0xE8900C00 LDM    r0,{r10,r11
3     0.3        Exec    __scatterload_rt2   0x00008010            0xE08AA000 ADD    r10,r10,r0
4     0.35       Exec    __scatterload_rt2   0x00008014            0xE08BB000 ADD    r11,r11,r0
5     0.4        Exec    __scatterload_rt2   0x00008018            0xE24A7001 SUB    r7,r10,#1
6     0.45       Exec    __scatterload_null  0x0000801C            0xE15A000B CMP    r10,r11
7     0.5        Exec    __scatterload_null  0x00008020            0x1A000000 BNE    0x8028 <__s
8     0.55       Exec    __scatterload_null  0x00008028            0xE8BA000F LDM    r10!,{r0-r3
9     0.6        Exec    __scatterload_null  0x0000802C            0xE24FE018 ADR    lr,{pc}-0x1
10    0.65       Exec    __scatterload_null  0x00008030            0xE3130001 TST    r3,#1
11    0.7        NoExec  __scatterload_null  0x00008034            0x1047F003 SUBNE  pc,r7,r3
12    0.75       Exec    __scatterload_null  0x00008038            0xE1A0F003 MOV    pc,r3
13    0.9        Exec    __scatterload_copy  0x00008044            0xE2522010 SUBS   r2,r2,#0x10
14    0.95       NoExec  __scatterload_copy  0x00008048            0x28B00078 LDMCS  r0!,{r3-r6}
15    1          NoExec  __scatterload_copy  0x0000804C            0x28A10078 STMCS  r1!,{r3-r6}
16    1.05       NoExec  __scatterload_copy  0x00008050            0x8AFFFFFB BHI    __scatterlo
17    1.1        Exec    __scatterload_copy  0x00008054            0xE1B02E82 LSLS   r2,r2,#29
```

Figure 10-81 Trace captured around a trigger

5. Select **Find Trigger** from the Analysis window **Find** menu. RealView Debugger searches the trace output and the line of trace corresponding to the trigger point is selected.

6. Scroll down the trace output to see the captured trace after the trigger point. Figure 10-82 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem   Time/usec  Type    Symbolic              Address     Data/Hex  Opcode          Other
6793   360.45     NoExec  GetAverage\#169       0x00008128            0xBAFFFFF9 BLT    0x8114 <TRA
6794   360.5      Exec    GetAverage\#172..#174 0x0000812C            0xE1A01000 MOV    r1,r0
6795   360.55     Exec    GetAverage\#172..#174 0x00008130            0xE1A03FC0 ASR    r3,r0,#31
6796   360.6      Exec    GetAverage\#174       0x00008134            0xE0803E23 ADD    r3,r0,r3,LS
6797   360.65     Exec    GetAverage\#174       0x00008138            0xE1A03243 ASR    r3,r3,#4
6798   360.7      Exec    GetAverage\#174       0x0000813C            0xE59FC180 LDR    r12,0x82c4
6799   360.75     Exec    GetAverage\#174       0x00008140            0xE58C3000 STR    r3,[r12,#0]
6800   360.8      Exec    GetAverage\#175       0x00008144            0xE12FFF1E BX     lr
*6801  361        Exec    main\#85              0x0000828C            0xEBFFFF85 BL     SendData <0
6802   361.05     Exec    SendData              0x000080A8            0xE59F2214 LDR    r2,0x82c4
6803   361.1      Exec    SendData\#176..#212   0x000080AC            0xE5922000 LDR    r2,[r2,#0]
6804   361.15     Exec    SendData\#212         0x000080B0            0xE202200F AND    r2,r2,#0xf
6805   361.2      Exec    SendData\#212         0x000080B4            0xE2821001 ADD    r1,r2,#1
6806   361.25     Exec    SendData\#213..#214   0x000080B8            0xE59F2208 LDR    r2,0x82c8
6807   361.3      Exec    SendData\#213..#214   0x000080BC            0xE59F3208 LDR    r3,0x82cc
6808   361.35     Exec    SendData\#214         0x000080C0            0xE5933000 LDR    r3,[r3,#0]
6809   361.4      Exec    SendData\#214         0x000080C4            0xE5832000 STR    r2,[r3,#0]
6810   361.45     Exec    SendData\#215         0x000080C8            0xE59F21F4 LDR    r2,0x82c4
6811   361.5      Exec    SendData\#215         0x000080CC            0xE5922000 LDR    r2,[r2,#0]
```

Figure 10-82 Captured trace around the trigger

7. Click **Stop** on the Debug toolbar to stop execution.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Trigger* on page 5-3
- *Setting a trigger point* on page 6-4

- Chapter 7 *Setting Conditional Tracepoints*
- the following in the *RealView Debugger Command Line Reference Guide*:
  - — Chapter 2 *RealView Debugger Commands* for details of the `TRACEDATAWRITE` command.

10.15.4 Capturing trace after a trigger

To display the capture trace after a trigger:

1. Unload and reload the image:
   a. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   b. Select **Unload** from the context menu. The image is unloaded.
   c. Right-click on the `Load Image+Symbols` entry in the Process Control view to display the context menu.
   d. Select **Load** from the context menu. The image is reloaded and the PC is reset to the image entry point (`0x8000`). The previous tracepoint is removed.
2. Set a trigger tracepoint on data writes to `output_port` that activates after 200 passes:
   a. Select **Debug** → **Tracepoints** → **Create Tracepoint...** from the Code window main menu to display the Create Tracepoint dialog box. Figure 10-83 shows an example:

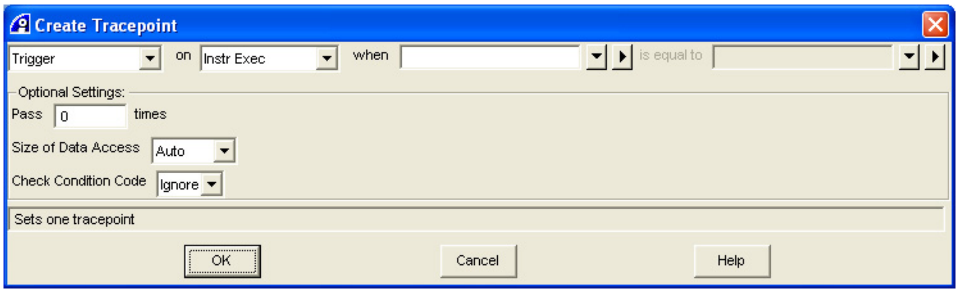


Figure 10-83 Create Tracepoint dialog box

   b. Select **Trigger** from the tracepoint type drop-down list.
   c. Select **Data Write** from the tracepoint comparison type drop-down list (that is, the `on` field).
   d. Enter the address to be traced (that is, the `when` field). In this case, the address to be traced corresponds to the symbol `output_port`. Therefore, enter **@trace\\output_port**.

      ——— **Note** ———

      Because this is a pointer, you do not have to prefix the variable name with `&`.

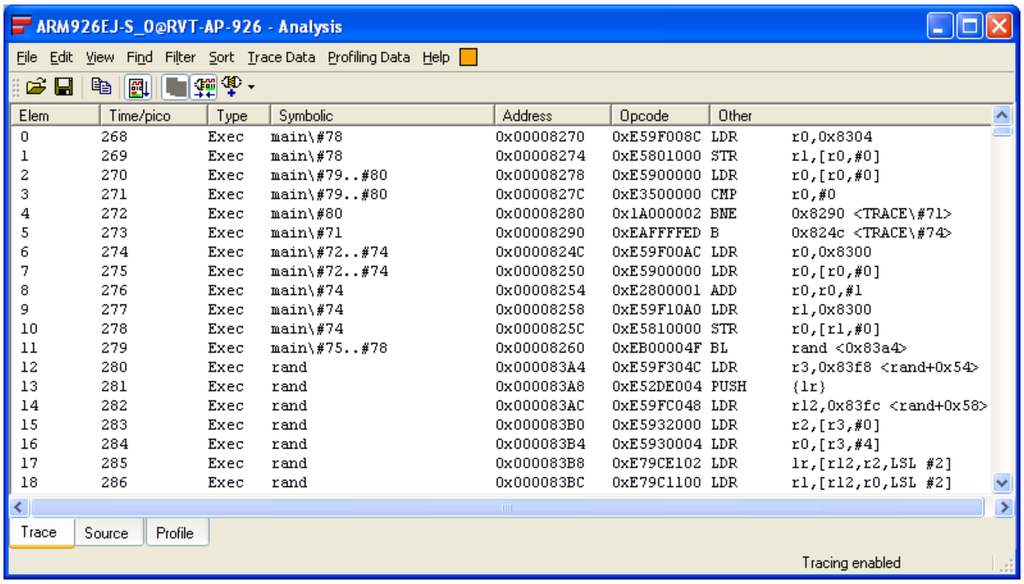
      ————————

   e. In the `Optional Settings` group, enter **200** in the `Pass times` text box.
   f. Click **OK**. The Create Tracepoint dialog box closes, and the tracepoint is set. RealView Debugger generates a `TRACEDATAWRITE` CLI command, which is also displayed in the **Cmd** tab of the Output view:

```
trcdwrite,hw_out:"Tracepoint Type=Trigger",hw_pass:200 @trace\\output_port
```

3. Select **Edit** → **Trigger Mode** → **Collect Trace After Trigger** from the Analysis window menu.

4. Click **Run** on the Debug toolbar to start execution. The trigger is activated after 200 writes to `output_port`, and the captured trace is displayed in the **Trace** tab of the Analysis window. Figure 10-84 shows an example:

