ARM® Profiler

Version 2.1.1

User Guide

Copyright © 2007- 2010 ARM Limited. All rights reserved.

ARM DUI 0414E

ARM Profiler
User Guide

Copyright © 2007- 2010 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
October 2007	A	Non-Confidential	Release Version 1.0
March 2008	B	Non-Confidential	Release Version 1.1
September 2008	C	Non-Confidential	Release Version 2.1.1
May 2009	D	Non-Confidential	Release Version 2.1
May 2010	E	Non-Confidential	Release Version 2.1.1

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

XVID Notice

THIS NOTICE IS FOR THE USE OF XVID. ARM IS ONLY DELIVERING XVID TO YOU FOR CONVENIENCE ON CONDITION THAT YOU ACCEPT THAT IT IS NOT LICENSED TO YOU BY ARM BUT THAT IT IS SUBJECT TO THE TERMS OF THE GNU GENERAL PUBLIC LICENSE VERSION 2 AND MAY BE SUBJECT TO OTHER PROPRIETARY LICENCES. YOU EXPRESSLY ASSUME ALL LIABILITIES AND RISKS WITH RESPECT TO YOUR USE AND DISTRIBUTION OF XVID.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

`http://www.arm.com`

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Contents
ARM Profiler User Guide

Preface

About this book x
Feedback xiv

Chapter 1 Introduction

1.1 About the ARM Profiler 1-2
1.2 Availability and compatibility 1-3
1.3 Installation 1-4
1.4 Licensing 1-5
1.5 Getting help 1-6

Chapter 2 Getting Started with the ARM Profiler

2.1 How to use this tutorial 2-2
2.2 Opening the ARM Workbench 2-3
2.3 Importing the xvid example 2-5
2.4 Re-building the project 2-9
2.5 Profiling using a Real-Time System Model 2-10
2.6 Stopping the execution using the live update view 2-11
2.7 Examining the new analysis file 2-12

Chapter 3 Data Collection Using RealView Trace 2

3.1 Required items 3-2

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.2 Opening hardware preferences within the ARM Workbench 3-3
3.3 Setting the connection options 3-4
3.4 Setting the image options 3-10
3.5 Setting profiling options 3-12
3.6 Exporting a launch script 3-16
3.7 Executing the run configuration 3-17
3.8 Hardware profiling restrictions 3-18
3.9 Hardware profiling execution speed 3-19

Chapter 4 Data Collection Using a Real-Time System Model

4.1 Overview 4-2
4.2 Using the ARM compilation tools in the ARM Workbench 4-3
4.3 Creating a profiling-enabled RTSM run configuration 4-4
4.4 Setting the connection options 4-5
4.5 Setting the image options 4-7
4.6 Setting the profiling options 4-8
4.7 Running the configuration 4-10
4.8 Enabling profiling outside of the ARM Workbench 4-11

Chapter 5 The Analysis Summary

5.1 Analysis summary overview 5-2
5.2 Opening an analysis summary 5-3
5.3 Analysis summary elements 5-4
5.4 Live update 5-13

Chapter 6 The Table Reports: Functions, Files, Classes, and Call Chains

6.1 Table report basics 6-2
6.2 Navigating to other reports 6-13
6.3 The functions report 6-17
6.4 The classes report 6-18
6.5 The files report 6-19
6.6 The call chains report 6-20

Chapter 7 The Code and Replay Views

7.1 Overview 7-2
7.2 Navigating to the code view 7-3
7.3 Basic code view functionality 7-5
7.4 The source panel 7-7
7.5 The disassembly panel 7-11
7.6 The replay view 7-15

Chapter 8 The Call Graph

8.1 Overview 8-2
8.2 Opening a call graph 8-3
8.3 Call graph layout 8-4
8.4 The mini-map 8-6

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.5 Color coding 8-7
8.6 Selection behavior 8-8
8.7 Contextual menu options 8-9
8.8 The toolbar 8-12
8.9 The outline view 8-14

Chapter 9 The Call Summary

9.1 Call summary breakdown 9-2
9.2 Function box statistics 9-3
9.3 Filtering instances 9-6
9.4 Call summary navigation 9-8
9.5 Navigating to other report types 9-9

Chapter 10 Merging Analysis Files

10.1 Reasons to merge analysis files 10-2
10.2 Analysis file compatibility 10-3
10.3 How to merge analysis files 10-4

Chapter 11 Preferences

11.1 Accessing the ARM Profiler color preferences 11-2
11.2 Color preference descriptions 11-3

Chapter 12 Profiling Applications Running on the Symbian OS

12.1 Building the ARM Profiler Symbian OS kernel extension 12-2
12.2 Building Symbian OS applications 12-4
12.3 Profiling your Symbian OS application 12-5

Chapter 13 Profiling Applications Running on Linux OS

13.1 Installing and patching the Linux kernel extension 13-2
13.2 Profiling your Linux application 13-4

Appendix A Using the Command Line

A.1 Configuring your system for running the ARM Profiler on the command line ... A-2
A.2 Command line options A-3

Appendix B Keyboard shortcuts

B.1 Table report keyboard shortcuts B-2
B.2 Code view keyboard shortcuts B-3
B.3 Call graph keyboard shortcuts B-5
B.4 Call summary keyboard shortcuts B-6

Appendix C Troubleshooting guide

C.1 Troubleshooting steps C-2

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Preface

This preface introduces the *ARM® Profiler User Guide*. It contains the following sections:

- *About this book* on page x
- *Feedback* on page xiv.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

About this book

This book describes the functionality of the ARM Profiler, providing information about its use and a detailed reference on every data type contained in an analysis report generated by the ARM Profiler.

Intended audience

This manual is written for developers who intend to use the ARM Profiler to profile code running on an ARM target. To get the most out of the toolset and this documentation, you must have a working knowledge of the ARM compiler and know how to use it to generate executable files. If you wish to profile your application on hardware, you will also need both a RealView® ICE debug unit and RealView Trace 2 capture unit.

Using this book

This book contains the following chapters:

Chapter 1 ***Introduction***

Chapter one gives you a high level overview of the features in the ARM Profiler and provides information about additional resources.

Chapter 2 ***Getting Started with the ARM Profiler***

Chapter two is a guide that steps you through the process of creating an analysis file using the xvid example.

Chapter 3 ***Data Collection Using RealView Trace 2***

Chapter three describes the process of gathering profiling data using the RealView ICE and RealView Trace 2 hardware.

Chapter 4 ***Data Collection Using a Real-Time System Model***

Chapter four illustrates how to gather analysis data using a Real-Time System Model created using RealView System Generator.

Chapter 5 ***The Analysis Summary***

Chapter five outlines the ARM Profiler reporting interface, describes the Summary Report in detail, and shows you how to navigate all of the various report types.

Chapter 6 ***The Table Reports: Functions, Files, Classes, and Call Chains***

Chapter six focuses on the various table reports of the ARM Profiler. It describes each of the report fields and how to sort and manage the data.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 7 ***The Code and Replay Views***

Chapter seven provides a complete look at the code and replay views. the code view shows detailed information about the line-by-line execution and performance of a single function. The replay view provides the full program trace if that option is enabled.

Chapter 8 ***The Call Graph***

Chapter eight describes the functionality of the call graph and how you can use it to pinpoint bottlenecks.

Chapter 9 ***The Call Summary***

Chapter nine provides a detailed description of how to use the call summary to explore your code.

Chapter 10 ***Merging Analysis Files***

Chapter ten shows you how to merge analysis reports into a single file.

Chapter 11 ***Preferences***

Chapter eleven details each of the preferences in the ARM Profiler and how to set them.

Chapter 12 ***Profiling Applications Running on the Symbian OS***

Chapter twelve shows you how to setup Symbian for profiling and the steps to profile a Symbian application.

Appendix A ***Using the Command Line***

Reference this Appendix for descriptions of each of the available command line options.

Appendix B ***Keyboard shortcuts***

Appendix B details all of the keyboard shortcuts in each of the ARM Profiler report types.

Appendix C ***Troubleshooting guide***

Appendix C lists the most common causes for hardware profiling failure.

This book assumes that the ARM software is installed in the default location. For example, on Windows this might be `volume:\Program Files\`. This is assumed to be the location of `install_directory` when referring to path names. For example `install_directory\ARM Profiler v2.1\Examples\...`. You might have to change this if you have installed your ARM software in a different location.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Typographical conventions

The following typographical conventions are used in this book:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes ARM processor signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`monospace 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.

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information on developing code for the ARM family of processors.

See `http://infocenter.arm.com/help/index.jsp` for access to ARM documentation.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets, addenda, and the ARM Frequently Asked Questions (FAQs).

ARM publications

See the following publications for detailed documentation on various components of the RealView Development Suite:

- *RealView® Development Suite Getting Started Guide* (ARM DUI 0255)
- *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain* (ARM DUI 0529)
- *ARM® Compiler toolchain Developing Software for ARM® Processors* (ARM DUI 0471)
- *ARM® Compiler toolchain Building Linux Applications with ARM® Compiler toolchain and GNU Libraries* (ARM DUI 0483)
- *ARM® Compiler toolchain Using the Compiler* (ARM DUI 0472)

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- *ARM® Compiler toolchain Using ARM® C and C++ Libraries and Floating-Point Support* (ARM DUI 0475)
- *ARM® Compiler toolchain Using the Assembler* (ARM DUI 0473)
- *ARM® Compiler toolchain Using the Linker* (ARM DUI 0474)
- *ARM® Compiler toolchain Compiler Reference* (ARM DUI 0491)
- *ARM® Compiler toolchain Assembler Reference* (ARM DUI 0489)
- *ARM® Compiler toolchain Linker Reference* (ARM DUI 0493)
- *ARM® Compiler toolchain ARM® C and C++ Libraries and Floating-Point Support Reference* (ARM DUI 0492)
- *ARM® Compiler toolchain Creating Static Software Libraries with armar* (ARM DUI 0476)
- *ARM® Compiler toolchain Using the fromelf Image Converter* (ARM DUI 0477)
- *ARM® Compiler toolchain Errors and Warnings Reference* (ARM DUI 0496)
- *ARM® Compiler toolchain Migration and Compatibility* (ARM DUI 0530)
- *RealView® Development Suite Real-Time System Model User Guide* (ARM DUI 0424)
- *RealView® Debugger Essentials Guide* (ARM DUI 0181)
- *RealView® Debugger User Guide* (ARM DUI 0153)
- *RealView® Debugger Target Configuration Guide* (ARM DUI 0182)
- *RealView® Debugger Trace User Guide* (ARM DUI 0322)
- *RealView® Debugger RTOS Guide* (ARM DUI 0323)
- *RealView® Debugger Command Line Reference Guide* (ARM DUI 0175).

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Feedback

ARM Limited welcomes feedback on both the ARM Profiler and its documentation.

Feedback on the ARM Profiler

If you have any problems with the ARM Profiler, contact your supplier. To help them provide a rapid and useful response, give:

- your name and company
- the serial number of the product
- details of the release you are using
- details of the platform you are running on, such as the hardware platform, operating system type and version
- a small standalone sample of code that reproduces the problem
- 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
- the version string of the tools, including the version number and build numbers.

Feedback on this book

If you notice any errors or omissions in this book, send an e-mail to `errata@arm.com` giving:

- the document title
- the number, ARM DUI 0414E
- the page number, or page numbers, to which your comments apply
- a concise explanation of the problem.

General suggestions for additions and improvements are also welcome.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 1
Introduction

This chapter introduces the ARM Profiler, discusses compatibility, and describes each of the primary components.

It contains the following sections:

- *About the ARM Profiler* on page 1-2
- *Availability and compatibility* on page 1-3
- *Installation* on page 1-4
- *Licensing* on page 1-5
- *Getting help* on page 1-6.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

1.1 About the ARM Profiler

The ARM Profiler enables you to see how your code performs on a target system, either by observing your code on target hardware using RealView ICE and RealView Trace 2 or by testing code against a *Real-Time System Model* (RTSM). After the ARM Profiler finishes observing your code in action, it produces an analysis file that provides fine grain feedback:

- by highlighting bottlenecks in your code
- by enabling you to test against large, real-world data samples in a short amount of time
- by providing data in tabular format overlaid against your C source and disassembly code.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

1.2 Availability and compatibility

Every feature of the toolset is integrated into the ARM Workbench, which is installed with the ARM Profiler. Use the customized perspectives and views of the ARM Profiler within the ARM Workbench to set up hardware preferences, run simulations to produce profile reports and analyze data.

——— **Note** ———

The ARM Profiler 2.1 requires a computer with a 2GHz dual-core processor and 1GB of memory or better.

———————————

Currently, the ARM Profiler works on the following systems:

- Windows XP Professional (32-bit or 64-bit) with service pack 2 installed
- Windows Vista Business (32-bit or 64-bit) with service pack 1 installed
- Windows Vista Enterprise (32-bit or 64-bit) with service pack 1 installed
- Red Hat Enterprise Linux WS version 4 for Intel x86 (32-bit or 64-bit) using GNOME Window Manager and Bash Shell
- Red Hat Enterprise Linux WS version 5 for Intel x86 (32-bit or 64-bit) using GNOME Window Manager and Bash Shell

——— **Note** ———

The ARM Profiler is not compatible with the DSTREAM hardware at this time.

———————————

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

1.3 Installation

The ARM Profiler is installed as part of the RealView Development Suite 4.1 Professional.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

1.4 Licensing

All licensing for the ARM Profiler is controlled by the FLEXnet license management system. Use the FLEXnet server software to track and control your ARM Profiler licenses. You can request licenses using the ARM Web Licensing page at `http://license.arm.com`. See the FLEXnet for *ARM Tools License Management Guide* (DUI 0207) for more information.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

1.5 Getting help

The ARM Profiler provides documentation and examples for you to familiarize yourself with the profiler.

To access the documentation in HTML format:

1. Select **Help** → **Help Contents** from the main menu.
2. From the **Contents** frame, select **ARM Profiler User Guide**.

To access the dynamic help you must:

1. Open the report type that interests you.
2. Click on the question mark icon or press F1 on your keyboard.

   Note

   On Red Hat Linux, press **Shift + F1** to get help.

This opens the help view with a list of topics that relate to the currently open report type or run configuration panel. Clicking on any of these links opens the corresponding help topic within the help view, enabling you to browse help topics without leaving the ARM Workbench.

To access the examples, select **All Programs** → **ARM** → **ARM Profiler v2.1** → **Examples** from the Windows **Start** menu.

On Red Hat Linux, select **Applications** → **ARM** → **ARM Profiler v2.1** → **Examples**.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 2
Getting Started with the ARM Profiler

This chapter guides you through the process of creating and exploring an analysis file using the xvid source code from the examples directory within the ARM Profiler installation.

It contains the following sections:

- *How to use this tutorial* on page 2-2
- *Opening the ARM Workbench* on page 2-3
- *Importing the xvid example* on page 2-5
- *Re-building the project* on page 2-9
- *Profiling using a Real-Time System Model* on page 2-10
- *Stopping the execution using the live update view* on page 2-11
- *Examining the new analysis file* on page 2-12.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

2.1 How to use this tutorial

This tutorial guides you through the basic steps needed to profile an application using the ARM Profiler. The source code for xvid, a video encoder and decoder, is installed in the following directory:

```
install_directory\Profiler\Contents\2.1\BuildNumber\examples
```


Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

2.2 Opening the ARM Workbench

To begin the tutorial and get started using the ARM Profiler, open the ARM Workbench by selecting:

Start → All Programs → ARM → ARM Workbench IDE v4.1

The ARM Workbench prompts you to either select an existing workspace directory or create a new one. Create a workspace if you have not done so already. For more information on workspaces, see the ARM Workbench IDE User Guide.

If this is the first time you have used this workspace directory, the ARM Workbench welcome window appears. This is pictured in Figure 2-1.

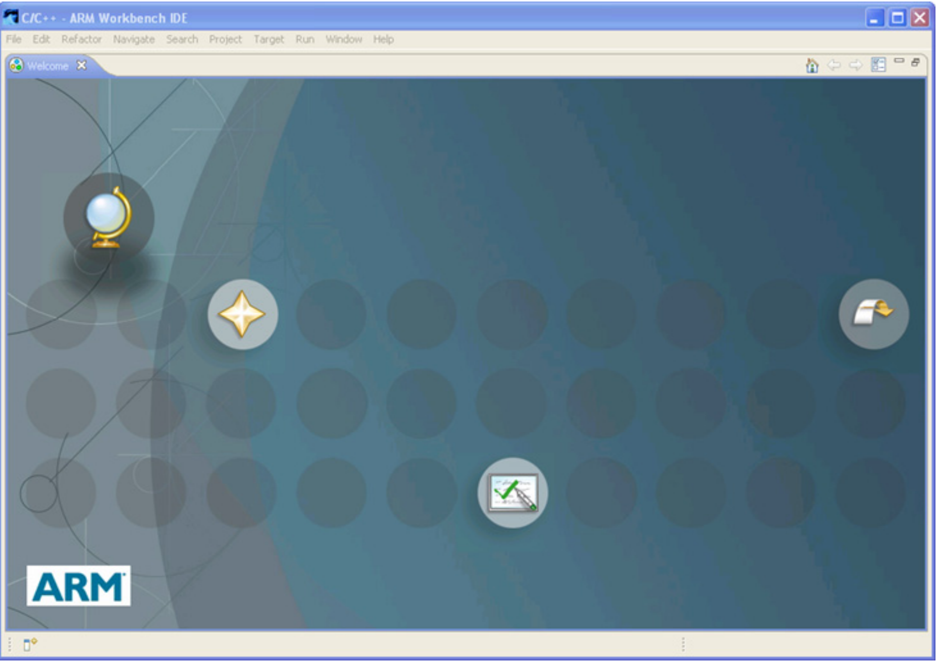


Figure 2-1 Welcome window

Click on the curved arrow to exit the welcome window and open the ARM Workbench.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

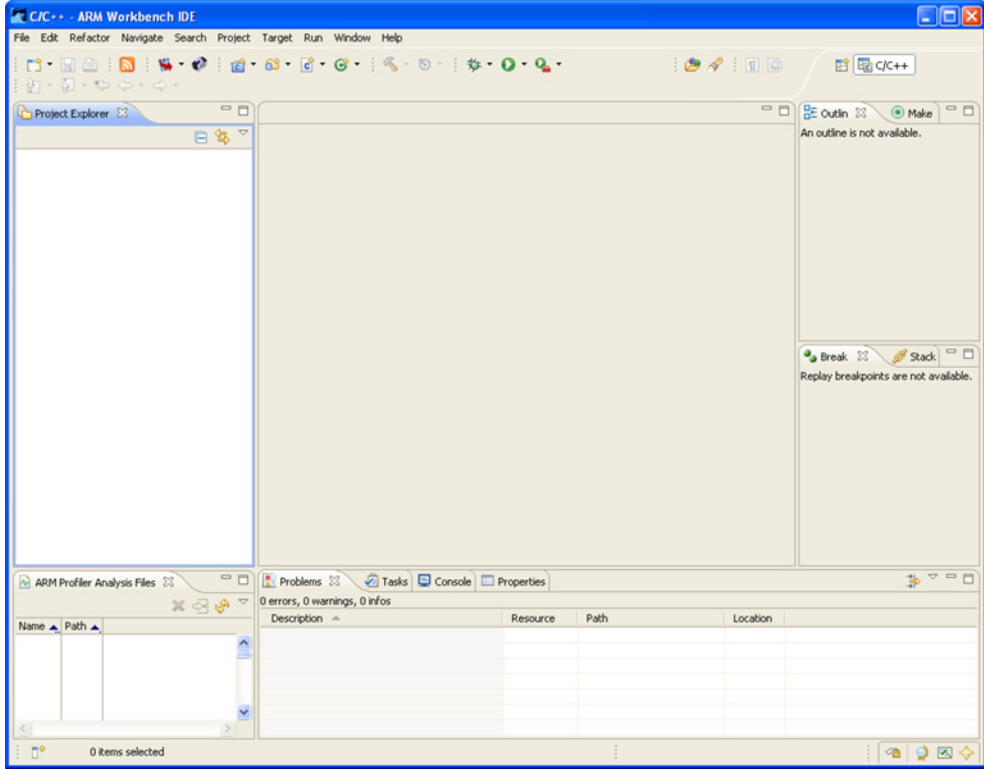


Figure 2-2 The ARM Workbench

Your workbench might vary from the one shown in Figure 2-2 if you have already worked in the ARM Workbench and modified the layout.