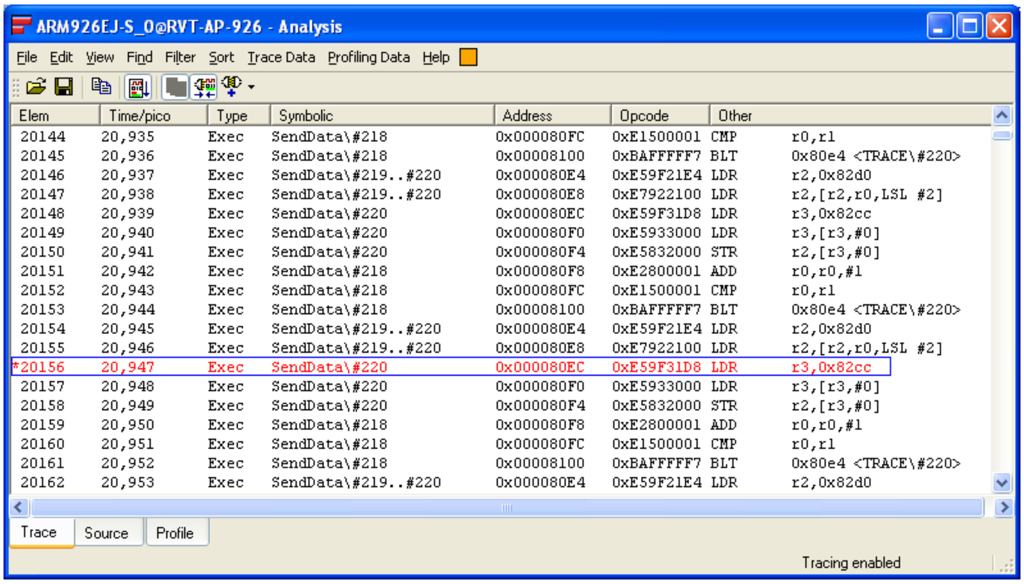

ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem  Time/pico  Type  Symbolic          Address     Opcode          Other
0     268        Exec  main\#78          0x00008270  0xE59F008C LDR  r0,0x8304
1     269        Exec  main\#78          0x00008274  0xE5801000 STR  r1,[r0,#0]
2     270        Exec  main\#79..#80     0x00008278  0xE5900000 LDR  r0,[r0,#0]
3     271        Exec  main\#79..#80     0x0000827C  0xE3500000 CMP  r0,#0
4     272        Exec  main\#80          0x00008280  0x1A000002 BNE  0x8290 <TRACE\#71>
5     273        Exec  main\#71          0x00008290  0xEAFFFFED B    0x824c <TRACE\#74>
6     274        Exec  main\#72..#74     0x0000824C  0xE59F00AC LDR  r0,0x8300
7     275        Exec  main\#72..#74     0x00008250  0xE5900000 LDR  r0,[r0,#0]
8     276        Exec  main\#74          0x00008254  0xE2800001 ADD  r0,r0,#1
9     277        Exec  main\#74          0x00008258  0xE59F10A0 LDR  r1,0x8300
10    278        Exec  main\#74          0x0000825C  0xE5810000 STR  r0,[r1,#0]
11    279        Exec  main\#75..#78     0x00008260  0xEB00004F BL   rand <0x83a4>
12    280        Exec  rand              0x000083A4  0xE59F304C LDR  r3,0x83f8 <rand+0x54>
13    281        Exec  rand              0x000083A8  0xE52DE004 PUSH {lr}
14    282        Exec  rand              0x000083AC  0xE59FC048 LDR  r12,0x83fc <rand+0x58>
15    283        Exec  rand              0x000083B0  0xE5932000 LDR  r2,[r3,#0]
16    284        Exec  rand              0x000083B4  0xE5930004 LDR  r0,[r3,#4]
17    285        Exec  rand              0x000083B8  0xE79CE102 LDR  lr,[r12,r2,LSL #2]
18    286        Exec  rand              0x000083BC  0xE79C1100 LDR  r1,[r12,r0,LSL #2]
```

Figure 10-84 Trace captured after a trigger

5. Select **Find Trigger** from the Analysis window **Find** menu. RealView Debugger searches the trace output and the line of trace corresponding to the trigger point is selected.

6. Scroll down the trace output to see the captured trace after the trigger point. Figure 10-85 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

```
Elem    Time/pico  Type  Symbolic             Address     Opcode          Other
20144   20,935     Exec  SendData\#218        0x000080FC  0xE1500001 CMP  r0,r1
20145   20,936     Exec  SendData\#218        0x00008100  0xBAFFFFF7 BLT  0x80e4 <TRACE\#220>
20146   20,937     Exec  SendData\#219..#220  0x000080E4  0xE59F21E4 LDR  r2,0x82d0
20147   20,938     Exec  SendData\#219..#220  0x000080E8  0xE7922100 LDR  r2,[r2,r0,LSL #2]
20148   20,939     Exec  SendData\#220        0x000080EC  0xE59F31D8 LDR  r3,0x82cc
20149   20,940     Exec  SendData\#220        0x000080F0  0xE5933000 LDR  r3,[r3,#0]
20150   20,941     Exec  SendData\#220        0x000080F4  0xE5832000 STR  r2,[r3,#0]
20151   20,942     Exec  SendData\#218        0x000080F8  0xE2800001 ADD  r0,r0,#1
20152   20,943     Exec  SendData\#218        0x000080FC  0xE1500001 CMP  r0,r1
20153   20,944     Exec  SendData\#218        0x00008100  0xBAFFFFF7 BLT  0x80e4 <TRACE\#220>
20154   20,945     Exec  SendData\#219..#220  0x000080E4  0xE59F21E4 LDR  r2,0x82d0
20155   20,946     Exec  SendData\#219..#220  0x000080E8  0xE7922100 LDR  r2,[r2,r0,LSL #2]
*20156  20,947     Exec  SendData\#220        0x000080EC  0xE59F31D8 LDR  r3,0x82cc
20157   20,948     Exec  SendData\#220        0x000080F0  0xE5933000 LDR  r3,[r3,#0]
20158   20,949     Exec  SendData\#220        0x000080F4  0xE5832000 STR  r2,[r3,#0]
20159   20,950     Exec  SendData\#218        0x000080F8  0xE2800001 ADD  r0,r0,#1
20160   20,951     Exec  SendData\#218        0x000080FC  0xE1500001 CMP  r0,r1
20161   20,952     Exec  SendData\#218        0x00008100  0xBAFFFFF7 BLT  0x80e4 <TRACE\#220>
20162   20,953     Exec  SendData\#219..#220  0x000080E4  0xE59F21E4 LDR  r2,0x82d0
```

Figure 10-85 Captured trace after the trigger

7. Click **Stop** on the Debug toolbar to stop execution.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10

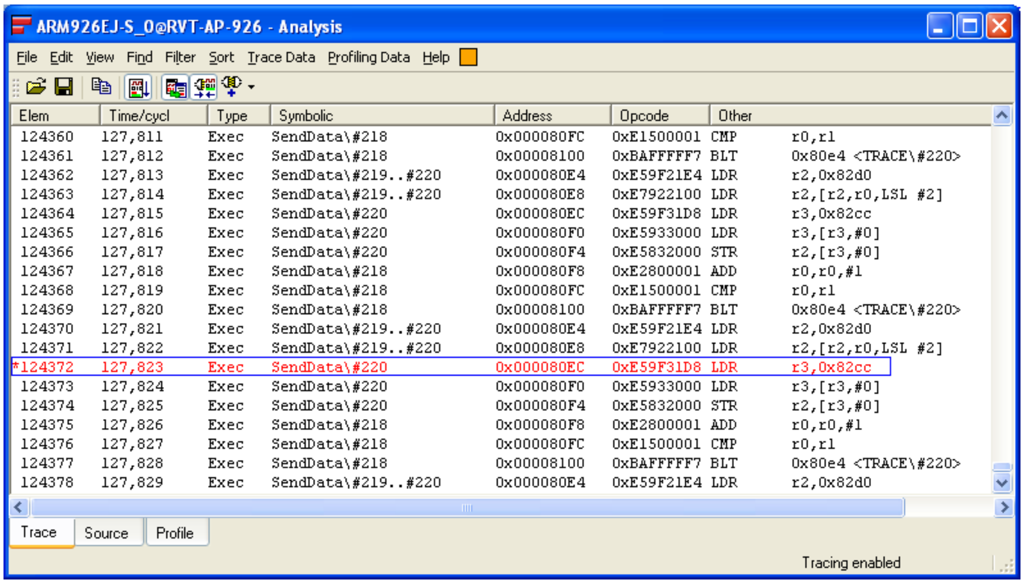
- *Trigger* on page 5-3
- *Setting a trigger point* on page 6-4
- Chapter 7 *Setting Conditional Tracepoints*
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `TRACEDATAWRITE` command.

10.15.5 Examining the captured trace

Examine the captured trace in the **Trace** tab:

- When collecting trace before a trigger point, a small amount of trace is often included after the trigger point.
- When collecting trace after a trigger point, a small amount of trace is often included before the trigger point.
- When collecting trace around a trigger point, the captured trace is distributed evenly before and after the trigger point. You can alter the amount of trace information before and after the trigger by using a trace start point to enable tracing before the trigger, and a trace end point to disable tracing after the trigger.
- If you choose to stop the processor when the trigger is activated, then the trace information that is collected when tracing around or after the trigger depends on your TPA. For RealView Trace used in this tutorial:
  1. Select **Edit** → **Trigger Mode** → **Stop Processor on Trigger** from the Analysis window menu.
  2. Repeat the steps described in *Capturing trace after a trigger* on page 10-80.

     Execution stops when the trigger is activated.
  3. Select **Find Trigger** from the **Find** menu to find the trace line for the trigger point. Figure 10-86 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Elem     Time/cycl  Type  Symbolic               Address      Opcode          Other
 124360  127,811    Exec  SendData\#218          0x000080FC   0xE1500001 CMP  r0,r1
 124361  127,812    Exec  SendData\#218          0x00008100   0xBAFFFFF7 BLT  0x80e4 <TRACE\#220>
 124362  127,813    Exec  SendData\#219..#220    0x000080E4   0xE59F21E4 LDR  r2,0x82d0
 124363  127,814    Exec  SendData\#219..#220    0x000080E8   0xE7922100 LDR  r2,[r2,r0,LSL #2]
 124364  127,815    Exec  SendData\#220          0x000080EC   0xE59F31D8 LDR  r3,0x82cc
 124365  127,816    Exec  SendData\#220          0x000080F0   0xE5933000 LDR  r3,[r3,#0]
 124366  127,817    Exec  SendData\#220          0x000080F4   0xE5832000 STR  r2,[r3,#0]
 124367  127,818    Exec  SendData\#218          0x000080F8   0xE2800001 ADD  r0,r0,#1
 124368  127,819    Exec  SendData\#218          0x000080FC   0xE1500001 CMP  r0,r1
 124369  127,820    Exec  SendData\#218          0x00008100   0xBAFFFFF7 BLT  0x80e4 <TRACE\#220>
 124370  127,821    Exec  SendData\#219..#220    0x000080E4   0xE59F21E4 LDR  r2,0x82d0
 124371  127,822    Exec  SendData\#219..#220    0x000080E8   0xE7922100 LDR  r2,[r2,r0,LSL #2]
*124372  127,823    Exec  SendData\#220          0x000080EC   0xE59F31D8 LDR  r3,0x82cc
 124373  127,824    Exec  SendData\#220          0x000080F0   0xE5933000 LDR  r3,[r3,#0]
 124374  127,825    Exec  SendData\#220          0x000080F4   0xE5832000 STR  r2,[r3,#0]
 124375  127,826    Exec  SendData\#218          0x000080F8   0xE2800001 ADD  r0,r0,#1
 124376  127,827    Exec  SendData\#218          0x000080FC   0xE1500001 CMP  r0,r1
 124377  127,828    Exec  SendData\#218          0x00008100   0xBAFFFFF7 BLT  0x80e4 <TRACE\#220>
 124378  127,829    Exec  SendData\#219..#220    0x000080E4   0xE59F21E4 LDR  r2,0x82d0
```

Figure 10-86 Captured trace for the port trigger

4. Select **Find Trigger** from the **Find** menu again. You might see a trace line for a second trigger point displayed. The first trigger point corresponds to the port trigger on the Trace Port, and the second trigger point corresponds to a protocol trigger that is encoded into the ETM protocol.

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Trigger* on page 5-3
- *Setting a trigger point* on page 6-4
- Chapter 7 *Setting Conditional Tracepoints*
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `TRACEDATAWRITE` command.

10.16 Capturing profiling information

This example demonstrates how you can use RealView Debugger to capture profiling information for your application. You use the profiling feature to get an overall view of the time spent in each function as a percentage of total execution time. You can profile the whole image, or specific parts of the image such as a loop.