The workbench is a highly versatile window that contains a set of Views. All of these Views can be re-sized, moved, or removed from the workbench. Make sure the Project Explorer view is open before continuing the tutorial. By default, the Project Explorer view is the long area on the left side of the workbench.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

2.3 Importing the xvid example

After you have successfully opened the ARM Workbench, follow these steps to import the xvid source code:

1. Select **File** → **Import...**. This opens an **Import** dialog like the one pictured in Figure 2-3.

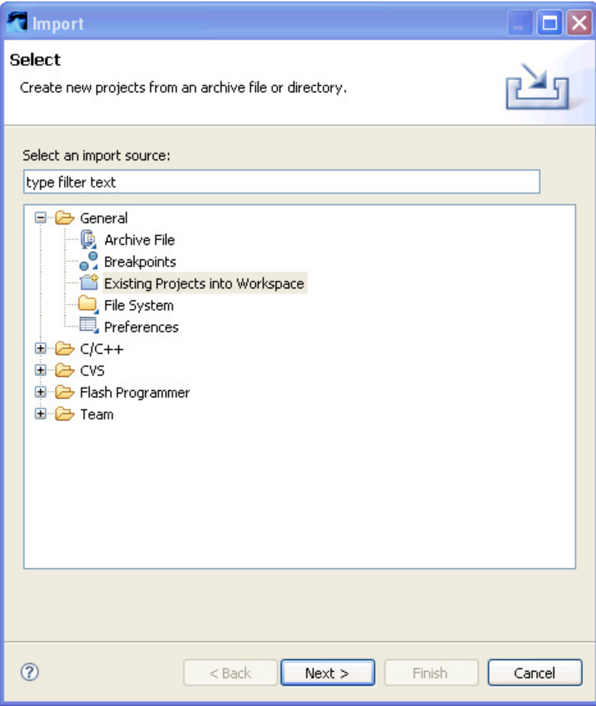


Figure 2-3 The Import Dialog

2. Open the **General** tab and select Existing Projects into Workspace as pictured in *The Import Dialog*.

3. Click the **Next** button to open a window like the one pictured in Figure 2-4 on page 2-6.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

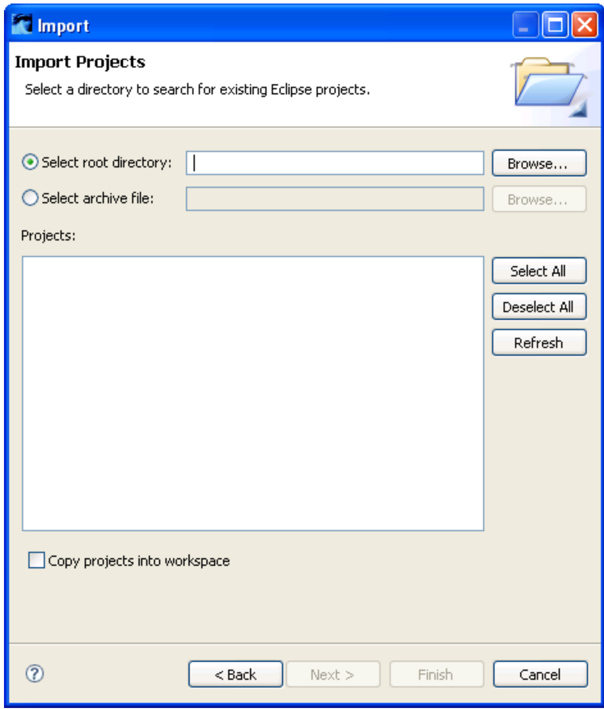


Figure 2-4 The import project dialog

4. Click the **Browse...** button to open a dialog that enables you to search the directories on your disk for the desired project files.

5. Use the windows to navigate to the `install_directory\Profiler\Contents\2.1\BuildNumber\examples` directory, as pictured in Figure 2-5 on page 2-7.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

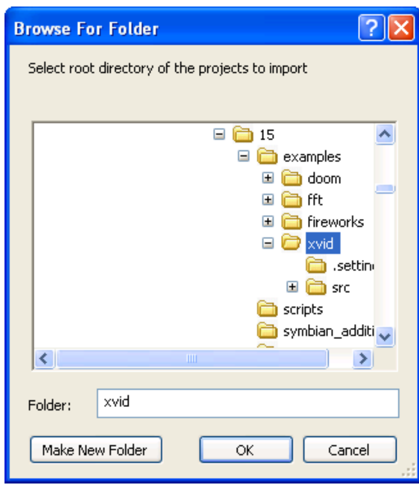


Figure 2-5 The Browse For Folder dialog

6. Select the xvid folder in the hierarchy and press **OK**. This closes the window and returns you to an updated **Import** dialog, as pictured in Figure 2-6.

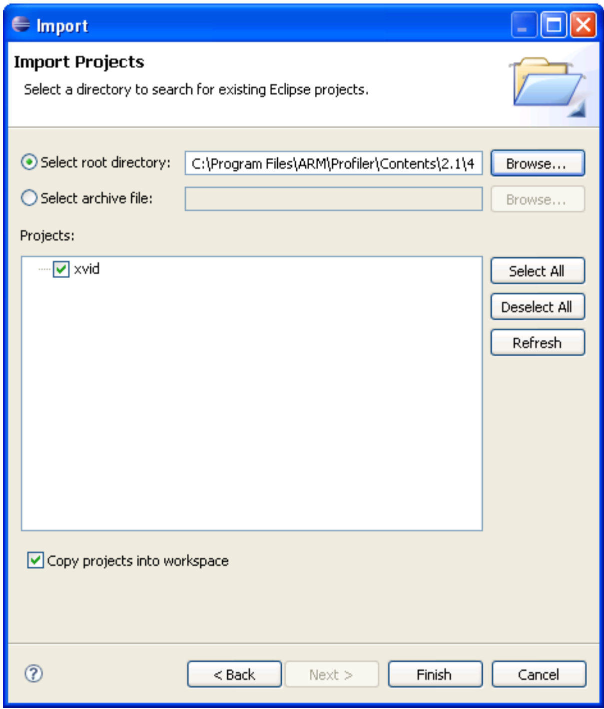


Figure 2-6 The updated Import dialog

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7. Make sure that the checkbox next to xvid is selected.
8. Make sure that the **Copy projects into workspace** checkbox is selected.
9. Click the **Finish** button to finish the import of the xvid project.
10. Import the ARM project located in the same directory location as the xvid project. Repeat steps 1 through 9, only select the ARM folder in step 6.

——— **Note** ———

The ARM project contains the `common.make` file, necessary to build all of the ARM Profiler examples.

————————

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

2.4 Re-building the project

Once you have imported the xvid project, select **Project** → **Build Project** to re-build the project.

The **Console** view keeps you updated on the progress, as pictured in Figure 2-7.

Figure 2-7 The Console view

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

2.5 Profiling using a Real-Time System Model

After you have imported xvid as a project and re-built the project, you are ready to profile the code on a Real-Time System Model. To do this, follow these steps:

1. Select **Run** → **Open Run Dialog...** from the menu.
2. Select ARM Target from the configuration list on the left hand side of the window.
3. Press the new launch configuration button just above it. The launch configuration window is pictured in Figure 2-8.

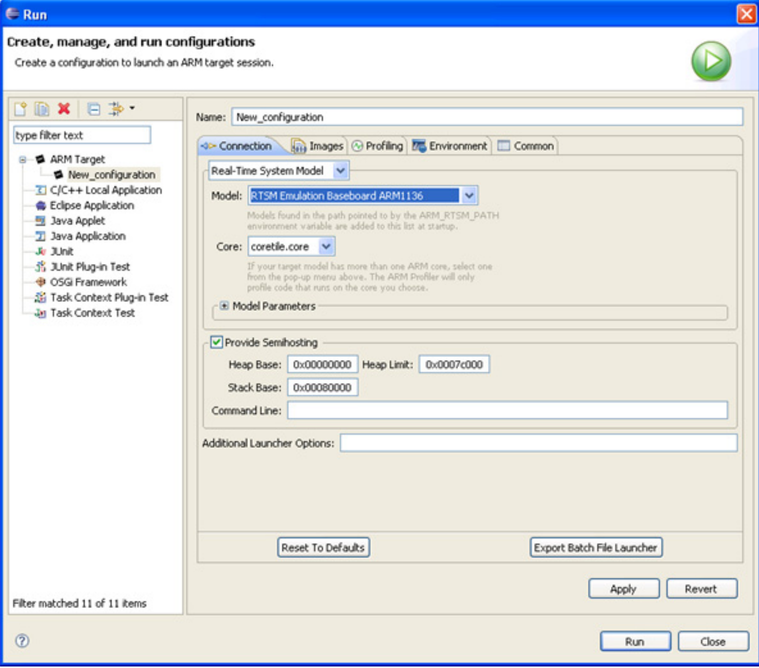


Figure 2-8 Running on the ARM 1136 EB RTSM

4. Select **RTSM Emulation Baseboard ARM1136** from the **Model** drop-down menu.
5. Press the **Run** button at the bottom of the launch configuration window.

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

2.6 Stopping the execution using the live update view

As soon as you press the run button, the sample video begins and the ARM Profiler opens the **Live Update** view, which provides a live look at the accumulating data as the xvid example executes.

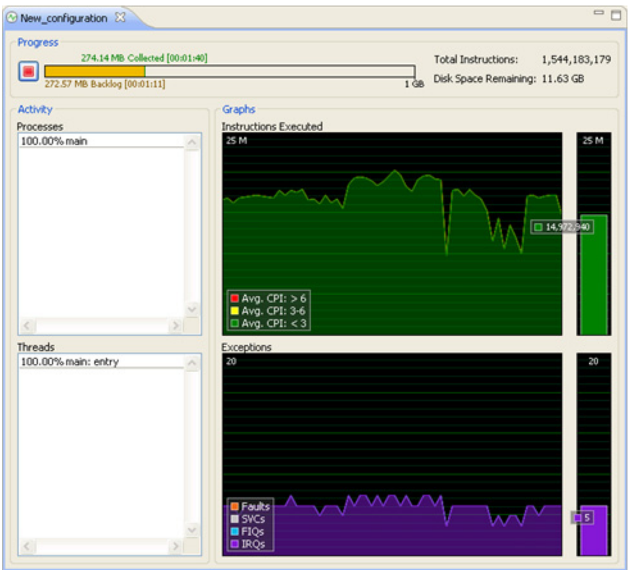


Figure 2-9 Live update

Let xvid run for a while, then click the stop button, located to the left of the progress bar. This terminates program execution and triggers the creation of an analysis directory titled `xvid_001.apd` and places it in the location you specified in the **Analysis Output** field in the **Profiling** tab of the Run configuration dialog box. By default, this is the `ARM Profiler` directory located in `My Documents`.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

2.7 Examining the new analysis file

The summary report of the newly created `xvid_001.apd` opens automatically. The summary report gives you a top level look at the performance of the xvid application with a variety of charts, graphs, and summary-level information.

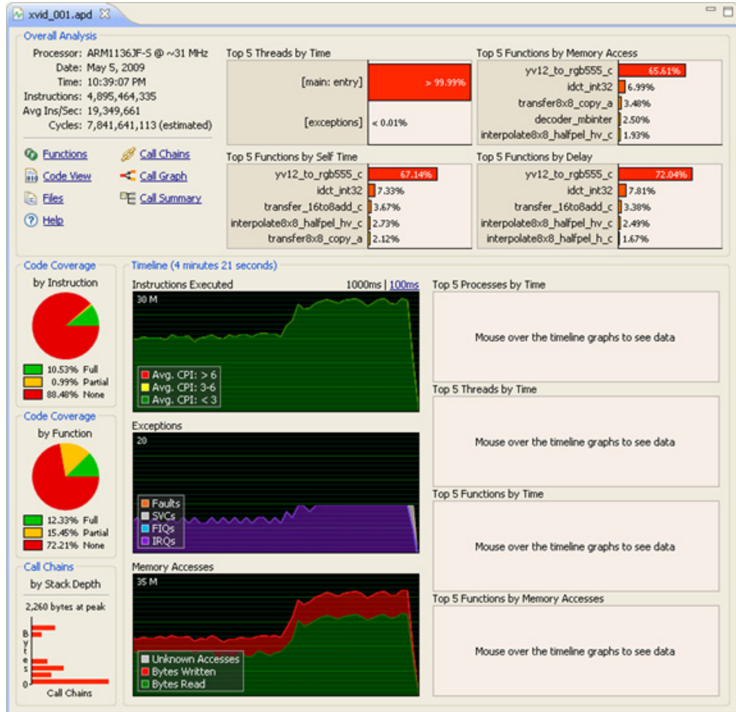


Figure 2-10 The summary report

The navigation section, located in the top left hand corner of the summary report provides links to the more detailed ARM Profiler report types. Clicking on one of the links opens the given report as a new tab in the editor section of the ARM Workbench. You can easily switch between report types by clicking on the labeled tabs.

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 3
Data Collection Using RealView Trace 2

This chapter describes how to set up your target hardware for profiling using RealView ICE and RealView Trace 2 and guides you through the process of creating a run configuration within the ARM Workbench.

It contains the following sections:

- *Required items* on page 3-2
- *Opening hardware preferences within the ARM Workbench* on page 3-3
- *Setting the connection options* on page 3-4
- *Setting the image options* on page 3-10
- *Setting profiling options* on page 3-12
- *Exporting a launch script* on page 3-16
- *Executing the run configuration* on page 3-17
- *Hardware profiling restrictions* on page 3-18
- *Hardware profiling execution speed* on page 3-19

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.1 Required items

To perform hardware profiling, you must have the following:

- The RealView ICE run control unit and accessories

  Note

  See the *RealView ICE Setting Up the Hardware* (ARM DUI 0515) for a complete list of all RealView ICE and RealView Trace 2 accessories.

- The RealView Trace 2 data capture unit and accessories
- An ARM based processor with an embedded trace macrocell and an external trace port
- Joint Test Action Group (JTAG) IEEE Standard 1149.1-2001 port on the target hardware
- At least one unused USB 2.0 port on the host machine

  The RealView Trace 2 unit requires one dedicated USB 2.0 port on your host machine and the RealView ICE requires either a second USB 2.0 port or an ethernet connection to your host. Connecting both to a USB hub will not work.

- ARM Software

  The ARM Profiler, the RealView ICE host software version 3.4 or greater, and the USB drivers must be installed on your host machine. You must also install the corresponding version of the firmware on your RealView ICE unit. For more details, see *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities* (ARM DUI 0498).

Note

The ARM Profiler is not compatible with the DSTREAM hardware at this time.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.2 Opening hardware preferences within the ARM Workbench

When you have successfully set up the RealView ICE and RealView Trace 2 hardware and you have created an image file, you are ready to set the parameters of your hardware profiling run. To open the run configuration window, follow these steps:

1. Select **Start** → **All Programs** → **ARM** → **ARM Workbench IDE v4.1**
2. Select **Run** → **Open Run Dialog...** from the menu to open the ARM Workbench run configuration and management window.
3. Select **ARM Target** in the explorer.
4. Click the new launch configuration button.
5. Select RealView Trace 2 from the drop-down menu at the top of the new configuration **Connection** tab.

This creates a new run configuration and updates the panel to look like the one shown in Figure 3-1:

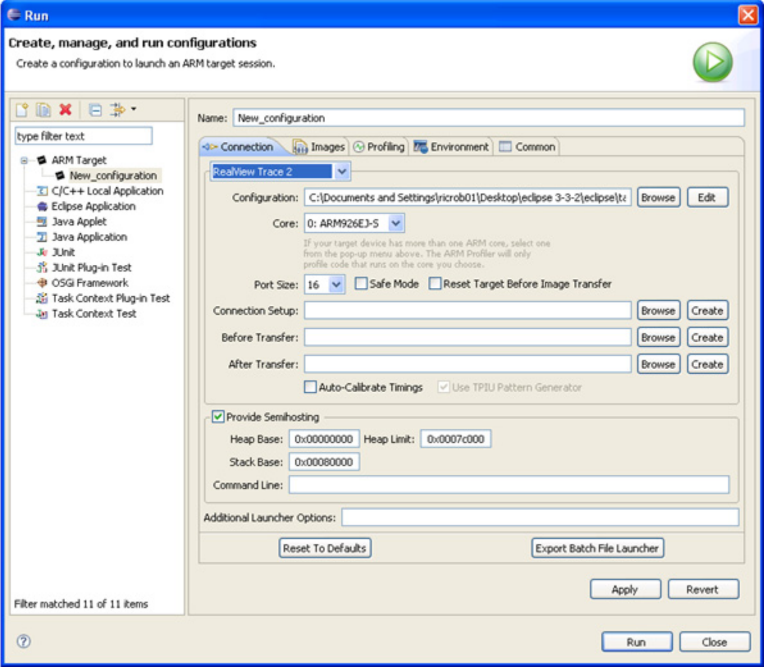


Figure 3-1 A New RealView Trace 2 Run Configuration

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.3 Setting the connection options

The **Connection** tab within the hardware run configuration window enables you to set options that relate to the target.

3.3.1 Changing the connection type

To profile using hardware, use the drop-down menu at the top of the **Connection** tab and select **RealView Trace 2**. If you use this drop-down menu to select Real-Time System Model, the ARM Profiler runs your image on a model and hardware targets connected to your host machine are not used. For more information on model-based profiling, see *Overview* on page 4-2.

3.3.2 Choosing hardware using the connection selector

To use RealView ICE and RealView Trace 2 for profiling, you must do one of the following:

- Enter the path name of a valid .rvc file in the **Configuration** field. Click the **Browse** button to search your file system.
- Click the **Create** button to open the RVConfig dialog to create a new .rvc file.

Each .rvc file found here is titled according to its configuration name.

If your device has more than one core, use the **Core** drop-down menu to choose which of the cores to profile.

3.3.3 Setting the port size

The port size is the maximum data width that can pass from the target to the RealView Trace 2 hardware.

Use the **Port Size** drop-down menu to specify the port size of the target's ETM. The port size is 16 by default, but can be changed to any of the following values:

- 1
- 2
- 4
- 8
- 16
- 32

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Note

Trace implementations with insufficient bandwidth limits may result in ETM buffer overflows when profiling in cycle accurate mode. This is most common in 8 bit or lower trace port widths.

3.3.4 Enabling and disabling safe mode

Safe mode turns off any ARM Profiler actions that might cause the target to stop. Use the **Safe Mode** checkbox to toggle safe mode on and off. By default, safe mode is disabled, but you can turn it on by checking the checkbox.

When safe mode is activated, the following hardware run configuration options are disabled:

- Reset before download
- Semihosting
- All scripting options

Any of these options could cause the target hardware to stop.

3.3.5 Resetting the target before image transfer

Select the **Reset before image transfer** checkbox to reset the target as soon as a new connection is established so that the ARM Profiler collects data from the beginning of the execution.

Note

Your target application and operating system must not configure the ETM, because it interferes with the configuration of the ETM by the ARM Profiler.

3.3.6 Setting up scripts

To set up script files to run before the execution of the image file, use the **Browse** button next to one of the script fields to search for an existing .etm script file or the **Create** button to write a new one. There are three script fields in **Connection** tab:

1. Connection Setup to set up the target connection
2. Before Transfer to execute the script file before the image is downloaded to the target
3. After Transfer to execute the script file just before the target execution starts.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Create a new .etm file and add any combination of the following commands to it:

- `Data.Set` - The `Data.Set` command can be useful in either a pre- or post- script to manually set specific memory addresses to known values before execution. Use the data set command in either of the following syntaxes:
  - `Data.Set` *`address`* `%format` *`value`*
  - `Data.Set` *`address`* `"string"`

  Use a hexadecimal address for *`address`* and a hexadecimal value for *`value`*. The possible values for %format are:
  - `%Long - 4 bytes`
  - `%Word - 2 bytes`
  - `%Byte - 1 bytes`
- `Register.Set` - Use the `Register.Set` command to specify register values. Use the register set command in one of the following syntaxes:
  - `Register.Set` *`register`* `value`
  - `Register.Set mcr_`*`opcode1_DestinationRegister_cp15RegisterNumber_RegisterAction_opcode2`* `value`. For example, `Register.Set mcr_p15_0_r1_c1_c0_0 0x5007A`

    Note

    Refer to the technical reference manual for your processor for a detailed description on how to write to the system control coprocessor.