The `dhrystone.axf` image used in this example performs a benchmarking sample that is executed *n* number of times, where you are prompted to indicate *n* when running the image. By performing multiple runs, you can sum the profiling data. This example assumes that you want to analyze the execution times of all functions that are executed in the main `for` loop that is run repeatedly. It also assumes that you are using an ETM-based target.

See also:

- *Tracing features used*
- *Capturing the initial profiling information*
- *Examining the profiling data* on page 10-86
- *Summing profiling data over multiple runs* on page 10-88
- *Profiling results obtained with 4-bit data width and Quad packing* on page 10-90.

10.16.1 Tracing features used

The following tracing features are used in this procedure:

- changing the trace buffer size
- capturing trace with ETM timestamping enabled
- capturing trace with trace start and end points
- analyzing the profiling data.

10.16.2 Capturing the initial profiling information

To capture profiling information using the `dhrystone` example:

1. Load the example image `dhrystone.axf` into the debugger. This file is located in the directory:

   `install_directory\RVDS\Examples\...\dhrystone\Debug`

   The tab for `dhry_1.c` is displayed in the Code window.

2. Select **Analysis Window** from the Code window **View** menu to view the Analysis window, and position the window so that it does not cover the Code window.

3. Configure the ETM for profiling:

   a. Select **Configure Analyzer Properties...** from the Analysis window **Edit** menu to display the Configure ETM dialog box.

   b. For timestamping, Trace buffer packing has the most effect on profiling resolution. Do one of the following:

      - Select **Normal packing** for the best profiling resolution. In this case, each packet contains one trace record.
      - Select **4 bit** for the Trace data width. The **Quad packing** setting is enabled.

      Note

      Select **Quad packing** if you want to receive multiple trace records that share the same timestamp. Be aware that this gives the poorest profiling resolution.

c. Select **Enable Timestamping**. To generate profiling information you must enable one of the settings that causes this to be captured. For the ETM, this can be either **Enable Timestamping** or **Cycle accurate tracing**.

——— **Note** ———

Be aware that cycle accurate tracing and timestamping might generate slightly different results depending on your trace settings and target speed.

———————————

d. Click **OK** to close the Configure ETM dialog box.

4. Set a trace start point at the start of the program loop as follows:

   a. Click the **dhry1.c** tab in the Code window to display the source view.

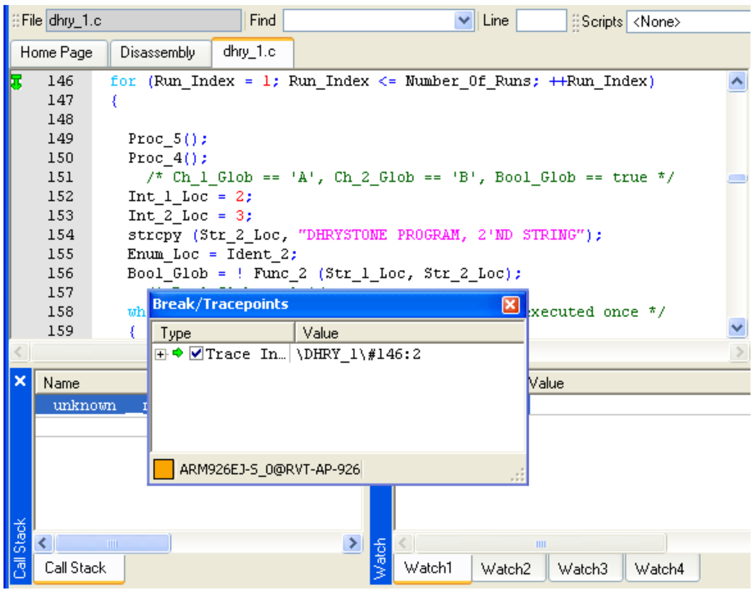
   b. Scroll down the source file until the code listing on line 146 is displayed.

   c. Right-click in the gray area to the left of the code listing in line 146 to display the context menu. This line represents the start of a `for` loop.

   d. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

   e. Select **Trace Start Point**.

   f. Click **OK** to close the New Tracepoint dialog. An arrow ⇩ is placed next to line 146 to indicate the start point you have set. Also, details of this tracepoint are displayed in the Break/Tracepoints view of the Code window. Figure 10-87 shows an example:



```
146  for (Run_Index = 1; Run_Index <= Number_Of_Runs; ++Run_Index)
147  {
148
149    Proc_5();
150    Proc_4();
151      /* Ch_1_Glob == 'A', Ch_2_Glob == 'B', Bool_Glob == true */
152    Int_1_Loc = 2;
153    Int_2_Loc = 3;
154    strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
155    Enum_Loc = Ident_2;
156    Bool_Glob = ! Func_2 (Str_1_Loc, Str_2_Loc);
157
158    wh                                          xecuted once */
159    {
```

Break/Tracepoints

Type	Value
⊞ ➡ ☑ Trace In…	\DHRY_1\#146:2

ARM926EJ-S_0@RVT-AP-926

Figure 10-87 Setting a trace start point

5. Set a trace end point after the end of the program loop as follows:

   a. Scroll down the source file and right-click in the gray area to the left of the code listing at line 204 to display the context menu.

      By placing the end point after the end of the loop, you ensure that RealView Debugger captures all the iterations of the loop, rather than a single iteration. In addition, the area bounded by the trace start and end points does not include any code that activates semihosting. The semihosting mechanism can affect the profiling results.

   b. Select **Insert Tracepoint...** from the context menu to display the New Tracepoint dialog box.

c. Select **Trace End Point**.

d. Click **OK** to close the New Tracepoint dialog box. An arrow is placed next to line 204 to indicate the end point you have set, and details of this control point are displayed in the Break/Tracepoints view of the Code window.

Because you have set trace start and end points, and not a trace range, you are instructing RealView Debugger to capture and display all trace information between the start and end points, including any instructions that might be branched to between the points.

6. Click **Run** on the Debug toolbar to start execution. The image is executed and a prompt is displayed in the Output view at the bottom of the Code window.

7. In the Output view, enter the number of runs through the benchmark you want RealView Debugger to perform. In this example, type **5000** and press Enter. Trace capture begins for the area of execution you have specified using tracepoints, that is, for the `for` loop area of code. The Analysis window is updated with the results of the trace capture.

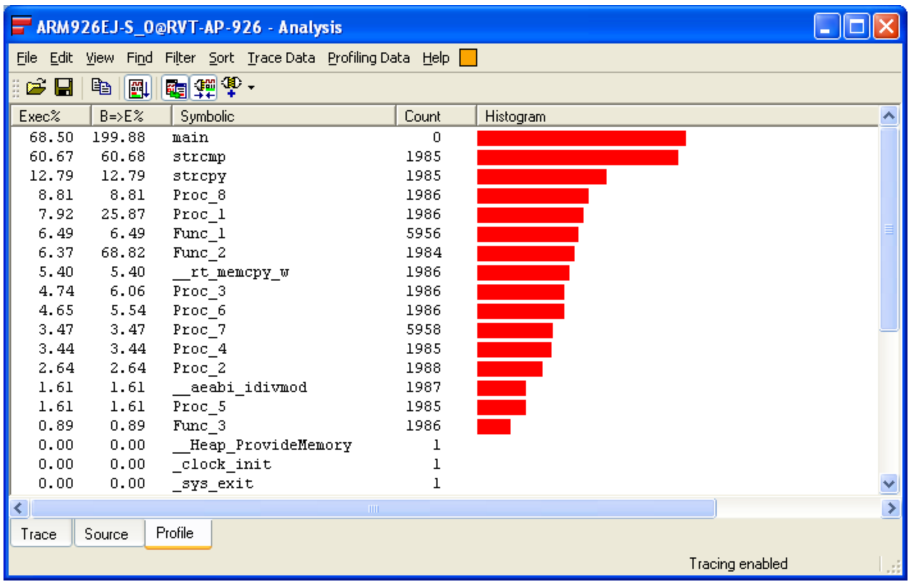
See also

- *Profiling trace information* on page 2-22
- *Profiling in RealView Debugger* on page 9-3
- *Changing the columns displayed in the Trace view* on page 9-4
- *Interleaving source with the trace output* on page 9-7
- *Viewing captured profiling information* on page 9-30
- *Using trace start and end points to trace and time a function* on page 10-18
- Chapter 4 *Configuring the ETM*
- Chapter 6 *Setting Unconditional Tracepoints*.

10.16.3 Examining the profiling data

To examine the profiling information:

1. In the Analysis window, click the **Profile** tab to display the profile data. Figure 10-88 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

Exec%	B=>E%	Symbolic	Count	Histogram
68.50	199.88	main	0	🟥
60.67	60.68	strcmp	1985	🟥
12.79	12.79	strcpy	1985	🟥
8.81	8.81	Proc_8	1986	🟥
7.92	25.87	Proc_1	1986	🟥
6.49	6.49	Func_1	5956	🟥
6.37	68.82	Func_2	1984	🟥
5.40	5.40	__rt_memcpy_w	1986	🟥
4.74	6.06	Proc_3	1986	🟥
4.65	5.54	Proc_6	1986	🟥
3.47	3.47	Proc_7	5958	🟥
3.44	3.44	Proc_4	1985	🟥
2.64	2.64	Proc_2	1988	🟥
1.61	1.61	__aeabi_idivmod	1987	🟥
1.61	1.61	Proc_5	1985	🟥
0.89	0.89	Func_3	1986	🟥
0.00	0.00	__Heap_ProvideMemory	1	
0.00	0.00	_clock_init	1	
0.00	0.00	_sys_exit	1	

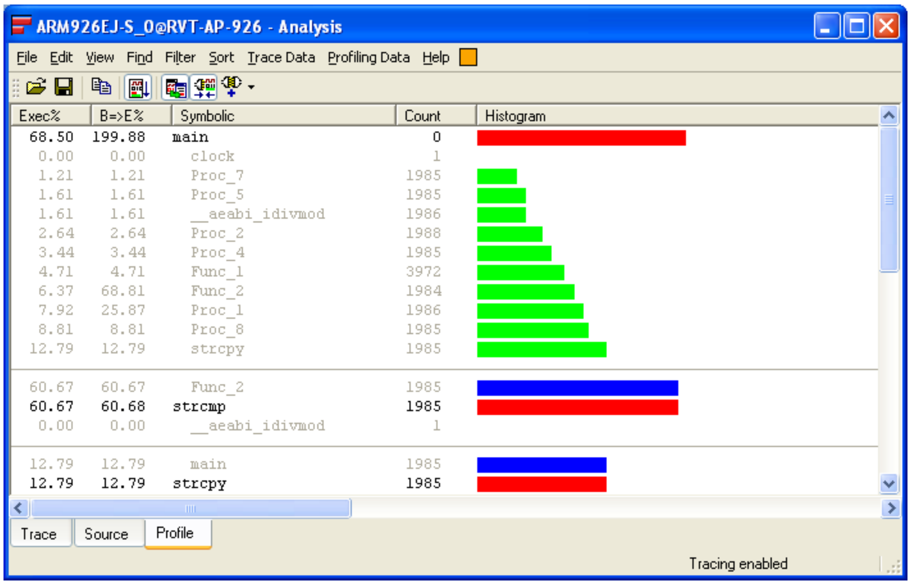
Figure 10-88 Results displayed in the Profile tab (Normal packing)

This example shows the results with Normal packing.