  The possible values for *`register`* are:
  - `cpsr`
  - `r0 - r14`
  - `pc`

  Use a hexadecimal value in place of *`value`*.
- `WAIT` - Use the wait command to add a delay before the next action. Use wait in the following syntax:

  `WAIT time`

  Time can be set in either seconds or milliseconds by using either .s or .ms after the time value. For example, to set the wait time to 12 seconds enter `WAIT 12.s` in the script file.
- `Data.LOAD.Binary` *`PathName address`* - Use this command along with a path to a binary file and an address to load a binary. This is useful when loading the Symbian binary, as demonstrated in the example that follows.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

To illustrate how each of these commands can be used, an example script file that makes use of each of the three possible commands is shown in Example 3-1:

Example 3-1 A sample script

```
Register.Set mcr_p15_0_r1_c1_c0_0 0x5007A
Data.LOAD.Binary C:\SymbianROMs\PB1176ARMV6.IMG 0x0
Data.Set 0x00000004  %Long 0x0000ffff
Data.Set 0x00000008  %Long 0x00003000
WAIT 12.ms
Register.Set r1 0x12
```

3.3.7 Enabling semihosting

You can use the RealView Trace 2 launch configuration dialog to set up semihosting, a feature that enables the target to communicate I/O requests made in the application code to the host system, rather than attempting to support the I/O itself. A simple example of this is the use of a host window to provide a system console, to which the output of functions like `printf()` can be written.

To disable semihosting, uncheck the **Provide Semihosting** checkbox. If semihosting is enabled, enter memory locations in the Heap Base, Limit, and Stack Base fields to define the parameters of the heap and stack and enter any command line arguments for the image file in the Command Line field.

Note

Uncheck the **Provide Semihosting** checkbox if you are profiling on an OS.

3.3.8 Auto-calibrate timings and using a TPIU pattern generator

Check the **Auto-Calibrate Timings** option to enable the ARM Profiler to work on targets whose trace signals have imperfect timing characteristics. The auto-calibration runs instead of a standard profiling run, but you need only use this option once per target configuration feature because it writes suitable timing adjustments into the configuration file. Subsequent ARM Profiler-enabled runs use these adjustments to capture uncorrupted trace packets from the target, using trace output from the target to determine when the data signals transition in relation to the clock edge. Trace packets, or other patterns, must be output on the trace signals to accomplish this. By default, the ARM Profiler achieves this by running your image with streaming trace enabled.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

The **Use TPIU Pattern Generator** (TPIU) option activates the other method of generating output on the trace lines. This is only available on targets that have a TPIU and support test pattern generation. Checking the **Use TPIU Pattern Generator** option tells the ARM Profiler to:

1. check that the target has a suitable pattern generation facility
2. make use of it if it does.

Disable this option if you suspect that your target's TPIU is not generating patterns correctly and wish to use trace generated by the image. In most cases, it is preferable to use the pattern generator if one is available on your target, as it produces reliable transitions on all of the trace data signals, while your applications may not.

Note

The ARM Profiler disables the **Use TPIU Pattern Generator** checkbox unless you have checked the **Auto-Calibrate Timings** checkbox.

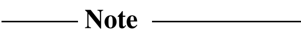
During execution, the ARM Profiler samples signals for data transitions until one of the following conditions are met:

- enough data has been collected
- the execution of the image terminates
- a time-out occurs while the ARM Profiler waits to detect edges on any remaining signals.

Note

You must provide an executable to run Auto-Calibrate. If you choose to run the Dhrystone example, set the iteration count to one billion. This gives the ARM Profiler enough time to complete the auto-calibration of your hardware.

After a successful auto-calibration, reset your target.



Note

You should run another auto-calibrated session if you increase the port size option within the ARM Profiler because you will be using trace signals that are not calibrated.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.3.9 Adding additional launcher options

To pass any additional commands to the launcher, enter them in the Additional Launcher Options field in the **Connection** tab. Enter the options exactly as you would on the command line, with the appropriate dashes and spacing. The `-h` option works here just as it would on the command line, so if you enter it here and hit the **Run** button, a list of available command line options appears in your console instead of a normal execution.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.4 Setting the image options

The **Images** panel within the run configuration window enables you to set options relating to the application.

3.4.1 Setting the working directory on the host

By default, the ARM Profiler sets the working directory to the current directory and any newly created analysis files are placed in the `ARM Profiler` directory under `My Documents` on Windows host machines and under `$HOME` on Linux host machines. To specify a different directory location, enter a new location in the Host Working Directory field or click the **Browse** button and locate the desired directory.

3.4.2 Setting the image file

Enter the directory location of an ARM executable file into the Image field or use the **Browse** button to manually search your directories for an image file to run on the target. To add multiple images, click on the plus sign below and to the right of the Image field and enter another image file. This can be repeated to define a list of image files.

Use this method to profile applications running on the Symbian OS or Linux. Enter the Symbian executable along with another application to run on an OS. For more information on profiling applications on the Symbian OS, see *Profiling your Symbian OS application* on page 12-5. For more information on Linux profiling, see *Profiling your Linux application* on page 13-4.

Note

The ARM Profiler analyzes your code regardless of the level of optimization set during compilation. At the highest level of optimization, however, the in-lining of functions may make it difficult to decipher your source code in the analysis reports.

Be aware of the following limitations when selecting an image file to profile:

- You can not use an image file with the low-level elf symbols stripped out. The ARM Profiler relies on symbols for function information.
- To get the most out of the code view report, you must also have debug information enabled during compilation. If debug information is not present, code view reports only display the disassembly panel. The process of matching source code to disassembly is not possible without debug information enabled.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

The image options drop-down menu

The image options drop-down menu enables you to choose between the following three image options:

- Load Image - Choose this option and the ARM Profiler loads the complete image on the target. At this time, only one image may be specified with this option when running on models.
- Symbols Only - Choose this option and the ARM Profiler only loads symbols from the binary image for use in profiling and does not transfer the image to the target. This option is useful for profiling an application already running on the target.
- Loaded by OS - Choose this option if the image is loaded by an OS. The ARM Profiler supports Linux and the Symbian OS.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.5 Setting profiling options

The Profiling panel in the RealView Trace 2 run configuration window contains options that relate specifically to profiling.

3.5.1 Enabling and disabling profiling

To activate profiling, you must check the box next to **Collect Profile**. Otherwise, the executable runs on the target hardware, but does not trigger the creation of an analysis file.

3.5.2 Naming the analysis file

By default, the ARM Profiler uses the `@F_@N.apd` naming scheme, where `@F` is the name of the image file and `@N` is a unique number given to analysis files in sequential order. For example, if `example_001.apd` already exists, the ARM Profiler calls the next analysis file generated `example_002.apd`. You can give newly generated analysis files any name you want by entering a valid file name into the Analysis Output field.

3.5.3 Setting the sample rate

When profiling your application, the ARM Profiler records every executed instruction, enabling it to accurately reconstruct and report the call chain sequence. In addition to that, the ARM Profiler also records timing information for executed instructions. This timing information is collected in samples from the trace stream captured by the RealView Trace 2 unit. The sample rate defines, in cycles, how frequently these samples are taken. Sampling therefore gives you an idea of how much time is spent on each instruction, which, in the bigger picture, allows you to gauge the performance of your application as a whole. A lower sample rate means more frequent samples are taken. This gives you a more accurate performance measurement but increases the volume of information sent over the trace port. It also increases the amount of data that the ARM Profiler has to parse, making it more difficult for slower host machines to keep up with faster targets.

The potential side effect of a sample rate that is too low is trace overflows. A higher sample rate, reduces the amount of data that is transmitted over the trace port, but means fewer samples are taken and the accuracy of the performance statistics reported by the ARM Profiler is reduced. The default value, 1021, tells the RealView Trace 2 unit to report the executing instruction every 1021 cycles.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Note

The ARM Profiler records every instruction executed, no matter what you have set as the sample rate. The sample rate only changes how often instruction timing information is recorded.

Use the **Sample Rate** drop-down menu to set the sample rate for the profiling run to one of the following preset values:

- Cycle Accurate
- 17
- 31
- 61
- 64
- 257
- 509
- 1021
- 1024
- Estimated Cycles.

Note

- To reduce the risk of trace port overflows, the default sample rate is set to 1021.
- The drop-down menu is populated with mostly prime numbers to ensure a more random sampling of executed instructions. This avoids the potential of a divisible sample rate matching execution loops. Some targets have a limited capacity for the sample rate that can be set.

Cycle Accurate provides the highest level of accuracy, as it records the cycle count for every instruction. If Cycle Accurate is selected for a target that does not allow it, the ARM Profiler sets it to the lowest supported value and gives you a message similar to the following: `Target does not support sample rate 1 - using 16.`

Setting the sample rate to Estimated Cycles provides maximum performance for smaller trace port widths, but turns sampling off. Just like profiling using an RTSM, the ARM Profiler estimates time for each instruction based on the instruction type and reports timing data based on these estimates, but provides no visibility to actual hardware stall behavior. This is not as accurate as sampling and does not provide insight into what instructions are performing more slowly than expected.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Note

Estimated Cycles does not use the cycle accurate mode of the Embedded Trace Macrocell (ETM) port, so it can be used with narrower ETM ports to reduce ETM buffer overflows.

Note

The Cortex-M3 allows a sample rate of either 64 or 1024. The ARM Profiler uses one of these two values based on the number chosen from this drop-down menu. Any values below 64 are converted to 64 and values above 64 are converted to 1024. The Cortex-M3 does not support the Estimated Cycles sampling setting.

3.5.4 Setting the core clock speed

The Core Clock field is only active if you set the sample rate to **Estimated Cycles** using the Sample Rate field. When you set the sample rate to Estimated Cycles, the target does not produce cycle information and the ARM Profiler has no way to measure the core clock speed. Setting the core clock speed to match your hardware ensures that the CPI data in the reports is accurate.

3.5.5 Setting an execution time limit

By default, a profiling run executes until it is finished or is terminated manually. To set up a finite test run, enter the maximum amount of hours, minutes, and seconds you would like the execution to continue into the Maximum Run Length fields.