In the **Profile** tab, as shown in Figure 10-88 on page 10-86, the execution times for all functions accessed during the `for` loop are displayed, and a graphical representation of their respective execution times is shown in the `Histogram` column.

2. To view call-graph data for the results:
   a. Select **Parents of Function** from the **Profiling Data** menu of the Analysis window.
   b. Select **Children of Function** from the **Profiling Data** menu of the Analysis window.

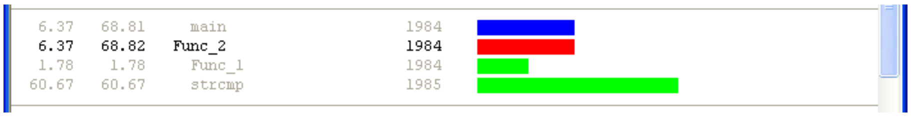
   The Profile view displays the parents and children of each function. Figure 10-89 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace Source Profile
Tracing enabled

Exec%	B=>E%	Symbolic	Count	Histogram
68.50	199.88	main	0	🟥
0.00	0.00	clock	1	
1.21	1.21	Proc_7	1985	🟩
1.61	1.61	Proc_5	1985	🟩
1.61	1.61	__aeabi_idivmod	1986	🟩
2.64	2.64	Proc_2	1988	🟩
3.44	3.44	Proc_4	1985	🟩
4.71	4.71	Func_1	3972	🟩
6.37	68.81	Func_2	1984	🟩
7.92	25.87	Proc_1	1986	🟩
8.81	8.81	Proc_8	1985	🟩
12.79	12.79	strcpy	1985	🟩
60.67	60.67	Func_2	1985	🟦
60.67	60.68	strcmp	1985	🟥
0.00	0.00	__aeabi_idivmod	1	
12.79	12.79	main	1985	🟦
12.79	12.79	strcpy	1985	🟥

Figure 10-89 Call-graph data displayed in the Profile tab

In addition to displaying the execution times for each function accessed during the `for` loop, the execution times of the parents and children of these functions are also displayed. Figure 10-90 shows an example for a single function, `Func_2`.



6.37	68.81	main	1984	🟦
6.37	68.82	Func_2	1984	🟥
1.78	1.78	Func_1	1984	🟩
60.67	60.67	strcmp	1985	🟩

Figure 10-90 Call-graph data for Func_2

The `Exec%` column shows that:

- 6.37% of the total execution time was spent in code of the function `Func_2` when called from the parent `main`.
- 6.37% of the total execution time was spent in code of the function `Func_2`.
- 1.78% of the total execution time was spent in code of the function `Func_1` when called as a child from `Func_2`.
- 60.67% of the total execution time was spent in code of the function `strcmp` when called as a child from `Func_2`.

The `B=>E%` column shows that:

- 68.81% of the total execution time was spent in calls to the function `Func_2` and its children when called from the parent `main`.
- 68.82% of the total execution time was spent in calls to the function `Func_2` and its children.

- 1.78% of the total execution time was spent in calls to the function `Func_1` called as a child from `Func_2`.
- 60.67% of the total execution time was spent in calls to the function `strcmp` called as a child from `Func_2`.

The `Count` column shows that:

- There were 1984 calls from the function `main` to the function `Func_2`.
- There were 1984 calls to the function `Func_2`.
- There were 1984 calls to the function `Func_1` from the function `Func_2`.
- There were 1985 calls to the function `strcmp` from the function `Func_2`.

See also

- *Profiling trace information* on page 2-22
- *Profiling in RealView Debugger* on page 9-3
- *Changing the columns displayed in the Trace view* on page 9-4
- *Interleaving source with the trace output* on page 9-7
- *Viewing captured profiling information* on page 9-30
- *Using trace start and end points to trace and time a function* on page 10-18
- Chapter 4 *Configuring the ETM*
- Chapter 6 *Setting Unconditional Tracepoints*.

10.16.4 Summing profiling data over multiple runs

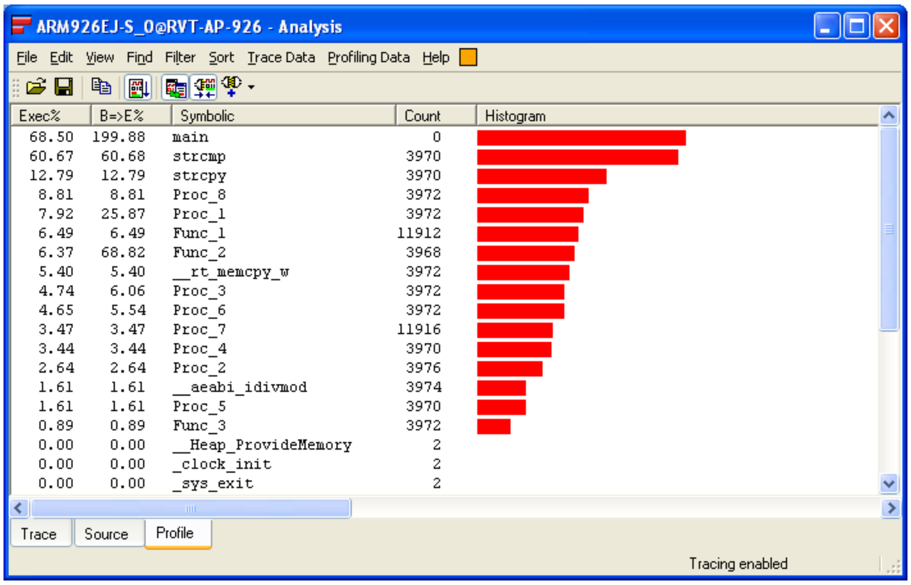
To sum profiling data over multiple runs, do the following:

1. If you have not yet done so, perform the procedure described in *Capturing the initial profiling information* on page 10-84.
2. Select **Sum Profiling Data** from the **Profiling Data** menu of the Analysis window.
3. Select **Set PC to Entry Point** from the **Debug** menu of the Code window.

   Note

   If you reload the image, or load a new image, the trace details are cleared and **Sum Profiling Data** is disabled.

4. Hide the call-graph data for the previous results:
   a. Select **Parents of Function** from the **Profiling Data** menu of the Analysis window.
   b. Select **Children of Function** from the **Profiling Data** menu of the Analysis window.
5. Click **Run** on the Debug toolbar of the Code window to start execution.
6. When prompted, enter **5000** for the number of runs through the benchmark. The profiling data is updated. Figure 10-91 on page 10-89 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Exec%	B=>E%	Symbolic	Count	Histogram
68.50	199.88	main	0	
60.67	60.68	strcmp	3970	
12.79	12.79	strcpy	3970	
8.81	8.81	Proc_8	3972	
7.92	25.87	Proc_1	3972	
6.49	6.49	Func_1	11912	
6.37	68.82	Func_2	3968	
5.40	5.40	__rt_memcpy_w	3972	
4.74	6.06	Proc_3	3972	
4.65	5.54	Proc_6	3972	
3.47	3.47	Proc_7	11916	
3.44	3.44	Proc_4	3970	
2.64	2.64	Proc_2	3976	
1.61	1.61	__aeabi_idivmod	3974	
1.61	1.61	Proc_5	3970	
0.89	0.89	Func_3	3972	
0.00	0.00	__Heap_ProvideMemory	2	
0.00	0.00	_clock_init	2	
0.00	0.00	_sys_exit	2	

Figure 10-91 Summing profiling data over multiple runs

Observations on summing profiling data over multiple runs

Compare Figure 10-91 with Figure 10-88 on page 10-86. The summing of profiling data reveals the following information:

Count column

The `Count` column reveals the following:

- After the first run:
  - functions `Proc_7` and `Func_1` were called the most times
  - all other functions were called a similar number of times.
- The second run shows the same trend as the first run.

B=>E% column

The `B=>E%` column reveals the following:

- The total execution time after two runs is equal to the total execution time of the first run, plus the total execution time of the second run.
- The time taken to execute a particular function after two runs is equal to the time taken to execute it after the first run.
- The percentage of the total execution time that is spent in the code of a particular function is the same as that after the first run. However, this might not always be the case.

Exec% column

The `Exec%` column reveals the following:

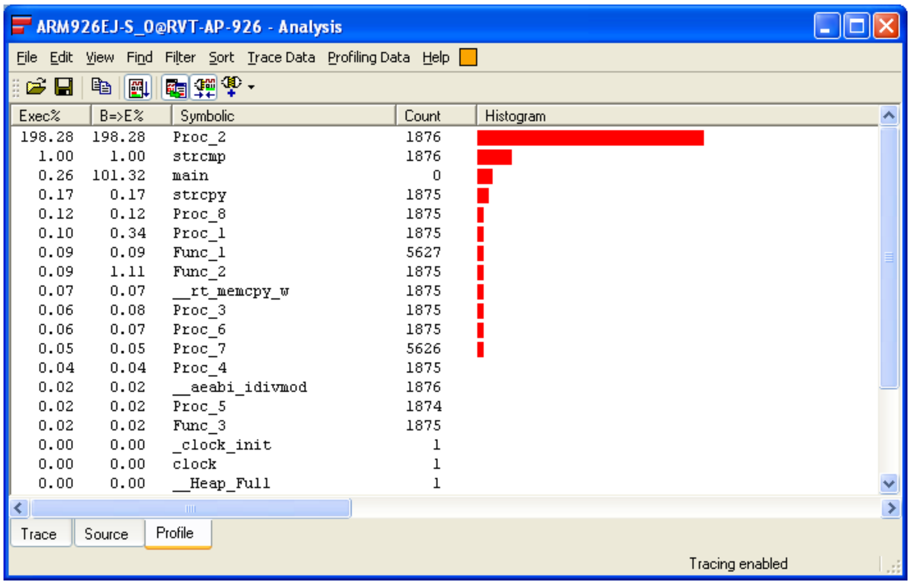
- The total execution time after two runs is equal to the total execution time of the first run, plus the total execution time of the second run.
- The time spent from entry to exit of a particular function after two runs is equal to the time spent from entry to exit of that function after the first run.
- The percentage of the total execution time that is spent in calls to a particular function is the same as that after the first run. However, this might not always be the case.

See also

- *Profiling trace information* on page 2-22
- *Profiling in RealView Debugger* on page 9-3
- *Changing the columns displayed in the Trace view* on page 9-4
- *Interleaving source with the trace output* on page 9-7
- *Viewing captured profiling information* on page 9-30
- *Using trace start and end points to trace and time a function* on page 10-18
- Chapter 4 *Configuring the ETM*
- Chapter 6 *Setting Unconditional Tracepoints*.

10.16.5 Profiling results obtained with 4-bit data width and Quad packing

Figure 10-92 shows the profiling results obtained with a 4-bit trace data width and Quad packing. Compare the results with those in Figure 10-88 on page 10-86.