3.5.6 Enabling instruction trace replay

Use the **Collect Instruction Trace Replay** checkbox to enable or disable program trace collection at runtime. When turned on, the ARM profiler stores every instruction called and presents it in the Replay View. You can then step forward and backward through the history of instruction calls in the Trace View, very much like a debugger. Enabling program trace adds the Replay view to the list of available ARM Profiler views and a number of options specific to navigating the additional trace data. For more information on program trace, see *The replay view* on page 7-15.

Note

- Collecting instruction trace replay does not have a significant impact on the host data collection speed if the sampling rate is set to cycle accurate. Program trace slows down host data collection when using faster sampling rates. If you are getting `trace buffer overflow` errors, try disabling this option.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- When program trace is turned off the size of the analysis file is not significantly affected by the length of execution. This is not the case when program trace is enabled because, in addition to recording which instructions are executed, the ARM Profiler logs the order in which the instructions were executed and the time each instruction took to complete.

3.5.7 Enabling ARM Profiler to generate ETM context IDs

When this option is checked, ARM Profiler creates ETM context IDs during capture. These are necessary for the ARM Profiler to capture data from an application running on a target OS. Enable this option if you are profiling on either Linux or the Symbian OS.

3.5.8 Enabling and disabling full live update

By default, the ARM Profiler provides a live update screen that provides streaming data as the image executes on the target. Disabling the full live update option could improve performance for slower host systems, but the statistics won't accumulate on the live update screen. The stop button is active even if you have disabled full live update.

3.5.9 Generating Profiler Guided Optimization Data

Select the **Generate Profiler Guided Optimization Data** checkbox to force the creation of a `.apa` file. This file is required if you want to use the profiled execution to inform the optimizations of the ARM Compiler toolchain. For more information on using the guided optimization feature, see:

- *ARM® Compiler toolchain Using the Compiler* (ARM DUI 0472)
- *ARM® Compiler toolchain Using the Linker* (ARM DUI 0474).

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.6 Exporting a launch script

You can use the **Export Script** button within the run configuration window to export the current configuration to an `ApplicationName.bat` (or `.sh` shell script, if you are using Red Hat Linux) file with all of the options you have specified in command line format. In this way, you can use the commands in the `ApplicationName.bat (.sh)` file to add to your own existing build script or use the generated file as a standalone script.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.7 Executing the run configuration

When all of the parameters are entered into the hardware run configuration dialog, you are ready to execute the image file on the target hardware to create a new analysis file. Make sure your target hardware is connected and powered on, then click the **Run** button at the bottom of the run configuration window.

The live update view appears as the application starts and shows you graphs, lists and progress information that updates as your code continues its execution. For more information on the live update window, see *Live update* on page 5-13.

The execution continues until one of the following conditions is met:

- The application finishes its run
- You terminate the run by pressing the stop button in the **Progress** section of the live update view
- The run reaches the time limit defined in the hardware run configuration dialog, if this option is enabled.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.8 Hardware profiling restrictions

When preparing your source code for profiling using the ARM Profiler in conjunction with RealView ICE and RealView Trace 2, it is important to keep the following points in mind:

- The ARM Profiler is not a debugger. Its purpose is to provide run time information to identify code execution bottlenecks, not to isolate where your code might be broken. Use RealView Debugger to make sure your code is in working condition before running it through the ARM Profiler.
- The ARM Profiler does not track memory interactions. It knows when an opcode accesses memory and tallies the size of the memory access in the **Accessed** column of the table reports, but it cannot track whether the memory access is to a cache or a slow external memory.
- Hardware profiling provides enough detailed information to make intelligent assumptions about your source code. It does this by providing a value for average cycles per instruction and delay. However, it does not know exactly why an instruction performs more slowly than expected. To better optimize your code, you must understand which opcodes have the biggest discrepancies.
- Currently, the ARM Profiler supports the ARM7TDMI™, ARM920T™, ARM926EJ-S™, ARM946E-S™, ARM966E-S™, ARM1136JF-S™, ARM1156T2F-S™, ARM1176JZF-S™, Cortex™-A5, Cortex™-A8, Cortex™-A9, Cortex™-R4, and Cortex™-M3 processors.
- The ARM Profiler only supports connections to targets configured for little endian operation. It does not currently support targets configured for big endian operation.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

3.9 Hardware profiling execution speed

Profiling does not have any effect on your application's execution speed on hardware.

Downloading your image to the target can take a little longer than normal when you enable profiling. You may also notice a delay when you terminate execution while analysis report is being generated. The amount of time required to generate an analysis file depends on varying factors, such as the sample rate and whether or not you have enabled program trace. The Backlog field in the live update view gives you a good indication of how much longer is needed for the ARM Profiler to create an analysis file.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 4
Data Collection Using a Real-Time System Model

This chapter describes the process of generating analysis reports using a Real-Time System Model (RTSM). It includes sections on how to enable profiling in the ARM Workbench **Run...** dialog, how to modify your standard build process in both Windows and Red Hat Linux to enable the ARM Profiler, and how to enable profiling from the command line.

It contains the following sections:

- *Overview* on page 4-2
- *Using the ARM compilation tools in the ARM Workbench* on page 4-3
- *Creating a profiling-enabled RTSM run configuration* on page 4-4
- *Setting the connection options* on page 4-5
- *Setting the image options* on page 4-7
- *Setting the profiling options* on page 4-8
- *Running the configuration* on page 4-10
- *Enabling profiling outside of the ARM Workbench* on page 4-11.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

4.1 Overview

To perform a comprehensive analysis of the behavior of your application in a modeled hardware environment, the ARM Profiler must observe it in action. You can use the ARM Compiler toolchain to build your image file, as normal, and one of the ARM RTSMs included with the ARM Profiler to execute your image file.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

4.2 Using the ARM compilation tools in the ARM Workbench

RealView Development Suite uses the ARM Workbench to create, manage, build, and profile your projects. This provides an integrated environment for all of the components of RVDS v4.1.

If you require help with generating an image file using the ARM Compiler toolchain, see the ARM Compiler toolchain documentation. For a list of ARM Compiler toolchain documents pertinent to the ARM Profiler, see *Further reading* on page xii.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

4.3 Creating a profiling-enabled RTSM run configuration

You must build an image file using the ARM compiler before you can start profiling your code. To initiate a profiling-enabled run on one of the supplied RTSMs, follow these steps:

1. Select an axf file in the Project Explorer.
2. Select **Run As → Open Run Dialog...** from the menu to open the ARM Workbench run configuration and management window. Notice the launch configuration panel is now populated with data specific to the axf file you selected in the previous step.
3. Click on the ARM Target in the explorer pane on the left and then click the new launch configuration button in the toolbar. After the successful creation of an ARM target run configuration, the ARM Workbench run window looks like the one shown in Figure 4-1:

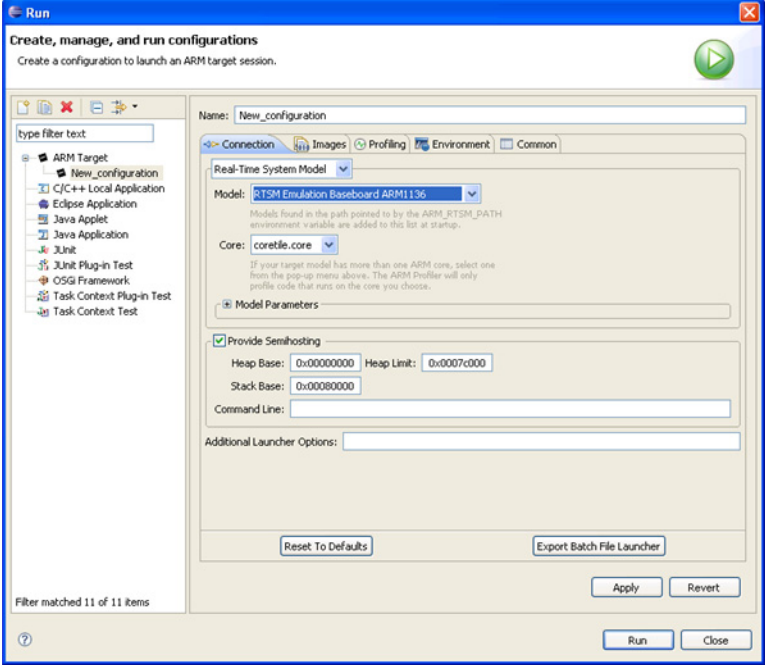


Figure 4-1 A New ARM RTSM Run Configuration

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

4.4 Setting the connection options

Select Real-Time System Model from the drop down menu at the top of the **Connection** tab of the run configuration window and use the drop-down menus to choose the hardware that you want to simulate.

From the **Model** drop-down menu, you can choose any of the predefined ARM models to run on, or select a custom option.

The following pre-defined models are available:

- RTSM_EB_ARM1136
- RTSM_EB_ARM1176
- RTSM_EB_ARM926
- RTSM_EB_Cortex-A5_MPx1
- RTSM_EB_Cortex-A5_MPx2
- RTSM_EB_Cortex-A8
- RTSM_EB_Cortex-A9_MPx1
- RTSM_EB_Cortex-A9_MPx2
- RTSM_EB_ARM_Cortex-R4
- RTSM_MPS_ARM_Cortex-M3.

If none of these predefined options match your target architecture, you can use your own customized RTSM. Place your RTSM files in the directory to which the system variable `ARM_PROFILER_RTSM_PATH` points. Any RTSM files found by the ARM Profiler will be added to the Model drop-down menu. You need a licensed copy of RealView System Generator to create custom RTSMs.

If you are profiling on a multi-core model, use the **Core** drop-down menu to choose which of the cores to profile. The ARM Profiler can not profile multiple cores during a single profiling run.

4.4.1 Setting model parameters

To set specific model parameters, use the Model Parameters disclosure control to reveal a series of fields and checkboxes that enable you to fine tune the configuration of the RTSM to more exactly match your existing hardware. Customizing the model parameter settings is optional. You can profile using the pre-configured models with the default model parameter settings.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

4.4.2 Enabling semihosting

The **Provide Semihosting** checkbox enables you to turn on semihosting, a mechanism that captures I/O requests made by code running on the target system and communicates these to the host system for handling. If semihosting is enabled, the Heap Base, Limit, and Stack Base fields are also active. Enter memory addresses in these fields to dictate the start of the heap and the stack and the maximum size to which the heap can grow.

The Launcher Command Line field enables you to pass command line arguments to the specified ARM executable. Enter them here exactly as you would on the command line.

Note

Uncheck the **Provide Semihosting** checkbox if you are profiling on an OS.

4.4.3 Adding additional launcher options

Use the Additional Launcher Options field in the run configuration dialog to pass options through to `model_shell`, a tool used to run the RTSM. Enter the `-h` option here and click the **Run** button and a list of all the options available for use with `model_shell` appears in the ARM Workbench console. For further details on setting model parameters, see the *RealView Development Suite Real-Time System Model User Guide*.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

4.5 Setting the image options

The second panel in the run configuration window is the **Images** tab. It provides options that relate to the image.

4.5.1 Setting the host working directory

By setting the working directory you are telling `model_shell`, the tool used to run the model, where to execute. By default, the working directory is set to the project directory of the selected file or the workspace directory if no file is selected. To change this behavior and have `model_shell` execute in a different location, either:

- enter a directory location in the **Host Working Directory** field
- click the **Browse...** button and locate the desired directory on disk.

4.5.2 Choosing image files

The Image field tells the ARM Profiler which ARM executable file you want to profile. Enter the directory location of the ARM executable file directly in the field or use the **Browse...** button to manually search your directories for the target file. To add multiple images, click on the plus sign below the Image field and enter another image file. This can be repeated to define a list of image files to execute.

The image options drop-down menu

The image options drop-down menu enables you to choose between the following three image options:

- Load Image- Choose this option and the ARM Profiler loads the complete image on the RTSM. When using an RTSM to profile, you can only load one image.
- Symbols Only- Choose this option and the ARM Profiler uses symbols from the image's binary for profiling, but does not load the image to the RTSM. Use this option when defining multiple image files.
- Loaded by OS - Choose this option if the operating system loads your application dynamically.

Note

The ARM Profiler does not connect to targets configured for big endian operation.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

4.6 Setting the profiling options

The Profiling panel in the run configuration window contains options that relate specifically to profiling.

4.6.1 Enabling profiling

To enable profiling, you must have the **Collect Profile** option selected. If this box is left deselected, profiling is disabled and the ARM Profiler does not create an analysis file at the end of execution.

——— **Note** ———

If both profiling and semihosting are disabled, the ARM Profiler only launches the images defined in the Images tab. It has no further involvement in the execution of the images. If profiling is disabled, but semihosting is turned on, the ARM Profiler handles semihosting during execution.

———————————

4.6.2 Naming the analysis file

If profiling is enabled, you can use the Analysis File field to name the resulting analysis file. By default, the ARM Profiler uses `@F_@N.apd`, where `@F` is the name of the image file and `@N` is a variable used to add a unique number to the end of the file name. Thus the second file generated using `example.axf` would be called `example_002.apd`, by default. You can use your own custom title by replacing the name in the **Analysis Output** field with your own.

4.6.3 Setting the execution time limit

By default, a run continues until it finishes or you terminate it. To add a limit to the execution time so that a run stops automatically after a defined period, enter values in the Maximum Run Length fields.

4.6.4 Enabling and disabling instruction trace replay

Use the **Collect Instruction Trace Replay** checkbox to enable or disable program trace collection at runtime. When turned on, the ARM profiler stores every instruction called and presents it in the Replay View. You can then step forward and backward through the history of instruction calls in the Replay View, very much like a debugger. Enabling program trace adds the program trace view to the list of available ARM Profiler views and a number of options specific to navigating the additional trace data. For more information on program trace, see *The replay view* on page 7-15.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Note

When program trace is turned off the size of the analysis file is not significantly affected by the length of execution. This is not the case when program trace is enabled because, in addition to recording which instructions are executed, the ARM Profiler logs the order in which the instructions were executed and the time each instruction took to complete.

4.6.5 Enabling full live update

By default, the ARM Profiler provides graphs that update as the application executes. Turning off full live update provides a minor boost to performance, but most of the statistics in the live update view are disabled.

4.6.6 Enabling disk backlogging

By default, the ARM Profiler creates a data buffer on your disk to temporarily store data as it executes. Uncheck the **Disk Backlog Allowed** checkbox to disable this behavior. If disk backlogging is turned off, the ARM Profiler slows the model down accommodate to its data collection.

4.6.7 Generating Profiler Guided Optimization Data

Checking the **Generate Profiler Guided Optimization Data** checkbox forces the creation of a `.apa` file, necessary if you want to use the profiled execution to inform the optimizations of the ARM Compiler toolchain. For more information on using the guided optimization feature, see:

- *ARM® Compiler toolchain Using the Compiler* (ARM DUI 0472)
- *ARM® Compiler toolchain Using the Linker* (ARM DUI 0474).

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

4.7 Running the configuration

When all of the parameters are defined, click the **Run** button located in the bottom right corner of the run configuration dialog to execute the configuration. When the execution is finished, the newly-generated analysis file automatically opens in the editor section of the ARM Workbench.

——— **Note** ———

- Running your code on an RTSM with profiling enabled slows the simulation speed on the host machine because the host machine will need to divide its resources in order to simultaneously simulate and profile your code. While it does not slow things down prohibitively, it is not quite as fast as an unprofiled execution.

- Analysis file size is more dependent on the size of the code being tested than on execution length. Execution time should not have a profound impact on the size of the analysis file, unless trace replay is enabled. If you turn on the **Collect Instruction Trace Replay** option, the ARM Profiler logs every instruction's order of execution and timing values. This can consume a lot of disk space.

———

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

4.8 Enabling profiling outside of the ARM Workbench

To use the ARM Profiler outside of the ARM Workbench, you must modify your standard build process, either invoke the ARM Profiler from a shell script (Red Hat Linux) or batch (Windows) file.

4.8.1 Using the export script command

To enable profiling and execute code from a batch file or shell script, the ARM Profiler provides the **Export Script** button in the RTSM run configuration window within the ARM Workbench. To use this feature, follow these steps:

1. Select **Run** → **Open Run Dialog...**
2. Select an RTSM run configuration in the explorer.
3. Click the **Export Script** button

If you are running the ARM Profiler on Windows, this command produces an `ApplicationName.bat` file with all the appropriate commands to do a profiled run. In the xvid example, the generated output looks something like this:

```
@echo off

REM Automatically generated on 8/6/08 2:34 PM
set ARM_PROFILER_TOOLS=C:\Program Files\ARM\Profiler\tools\2.1.0.200808191200
pushd C:\workspace\xvid
"%ARM_PROFILER_TOOLS%\tools\rtsm\win32\model_shell.exe" -m
"%ARM_PROFILER_TOOLS%\tools\rtsm\win32\RTSMEmulationBaseboard_CT1136.dll" -a
coretile.core=xvid.axf --timelimit 2147483 -C
coretile.core.profiler-enable=1 -C
coretile.core.profiler-output_file=@F_@N.apd -C
coretile.core.vfp-enable_at_reset=1 -C
coretile.core.semihosting-heap_base=0x00000000 -C
coretile.core.semihosting-heap_limit=0x0f000000 -C
coretile.core.semihosting-stack_base=0x10000000
popd
```

This command produces a shell script instead of a batch file in Red Hat Linux:

```
#!/bin/sh

# Automatically generated on 8/6/08 10:31 AM

export ARM_PROFILER_TOOLS="/home/ARM/Profiler/tools/2.1.0.200808191200"
cd tools/ARM/Profiler/Contents/2.1/0/examples/xvid
LD_LIBRARY_PATH=/home/ARM/Profiler/tools/2.1.0.200808191200/rtsm/linux
$ARM_PROFILER_TOOLS/rtsm/linux/model_shell -m
$ARM_PROFILER_TOOLS/rtsm/linux/RTSMEmulationBaseboard_CT1136.so -a
```


Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

```
coretile.core=xvid.axf --timelimit 2147483 -C
coretile.core.profiler-enable=1 -C
coretile.core.profiler-output_file=@F_@N.apd -C
coretile.core.vfp-enable_at_reset=1 -C
coretile.core.semihosting-heap_base=0x00000000 -C
coretile.core.semihosting-heap_limit=0x0f000000 -C
coretile.core.semihosting-stack_base=0x10000000
```

You can use any or all of the commands created by the export script in your own build system, to make profiling your code a seamless part of the process.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 5
The Analysis Summary

This chapter describes the analysis summary, the default view when you open an analysis file. It includes an overview, a description of how to open an analysis file, and an in-depth look at each component of the analysis summary.

It contains the following sections:

- *Analysis summary overview* on page 5-2
- *Opening an analysis summary* on page 5-3
- *Analysis summary elements* on page 5-4
- *Live update* on page 5-13.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

5.1 Analysis summary overview

The analysis data provided by the ARM Profiler allows you to quickly uncover the performance bottlenecks.

The analysis summary is the first report you see when you open an analysis file. From here, you can navigate to all of the other report types using the links beneath the **Overall Analysis** section in the top left corner of the report. It also gives a top-level overview of your profiling results. The top five self time consuming functions are shown as a bar chart, and your code coverage percentages are shown for functions and instructions as pie charts. For a detailed description of the **Self Time** statistic, see *Table report column headers* on page 6-3. The Timeline section describes the behavior of code at intervals. Hover over the timeline graphs to see which processes, threads and functions are dominant at any given time during execution.

The Call Chains by Stack Depth chart gives you an overview of stack depth distribution. In addition to the navigation section, the bar charts and pie charts can also be used to navigate to more detailed reports. If you want to see more information on a critical function, you can click on that function in the bar chart labeled Top 5 Functions by Self Time to open the code view report for that function.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

5.2 Opening an analysis summary

To open an analysis file, double-click on the analysis file in the ARM Profiler Data view, or right-click on the analysis file and choose **Analysis Summary** from the contextual menu. Either action opens the analysis summary in the editor section of the ARM Workbench, a quick top-level glimpse at the performance of your application.

The ARM Profiler Data view displays the analysis file icon next to any analysis file. Note that analysis files are usually named according to their project title and order of creation. For example, the first analysis file created in the xvid project is called `xvid_001.apd`, by default.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

5.3 Analysis summary elements

In addition to providing general information and navigation options, the analysis summary provides information about where to begin your search for optimizations.