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

```
Exec%    B=>E%    Symbolic          Count   Histogram
198.28   198.28   Proc_2            1876
  1.00     1.00   strcmp            1876
  0.26   101.32   main                 0
  0.17     0.17   strcpy            1875
  0.12     0.12   Proc_8            1875
  0.10     0.34   Proc_1            1875
  0.09     0.09   Func_1            5627
  0.09     1.11   Func_2            1875
  0.07     0.07   __rt_memcpy_w     1875
  0.06     0.08   Proc_3            1875
  0.06     0.07   Proc_6            1875
  0.05     0.05   Proc_7            5626
  0.04     0.04   Proc_4            1875
  0.02     0.02   __aeabi_idivmod   1876
  0.02     0.02   Proc_5            1874
  0.02     0.02   Func_3            1875
  0.00     0.00   _clock_init          1
  0.00     0.00   clock                1
  0.00     0.00   __Heap_Full          1
```

Figure 10-92 Results displayed in the Profile tab (Quad packing)

See also

- *Profiling trace information* on page 2-22
- *Profiling in RealView Debugger* on page 9-3
- *Changing the columns displayed in the Trace view* on page 9-4
- *Interleaving source with the trace output* on page 9-7
- *Viewing captured profiling information* on page 9-30
- *Using trace start and end points to trace and time a function* on page 10-18
- Chapter 4 *Configuring the ETM*
- Chapter 6 *Setting Unconditional Tracepoints*.

10.17 Setting up a conditional tracepoint

This example demonstrates how you can set up a conditional tracepoint. It uses the `dhrystone.axf` image described in *Capturing profiling information* on page 10-84. This example assumes that you are connected to an ETM-based target, and that you want to trigger trace in `Proc_2`, but only after 50 executions of the function `Proc_1`.

Note

You cannot set conditional tracepoints on *RealView ARMulator® ISS* (RVISS) targets or DSPs.

See also:
- *Tracing features used*
- *Procedure*.

10.17.1 Tracing features used

The following tracing features are used in this procedure:
- using the Trace on X after Y executed N times dialog box
- examining details of tracepoints in the Break/Tracepoints view
- setting a trigger tracepoint
- collecting trace before a trigger.

10.17.2 Procedure

To set up a conditional tracepoint using the `dhrystone` example:

1. Load the example image `dhrystone.axf` into the debugger. This file is located in the directory:

   `*install_directory*\RVDS\Examples\...\dhrystone\Debug`

   The tab for `dhry_1.c` is displayed in the Code window.

2. Select **Debug** → **Tracepoints** → **Trace on X after Y executed N times...** from the Code window main menu to display the Trace on X after Y executed N times dialog box. Figure 10-93 shows an example:

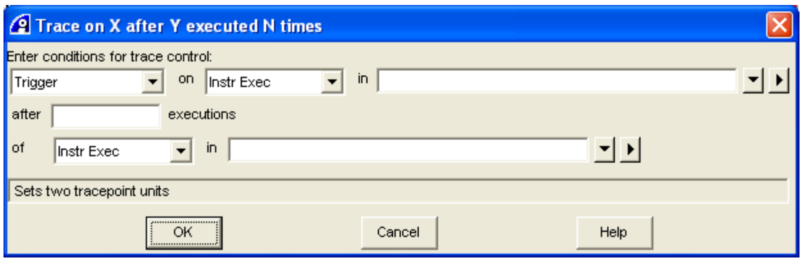

Trace on X after Y executed N times
Enter conditions for trace control:
Trigger
on
Instr Exec
in
after
executions
of
Instr Exec
in
Sets two tracepoint units
OK
Cancel
Help

Figure 10-93 Trace on X after Y executed N times dialog box

3. Set up the first part of the tracepoint as follows:
   a. Select **Trigger** from the first drop-down list.
   b. Select **Instr Exec** from the second drop-down list.
   c. Click the drop-down arrow ▾ to the right of the text box to display a menu of items saved in your personal history file.
   d. Select **Function list...** from this menu to display the Function List dialog box.
   e. Select **DHRY_1\Proc_2 of @dhrystone** from the list of functions.

This selects the function `Proc_2` as the function to trigger on.

f. Click **Select** to close the Function List dialog box.

g. Type **50** into the text box on the second line of the dialog box, to specify that Proc_1 is to be executed 50 times.

h. Select **Instr Exec** from the drop-down list in the last line of the dialog box.

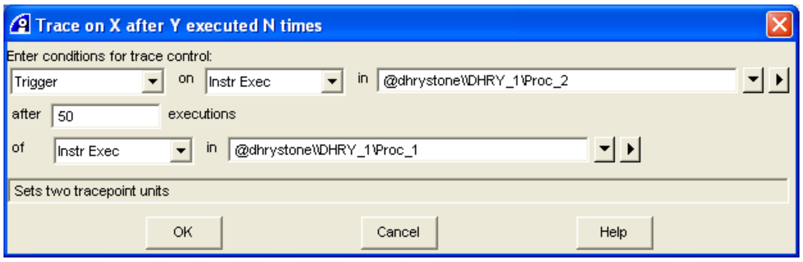
4. Set up the last part of the tracepoint as follows:

a. Click the drop-down arrow to the right of the text box to display a menu of items saved in your personal history file.

b. Select **Function list...** from this menu to display the Function List dialog box.

c. Select **DHRY_1\Proc_1 of @dhrystone** from the list of functions. This selects the function `Proc_1` as the function that must be executed 50 times before the trigger can occur.

d. Click **Select** to close the Function List dialog box. The Trace on X after Y executed N times dialog box is now complete. Figure 10-94 shows an example:



Trace on X after Y executed N times

Enter conditions for trace control:

Trigger on Instr Exec in @dhrystone\DHRY_1\Proc_2

after 50 executions

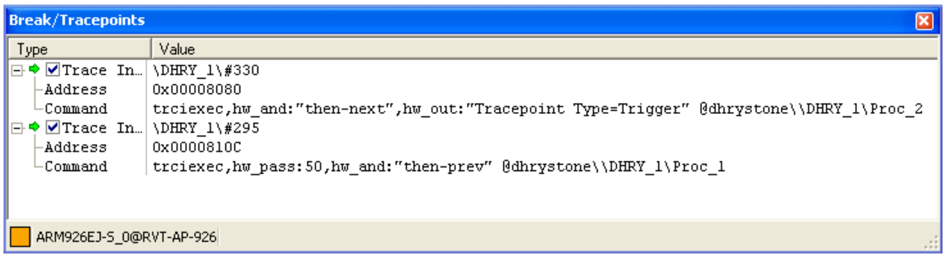
of Instr Exec in @dhrystone\DHRY_1\Proc_1

Sets two tracepoint units

Figure 10-94 Completed conditional tracepoint dialog box

e. Click **OK** to set the tracepoint as specified.

5. If you want to view the tracepoint details, select **Break/Tracepoints** from the Code window **View** menu. The Break/Tracepoints view is displayed. Figure 10-95 shows an example:



Break/Tracepoints

```
Type           Value
☑ Trace In...  \DHRY_1\#330
  Address      0x00008080
  Command      trciexec,hw_and:"then-next",hw_out:"Tracepoint Type=Trigger" @dhrystone\\DHRY_1\Proc_2
☑ Trace In...  \DHRY_1\#295
  Address      0x0000810C
  Command      trciexec,hw_pass:50,hw_and:"then-prev" @dhrystone\\DHRY_1\Proc_1
```

ARM926EJ-S_0@RVT-AP-926

Figure 10-95 Break/Tracepoints view with conditional tracepoints set

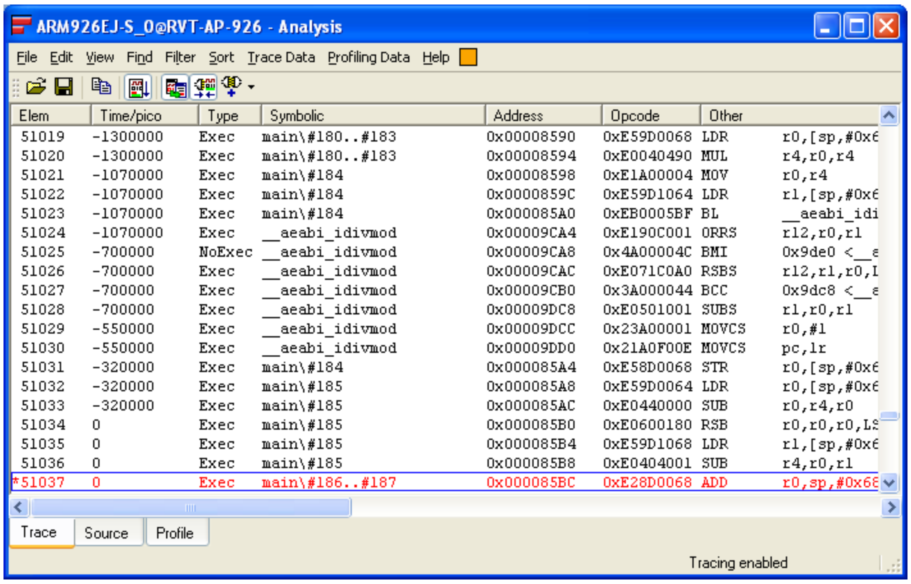
6. Select **Analysis Window** from the **View** menu of the Code window to display the Analysis window.

7. Select **Edit** → **Trigger Mode** → **Collect Trace Before Trigger** from the Analysis window menu. This is the default.

8. Click **Run** on the Debug toolbar of the Code window to start execution. Image execution starts, and a prompt is displayed in the **StdIO** tab of the Output view at the bottom of the Code window.

9. Enter **100** at the prompt for the number of runs through the benchmark. The results of the trace capture are displayed in the Analysis window.

10. Select **Find Trigger** from the **Find** menu to display the trace line for the trigger point. Figure 10-96 shows an example:


ARM926EJ-S_0@RVT-AP-926 - Analysis
File Edit View Find Filter Sort Trace Data Profiling Data Help
Trace
Source
Profile
Tracing enabled

Elem	Time/pico	Type	Symbolic	Address	Opcode	Other
51019	-1300000	Exec	main\#180..#183	0x00008590	0xE59D0068 LDR	r0,[sp,#0x6
51020	-1300000	Exec	main\#180..#183	0x00008594	0xE0040490 MUL	r4,r0,r4
51021	-1070000	Exec	main\#184	0x00008598	0xE1A00004 MOV	r0,r4
51022	-1070000	Exec	main\#184	0x0000859C	0xE59D1064 LDR	r1,[sp,#0x6
51023	-1070000	Exec	main\#184	0x000085A0	0xEB0005BF BL	__aeabi_idi
51024	-1070000	Exec	__aeabi_idivmod	0x00009CA4	0xE190C001 ORRS	r12,r0,r1
51025	-700000	NoExec	__aeabi_idivmod	0x00009CA8	0x4A00004C BMI	0x9de0 <__a
51026	-700000	Exec	__aeabi_idivmod	0x00009CAC	0xE071C0A0 RSBS	r12,r1,r0,L
51027	-700000	Exec	__aeabi_idivmod	0x00009CB0	0x3A000044 BCC	0x9dc8 <__a
51028	-700000	Exec	__aeabi_idivmod	0x00009DC8	0xE0501001 SUBS	r1,r0,r1
51029	-550000	Exec	__aeabi_idivmod	0x00009DCC	0x23A00001 MOVCS	r0,#1
51030	-550000	Exec	__aeabi_idivmod	0x00009DD0	0x21A0F00E MOVCS	pc,lr
51031	-320000	Exec	main\#184	0x000085A4	0xE58D0068 STR	r0,[sp,#0x6
51032	-320000	Exec	main\#185	0x000085A8	0xE59D0064 LDR	r0,[sp,#0x6
51033	-320000	Exec	main\#185	0x000085AC	0xE0440000 SUB	r0,r4,r0
51034	0	Exec	main\#185	0x000085B0	0xE0600180 RSB	r0,r0,r0,LS
51035	0	Exec	main\#185	0x000085B4	0xE59D1068 LDR	r1,[sp,#0x6
51036	0	Exec	main\#185	0x000085B8	0xE0404001 SUB	r4,r0,r1
*51037	0	Exec	main\#186..#187	0x000085BC	0xE28D0068 ADD	r0,sp,#0x68

Figure 10-96 Trace line for the trigger point

See also

- *Configuring how trace information is collected at a trigger* on page 3-10
- *Trigger* on page 5-3
- *Setting a trigger point* on page 6-4
- Chapter 7 *Setting Conditional Tracepoints*.

Appendix A
Setting up the Trace Hardware

This appendix describes how to set up the hardware for the trace configurations supported by RealView® Debugger. See Chapter 2 *Getting Started with Tracing* for details on how trace hardware components interact with RealView Debugger to enable you to perform tracing.

When setting up the trace capture system, do not exceed the timing specifications of the target *Embedded Trace Macrocell*™ (ETM™) or of the *Trace Port Analyzer* (TPA). See *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities* for more information.

This appendix contains the following sections:

A.1 RealView Trace and RealView ICE

The RealView Trace analyzer is available from ARM. When used with RealView ICE, it forms a complete trace solution. You connect it to your development board using the supplied ribbon cable and adaptor, and to the host workstation using a 10BaseT ethernet cable.

——— **Note** ———

To capture trace directly from an ETM, you must use a RealView Trace unit.

———————————

To set up your hardware to enable tracing in RealView Debugger as shown in Figure A-1, connect and configure the RealView ICE and RealView Trace units.

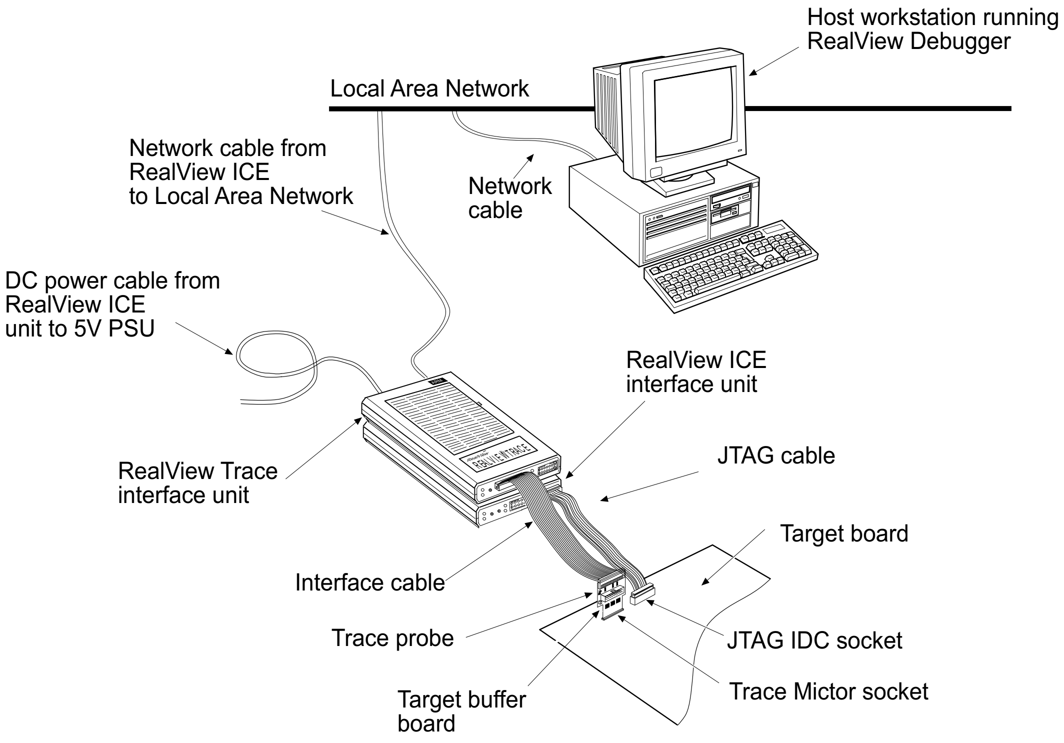

Host workstation running RealView Debugger
Local Area Network
Network cable from RealView ICE to Local Area Network
Network cable
DC power cable from RealView ICE unit to 5V PSU
RealView ICE interface unit
JTAG cable
RealView Trace interface unit
Target board
Interface cable
Trace probe
JTAG IDC socket
Trace Mictor socket
Target buffer board

Figure A-1 Connections for RealView ICE and RealView Trace

——— **Note** ———

If the target buffer board contains a 20-way JTAG IDC connector, it is suggested that you use it. However, if the target board has only a Mictor socket, and no JTAG IDC socket, then you must use the JTAG IDC socket on the target buffer board.

———————————

See also:

- *Configuring trace capture from an ETM with RealView Trace* on page B-2
- *Configuring trace capture from a CoreSight ETM with RealView Trace* on page B-5
- the following in the *RealView Debugger User Guide*:
  — Chapter 3 *Target Connection*
- *RealView ICE and RealView Trace Setting Up the Hardware*.

A.2 Embedded Trace Buffer and RealView ICE

If you are tracing a target that has an *Embedded Trace Buffer*™ (ETB™), then you do not require a *Trace Port Analyzer* (TPA). However, you must configure RealView ICE to recognize the ETB.

See also:

- *Configuring trace capture from an ETB with DSTREAM or RealView ICE* on page B-3
- *Configuring trace capture from a CoreSight ETB with DSTREAM and RealView ICE* on page B-7
- *RealView ICE and RealView Trace Setting Up the Hardware*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*.

A.3 Embedded Trace Buffer and DSTREAM

If you are tracing a target that has an *Embedded Trace Buffer*™ (ETB™), then you must configure DSTREAM to recognize the ETB.

See also:

- *Configuring trace capture from an ETB with DSTREAM or RealView ICE* on page B-3
- *Configuring trace capture from a CoreSight ETB with DSTREAM and RealView ICE* on page B-7
- *DSTREAM Setting Up the Hardware*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*.

Appendix B
Setting up the Trace Software

This appendix describes how to set up the software for the configurations of trace that are supported by RealView® Debugger. These instructions assume you have set up the hardware as described in Appendix A *Setting up the Trace Hardware*.

This appendix contains the following sections:

——— **Note** ———

This document assumes that you have installed the DSTREAM and RealView ICE v4.0 host software, and updated the DSTREAM or RealView ICE firmware. For RealView ICE, you must use firmware v3.2 or later.

————————

See also:

- the following in the *RealView Debugger User Guide*:
  - — Chapter 3 *Target Connection* for general information on connecting to targets.
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*.

B.1 Configuring trace capture from an ETM with RealView Trace

Before you can capture trace from an *Embedded Trace Macrocell*™ (ETM™), you must set up RealView Debugger and RealView Trace in conjunction with RealView ICE.

To capture trace from an ETM:

1. Connect the TPA hardware.
2. Install the RealView ICE host software.

   Note

   The RealView ICE host software is included with RVDS, and does not require a separate installation.

3. Select **Start** → **Programs** → **ARM** → **v4.1** → **RealView Debugger v4.1** to start RealView Debugger.
4. Configure RealView ICE to identify the targets on your development platform.
5. Connect to the required target on your development platform.
6. Connect to the TPA.
7. Configure the ETM, if required.

See also:

- *Connecting to an analyzer* on page 2-17
- *RealView Trace and RealView ICE* on page A-2
- Chapter 4 *Configuring the ETM*
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
  - *Connecting to a target* on page 3-27
- *RealView ICE and RealView Trace Setting Up the Hardware*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*.

B.2 Configuring trace capture from an ETB with DSTREAM or RealView ICE

Before you can capture trace from an *Embedded Trace Buffer*™ (ETB™), you must set up RealView Debugger and the DSTREAM or RealView ICE debug interface unit.

The difference between configuring trace capture from an ETB for use with DSTREAM or RealView ICE is the Debug Interface that you use. The following procedure uses the `RealView ICE` Debug Interface as an example.

To capture trace from an ETB:

1. Make sure the DSTREAM and RealView ICE host software is installed.

   Note

   The DSTREAM and RealView ICE host software is included with RVDS, and does not require a separate installation.

2. Connect the debug interface unit.
3. Select **Start** → **Programs** → **ARM** → **RealView Development Suite v4.1** → **RealView Debugger v4.1** to start RealView Debugger.
4. In the Code window, select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
5. Right-click on the `RealView-ICE` Debug Configuration to display the context menu.
6. Select **Configure...** from the context menu to display the RVConfig utility. Figure B-1 shows an example:

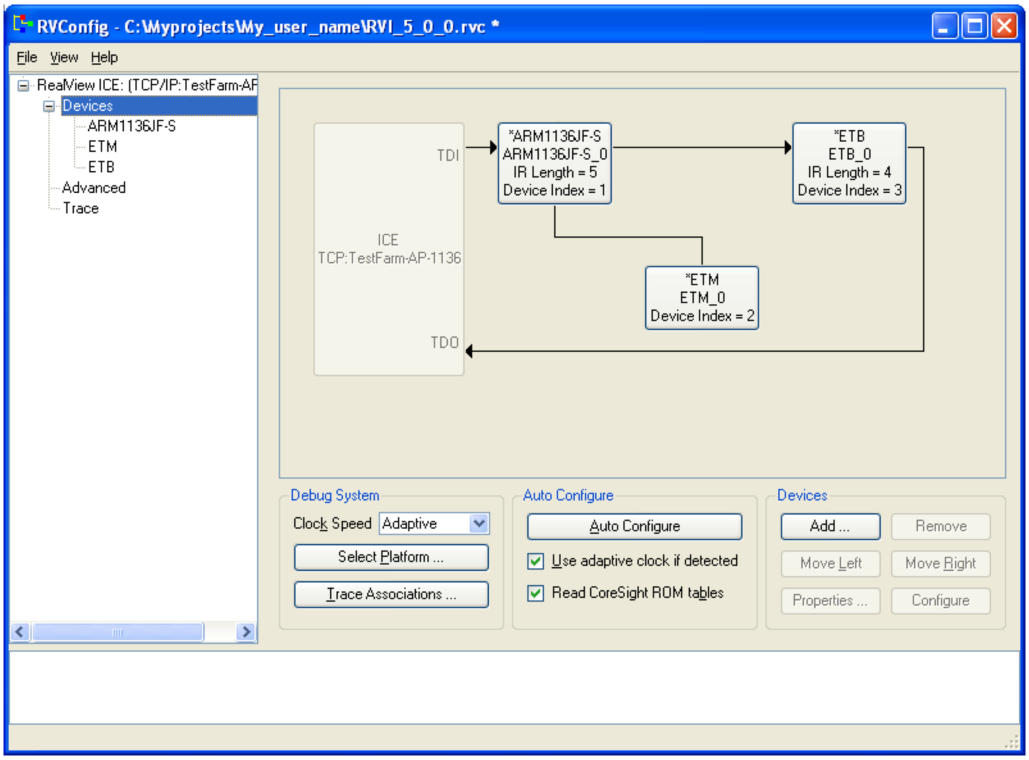
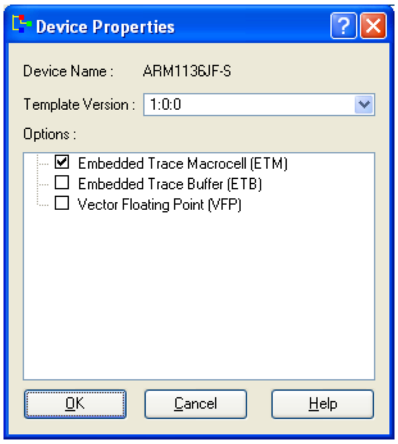


Figure B-1 RVConfig utility

7. Right-click on the box in the scan chain schematic diagram corresponding to the processor that you are tracing.

8. Select **Properties...** from the context menu to display the Device Properties dialog box. Figure B-2 shows an example:



Device Name : ARM1136JF-S
Template Version : 1:0:0
Options :
- [✓] Embedded Trace Macrocell (ETM)
- [ ] Embedded Trace Buffer (ETB)
- [ ] Vector Floating Point (VFP)

Figure B-2 Device Properties dialog box

9. Select **Embedded Trace Buffer (ETB)** from the list.

10. Click **OK** to close the Device Properties dialog box.

11. Select **Save** from the RVConfig **File** menu.

12. Select **Exit** from the **File** menu to close the RVConfig utility.

13. Connect to the target in the usual way.

See also:
- *Embedded Trace Buffer and RealView ICE* on page A-3
- *Embedded Trace Buffer and DSTREAM* on page A-4
- *DSTREAM Setting Up the Hardware*
- *RealView ICE and RealView Trace Setting Up the Hardware*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*.

B.3 Configuring trace capture from a CoreSight ETM with RealView Trace

Before you can capture trace from a CoreSight™ ETM, you must set up RealView Debugger and RealView Trace in conjunction with RealView ICE.

To capture trace from a CoreSight ETM:

1. Connect the TPA hardware.
2. Install the RealView ICE host software.

   ——— Note ———

   The RealView ICE host software is included with RVDS, and does not require a separate installation.

   ———————

3. Select **Start** → **Programs** → **ARM** → **RealView Development Suite v4.1** → **RealView Debugger v4.1** to start RealView Debugger.
4. Configure RealView ICE to identify the processors and CoreSight components on your development platform.

   ——— Note ———

   If your development platform has multiple trace sources, then you must add the CoreSight Trace Funnel (`CSTFunnel`).

   ———————

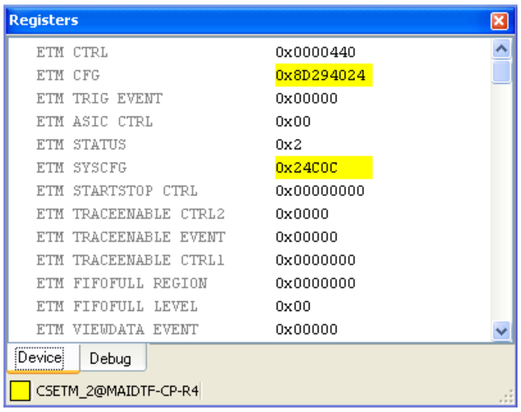
5. Select **Registers** from the **View** menu to display the Registers view.
6. If your development platform has multiple trace sources, then you must configure the CoreSight Trace Funnel to identify the source from which trace is to be captured:
   a. Connect to the `CSTFunnel_`*n* target.
   b. Set up the Trace Funnel registers.

   ——— Note ———

   If your development platform contains multiple processors and multiple ETMs, only one CoreSight ETM at a time can be chosen as the trace source in this release.

   ———————

7. Configure the CoreSight ETM registers:
   a. Connect to the `CSETM_`*n* target.
   b. Set up the CSETM registers. Figure B-3 shows an example:

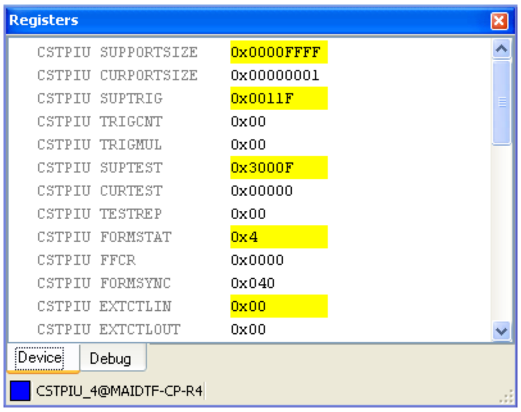


```
ETM CTRL                  0x0000440
ETM CFG                   0x8D294024
ETM TRIG EVENT            0x00000
ETM ASIC CTRL             0x00
ETM STATUS                0x2
ETM SYSCFG                0x24C0C
ETM STARTSTOP CTRL        0x00000000
ETM TRACEENABLE CTRL2     0x0000
ETM TRACEENABLE EVENT     0x00000
ETM TRACEENABLE CTRL1     0x0000000
ETM FIFOFULL REGION       0x0000000
ETM FIFOFULL LEVEL        0x00
ETM VIEWDATA EVENT        0x00000
```

CSETM_2@MAIDTF-CP-R4

Figure B-3 CSETM registers in the Registers view

8. Configure the CoreSight *Trace Port Interface Unit* (TPIU) registers:
   a. Connect to the `CSTPIU_n` target.
   b. Set up the TPIU registers. Figure B-4 shows an example:



```
CSTPIU SUPPORTSIZE    0x0000FFFF
CSTPIU CURPORTSIZE    0x00000001
CSTPIU SUPTRIG        0x0011F
CSTPIU TRIGCNT        0x00
CSTPIU TRIGMUL        0x00
CSTPIU SUPTEST        0x3000F
CSTPIU CURTEST        0x00000
CSTPIU TESTREP        0x00
CSTPIU FORMSTAT       0x4
CSTPIU FFCR           0x0000
CSTPIU FORMSYNC       0x040
CSTPIU EXTCTLIN       0x00
CSTPIU EXTCTLOUT      0x00
```

Figure B-4 CSTPIU registers in the Registers view

9. Connect to the target processor on your development platform.

10. Connect to the TPA.

11. Configure the ETM.

See also:

- *RealView ICE and RealView Trace Setting Up the Hardware*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*
- *Connecting to an analyzer* on page 2-17
- Chapter 4 *Configuring the ETM*
- the following in the *RealView Debugger User Guide*:
  — Creating a Debug Configuration on page 3-10
  — Connecting to a target on page 3-19.

B.4 Configuring trace capture from a CoreSight ETB with DSTREAM and RealView ICE

Before you can capture trace from a CoreSight *Embedded Trace Buffer* (CoreSight ETB), you must set up RealView Debugger and the DSTREAM or RealView ICE debug interface unit.

The difference between configuring trace capture from a CoreSight ETB for use with DSTREAM or RealView ICE is the Debug Interface that you use. The following procedure uses the `RealView ICE` Debug Interface as an example.

To capture trace from a CoreSight ETB:

1. Connect the debug interface unit.
2. Install the RealView ICE host software.

   Note

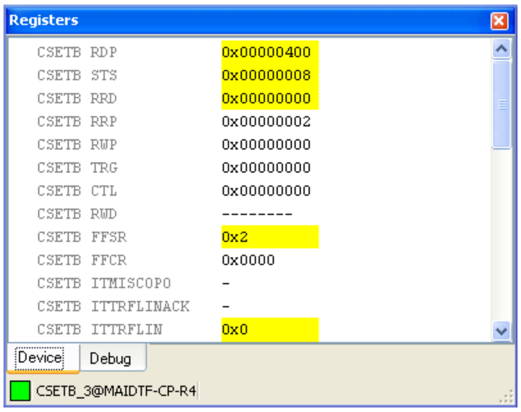
   The RealView ICE host software is included with RVDS, and does not require a separate installation.

3. Select **Start** → **Programs** → **ARM** → **RealView Development Suite v4.1** → **RealView Debugger v4.1** to start RealView Debugger.
4. Configure RealView ICE to identify the processors and CoreSight components on your development platform. You must also configure RealView ICE to access the CoreSight ETB.
5. Select **Registers** from the **View** menu to display the Registers view.
6. If your development platform has multiple trace sources, then you must configure the CoreSight Trace Funnel to identify the source from which trace is to be captured:
   a. Connect to the `CSTFunnel_`*`n`* target.
   b. Set up the Trace Funnel registers.

   Note

   If your development platform contains multiple processors and multiple ETMs, only one CoreSight ETM at a time can be chosen as the trace source in this release.

7. Configure the CoreSight ETB registers:
   a. Connect to the `CSETB_`*`n`* target.
   b. Set up the CSETB registers. Figure B-5 shows an example:



```
CSETB RDP          0x00000400
CSETB STS          0x00000008
CSETB RRD          0x00000000
CSETB RRP          0x00000002
CSETB RWP          0x00000000
CSETB TRG          0x00000000
CSETB CTL          0x00000000
CSETB RWD          --------
CSETB FFSR         0x2
CSETB FFCR         0x0000
CSETB ITMISCOP0    -
CSETB ITTRFLINACK  -
CSETB ITTRFLIN     0x0
```

CSETB_3@MAIDTF-CP-R4

Figure B-5 CSETB registers in the Registers view

8. Connect to the target processor on your development platform.

9. Connect to the TPA.

See also:

- *RealView ICE and RealView Trace Setting Up the Hardware*
- *RealView ICE Using the Debug Hardware Configuration Utilities*
- *Connecting to an analyzer* on page 2-17
- *Configuring DSTREAM or RealView ICE to access the CoreSight ETB*
- Chapter 4 *Configuring the ETM*
- the following in the *RealView Debugger User Guide*:
  - — Creating a Debug Configuration on page 3-10
  - — Connecting to a target on page 3-19.

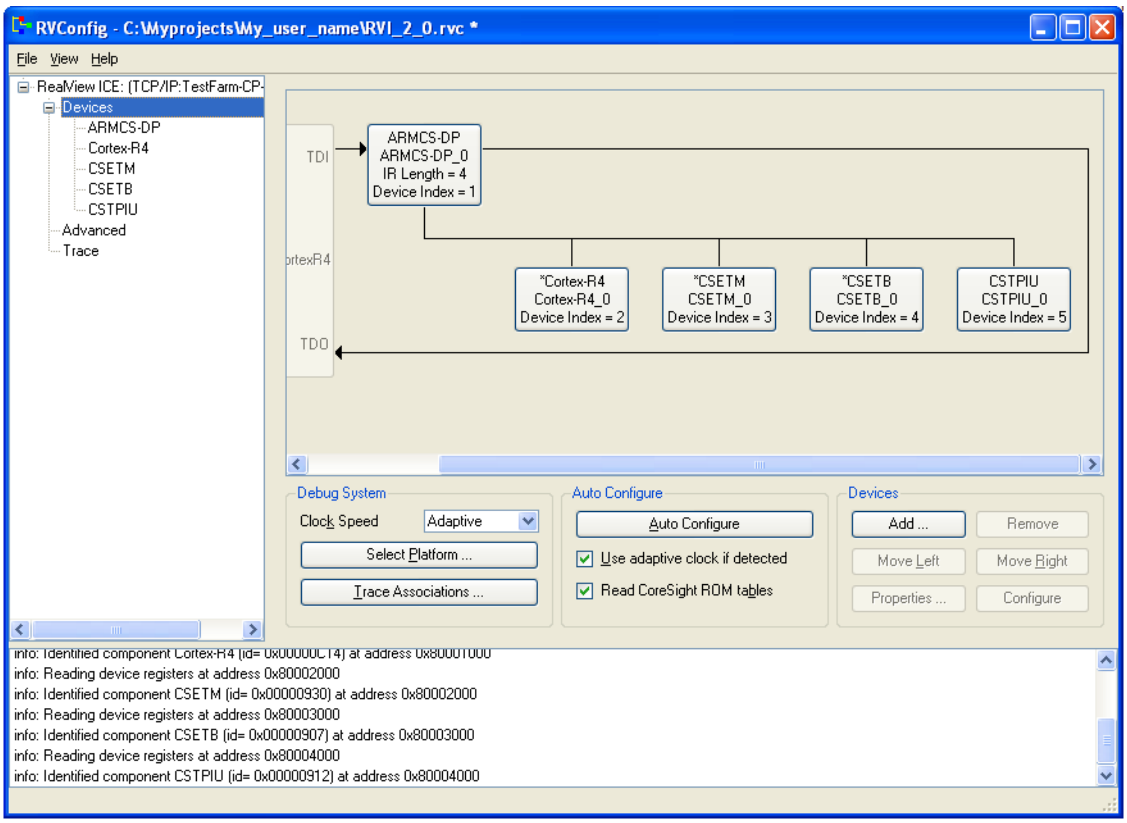
B.4.1 Configuring DSTREAM or RealView ICE to access the CoreSight ETB

To configure DSTREAM or RealView ICE to access the CoreSight ETB:

1. In the Code window, select **Connect to Target...** from the **Target** menu to display the Connect to Target window.

2. Right-click on the your Debug Configuration to display the context menu. For example, right-click on `RealView ICE`.

3. Select **Configure...** from the context menu to display the RVConfig utility. Figure B-6 shows an example:



```
info: Identified component Cortex-R4 (id= 0x00000C14) at address 0x80001000
info: Reading device registers at address 0x80002000
info: Identified component CSETM (id= 0x00000930) at address 0x80002000
info: Reading device registers at address 0x80003000
info: Identified component CSETB (id= 0x00000907) at address 0x80003000
info: Reading device registers at address 0x80004000
info: Identified component CSTPIU (id= 0x00000912) at address 0x80004000
```

Figure B-6 RVConfig utility

4. Right-click on the box in the scan chain schematic diagram corresponding to the processor that you are tracing.

5. Select **Properties...** to display the Device Properties dialog box. Figure B-7 shows an example:

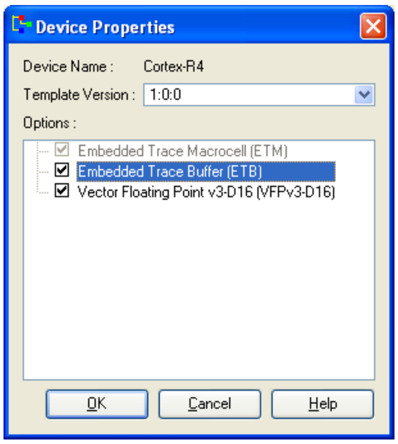

Device Properties
Device Name : Cortex-R4
Template Version : 1:0:0
Options :
☑ Embedded Trace Macrocell (ETM)
☑ Embedded Trace Buffer (ETB)
☑ Vector Floating Point v3-D16 (VFPv3-D16)
OK
Cancel
Help

Figure B-7 Device Properties dialog box

6. Select **Embedded Trace Buffer (ETB)** from the list.

7. Click **OK** to close the Device Properties dialog box.

8. Select **Save** from the RVConfig **File** menu.

9. Select **Exit** to close the RVConfig utility.

10. Connect to the target in the usual way.

See also

- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*
- *Connecting to an analyzer* on page 2-17
- *Configuring DSTREAM or RealView ICE to access the CoreSight ETB* on page B-8
- Chapter 4 *Configuring the ETM*
- the following in the *RealView Debugger User Guide*:
  - Creating a Debug Configuration on page 3-10
  - Connecting to a target on page 3-19.

Appendix C
Status Messages in Captured Trace

This appendix lists the error and warning messages that might appear in captured trace. It includes:

- *Error messages* on page C-2
- *Warning messages* on page C-4.

C.1 Error messages

The following error messages might be displayed in the captured trace:

Error: Synchronization Lost

Indicates that the trace data cannot be decoded because:

- RealView® Debugger has detected trace data that does not correspond to the image loaded into the debugger
- the trace data cannot be decompressed, for example, if the trace data is corrupt.

Error: ETM FIFO Overflow

Indicates that tracing was temporarily suspended because the *Embedded Trace Macrocell*™ (ETM™) FIFO buffer became full. When this occurs, there is a discontinuity of returned trace information.

See also *FIFO overflow protection* on page 4-8.

Error: Coprocessor data transfer of unknown size

When tracing data, RealView Debugger executed an unrecognized coprocessor memory access instruction, and the decompressor could not deduce the amount of data transferred by the instruction. Decompression of data tracing, and data address tracing, stop until appropriate synchronization points are found in the trace data.

Error: Data synchronization lost following FIFO overflow

Some versions of the ARM ETM can cause corrupt data trace after a FIFO overflow has occurred. If the decompressor sees a case where this is likely to have happened, it outputs this message, and suppresses data and data address tracing until it can resynchronize.

See also *FIFO overflow protection* on page 4-8.

Error: Trace branch address does not match instruction's branch address

RealView Debugger has branch addresses from both the trace and from the image loaded into the debugger, and these addresses do not match. This error can be detected only if you are using an XScale target. The source of the error is probably an incorrect image.

Error: Unexpected exception

The instruction has marked an exception, but the exception address does not appear to be a valid exception address.

Error: Instruction not known

The decompressor was not in sync for this instruction, but later discovered that this instruction was an exception.

Error: Incorrect synchronization address

An address broadcast for synchronization did not match that being maintained by the decompressor.

Error: Instruction data overflowed end of buffer

The data for the instruction is not in the buffer. This can occur when trace capture has stopped because it filled the buffer between the instruction being traced and its data being traced. All available data addresses and data are traced.

Error: The next instruction was traced as a branch

The instruction on the next line is not a branch, but the ETM traced it as a branch. This usually indicates that the wrong image is loaded into the debugger, or the code is self-modifying.

Error: The next instruction was not traced as an indirect branch

The instruction on the next line is an indirect branch, but the ETM did not trace it as an indirect branch. This usually indicates that the wrong image is loaded into the debugger, or the code is self-modifying.

Error: The next instruction was traced as a memory access instruction

The trace from the ETM indicated that the instruction on the next line read some data from memory, or wrote some data to memory, but the instruction is not a memory access instruction. This usually indicates that the wrong image is loaded into the debugger, or the code is self-modifying. Decompression of data tracing and data address tracing stops until appropriate synchronization points are found in the trace data.

Error: The next instruction should have been executed unconditionally

The trace from the ETM indicated that the instruction on the next line failed its condition code test, so was not executed, but the instruction is one that must be executed unconditionally. This usually indicates that the wrong image is loaded into the debugger, or the code is self-modifying.

Error: Corrupt address in trace data

The trace data contains an impossible address. This only occurs as a result of a hardware problem (such as a faulty connector).

Error: The next instruction was not traced as a branch

The current instruction is a branch but the trace does not indicate this. This usually indicates that the wrong image is loaded into the debugger, or the code is self-modifying. Decompression of data tracing and data address tracing stop until appropriate synchronization points are found in the trace data.

Error: The next instruction could not be read

The memory containing the traced instruction could not be read. This might occur if the application image attempts to execute a region of unreadable memory, in which case the instruction aborts with a Prefetch Abort. It might also occur if trace is decoded while the image is still running, and the image attempts to execute code outside the loaded image.

C.2 Warning messages

The following warning messages might be displayed in the captured trace:

Warning: Debug State

Indicates that tracing was suspended for several processor cycles because the processor entered debug state.

Warning: Trace Pause

Indicates that tracing was temporarily suspended because of the trace conditions that have been set. `Trace Pause` represents the period of execution between the areas you have defined to be traced.

Warning: Instruction address synchronization has been restored

This message occurs after a problem in which instruction address synchronization has been lost. It indicates that the decompressor has found a point at which it can resume decompressing instruction addresses.

Warning: Unable to trace Jazelle state, trace data ignored

The ETM detected the processor entering Jazelle® (bytecode) state. The decompressor is unable to decompress Jazelle bytecode execution, so all trace output is suppressed until the processor leaves Jazelle state.

Warning: Data address synchronization restored

This message occurs after a problem in which data address synchronization has been lost. It indicates that the decompressor has found a point at which it can resume decompressing data addresses.

Warning: No data in trace buffer

The trace buffer is composed entirely of zero. This warning is very rare, and only occurs if you are using an XScale target.

Warning: Data synchronization restored

This message occurs after a problem in which data synchronization has been lost. It indicates that the decompressor has found a point at which it can resume decompressing data.

Warning: Too many checkpoints in XScale trace buffer

Indicates that more than two checkpointed entries were found in the buffer. The decompressor has attempted to use the most recent entries. This message only occurs when you are using an XScale target.

Warning: Memory address unknown, insufficient trace data

This warning only occurs near the beginning of the decoded trace when the trace buffer (not the FIFO) of the TPA has overflowed. It means that there has not yet been a complete memory access address in the trace data, and therefore the trace decoder cannot calculate the address of a data access. The ETM outputs a complete address on the first data access traced, and repeats this every 1024 cycles after this, if there are data accesses to be traced. To reduce buffer usage, other memory addresses are output relative to the last full memory address. If the buffer overflows, and the complete address is lost, the decoder cannot calculate data addresses that occur before the next full data address is emitted.

Warning: Data suppression protected ETM FIFO from overflow

This warning occurs when you have enabled data suppression, and the ETM has to suppress the output of data in an attempt to prevent an ETM FIFO overflow. This is usually followed by the `Warning: Data synchronization restored` status line when data trace is restored.

See also *FIFO overflow protection* on page 4-8.