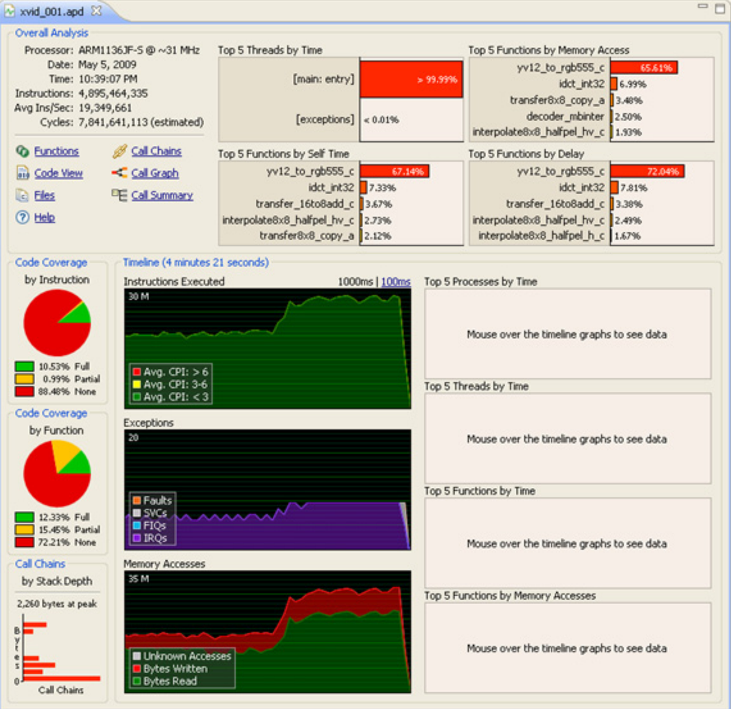


Figure 5-1 A sample analysis summary

As you can see in Figure 5-1, the analysis summary gives you a general overview of your algorithm's performance and is divided into the following sections:

- Overall analysis
- Navigation
- Code Coverage by Instruction
- Code Coverage by Function
- Top 5 Functions by Self Time
- Top 5 Threads by Time
- Top 5 Functions by Delay

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Top 5 Functions by Memory Access
- Call Chains by Stack Depth
- Timeline

For the purposes of this section, the code coverage pie charts and top five bar charts are grouped together.

5.3.1 Navigation

Located in the top left of any analysis summary, the report links section enables you to open any of the ARM Profiler report types:

- Functions - lists statistics for every function organized by execution, code coverage and time spent.
- Code View - shows you both your source code and disassembly code with line-by-line performance statistics for a finer-grain level of detail.
- Classes - contains statistics broken down by C++ class.
- Files - contains statistics broken down by source file.
- Call Chains - a hierarchical table report that allows you to explore every branch of your code. In it, statistics are broken down by instance, so if a function is called in more than one place, you can see its usage statistics for each of the functions it is called from.
- Call Graph - a visual representation of your code's hierarchy, color-coded to show you the performance bottlenecks. The top 5 Functions by Self Time are colored using the same palette as the top five bar charts.
- Call Summary - enables you to explore your code visually to see how certain function instances and called functions affect your code's performance statistics. Unlike the call chains report, the call summary gives a visual representation of your function instances instead of hierarchical table report data.
- Help - opens the ARM Workbench help view with a list of help topics that relate specifically to the analysis summary. To access this list, click on the **Help** link or press **F1** (**Shift + F1** in Red Hat Linux) when the analysis summary is active.

Note

The classes report only appears in reports generated using C++.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

If any filtering is currently active, the **Remove Filtering** link appears below the other navigation links. Clicking this link removes active filters from all reports. This is pictured in Figure 5-2:

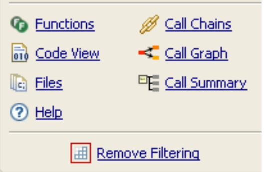


Figure 5-2 The Remove Filtering Link

5.3.2 Overall analysis section

The overall analysis section provides you with information regarding the execution count, total time and sampling information, if applicable. It reports the following:

- The processor type and speed
- The date the run took place
- The time the run took place
- The number of instructions executed
- The total execution time, measured in seconds
- The average number of instructions executed per second
- The sampling rate, if sampling was used
- If you did not turn on sampling, the overall analysis section reports the total number of estimated cycles

5.3.3 The code coverage pie charts

The code coverage pie charts are located in the bottom left of the analysis summary and give you a top-level look at the percentage of functions and instructions actually executed during the captured execution.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

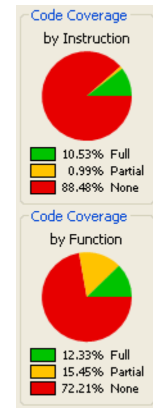


Figure 5-3 Sample code coverage pie charts

The code coverage by instruction pie chart shows graphically the percentage of executed assembly instructions. Green represents the percentage of completely executed instructions, red represents unexecuted instructions, and yellow represents partially executed instructions. For example, if the application follows only one path of a conditional instruction, the instruction is marked as having partial code coverage.

The second pie chart is similar to the code coverage by instruction chart, but shows code coverage by function. Here, the yellow partial coverage slice is likely to be larger as functions where even one instruction is not fully executed are labeled as having partial coverage.

Double-clicking on one of the pie chart slices opens the functions report with each function of that code coverage type highlighted. If accessed in this manner, the functions report automatically sorts by code coverage. For example, double-clicking on either of the red slices opens the functions report with all of the unexecuted functions selected and centered in view.

5.3.4 The call chains by stack depth chart

The stack depth histogram provides a summary of the call chains' stack usage during the captured execution. The vertical baseline measures the maximum stack depth usage during execution while the red horizontal lines signify the number of call chains that had a stack depth value that fell in each range. If only a few call chains have a high stack depth value, the red bars at the top of the chart are very small or non-existent. In this case, reducing overall application stack usage should not be difficult, as there are only a few call chains to optimize. If the call chain by stack depth chart shows a lot of long horizontal bars at the top of the chart, reducing overall application stack usage is more difficult.

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

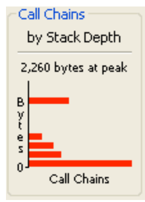


Figure 5-4 The stack depth histogram

Double-clicking on a bar in the Call Chains by Stack Depth chart opens the call chains report with each of the represented call chains' leaf nodes selected. If, for example, you double click on a red bar that represents the five call chains with the highest stack depth value, the call chains report opens with the leaf functions in the hierarchy of these call chains selected. Double-clicking on a non-bar area, such as where the bytes at peak info is listed, will open the functions report and sort it by the Order column.

5.3.5 The top five bar charts

The top five bar charts show the overall performance of your code, displaying the top five functions in a few different categories.

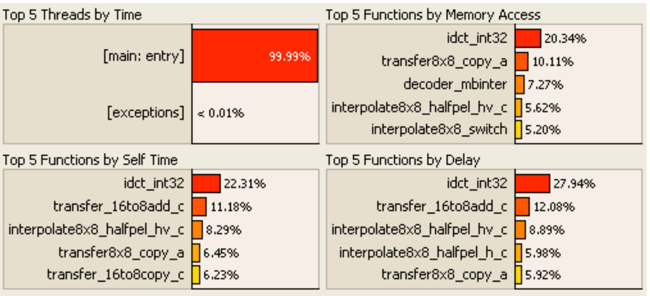


Figure 5-5 The Top Five Bar Charts

In the Top 5 Threads by Time bar charts, the time value depends on the chosen sample rate. If you are using hardware and sampling is turned on, then the time value is based on samples. If you are using estimated cycles, the time value will be based on cycles calculated by using the instruction count and type. To see a list of the different threads generated by your code, alongside more detailed performance statistics, open the call chains report.

The Top 5 Functions by Self Time bar chart shows you which functions most frequently occur during the captured run. For example, if an application spends half its time in one particular function, that function will be represented by a bar labeled 50%. Double-clicking on any of the bars opens the code view with that function selected. This

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

is not the only navigation option provided by the top five functions bar chart. Right-click on any of the chart's bars to open a contextual menu with the following menu options:

- Set Breakpoint
- Run Forward to Selection
- Run Backward to Selection
- Run Forward off Selection
- Run Backward off Selection
- Jump to Next Instance in Replay
- Jump to Previous Instance in Replay
- Filter
- Filter Others
- Select in Functions
- Select in Classes
- Select in Files
- Select in Code View
- Select in Call Graph
- Select in Call Chains
- Select in Call Summary
- Edit Source

The first seven menu items are only available if you checked the **Collect Instruction Trace Replay** box in the **Profiling** tab of the run configuration launcher. The **Set Breakpoint** option sets a breakpoint in the instruction trace where the replay will halt when you execute one of the run commands. The **Run Forward to Selection** and **Run Backward to Selection** commands can then be used to move the trace position between breakpoints.**Run Forward off Selection** moves forward to the next unselected instruction, the next breakpoint, or the end of the program trace, while **Run Backward off Selection** works the same, only it moves backward. The **Jump to Next Instance in Replay** and **Jump to Previous Instance in Replay** take you forward and backward through the trace to the next instance of this function's instructions in the Replay View.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

The next menu options, **Filter** and **Filter Others**, filter the selected function's data from all reports. **Select in Functions** opens the functions report, with the function selected. **Select in Classes** only appears if the analysis file has been generated using C++. It opens a classes report with the chosen function's class selected. **The Select in Files** option opens the files report with the function's source file selected. **Select in Code View** opens the code view for the selected function, with all of the instructions and pertinent lines of code highlighted.**Select in Call Chains** opens the call chains report with every instance of the function selected and the hierarchal report broken down so that they are all visible. The **Select in Call Graph** and **Select in Call Summary** options open the visual reports with the function active and centered, while the **Edit Source** option opens the function's source file in the ARM Workbench default editor.

——— **Note** ———

The **Edit Source** contextual menu option only appears if the file is a standard C or C++ file. If the bar represents a function without source, like a third-party library function, the **Edit Source** option does not appear.

————————

The **Top 5 Functions by Memory Access** and **Top 5 Functions by Delay** charts are similar to the **Top 5 Functions by Self Time** bar chart, only they display the five functions with the highest **Delay** and **Accessed** values, respectively. The navigation options for this bar chart are the same as in the **Top 5 Functions by Self Time** report.

——— **Note** ———

If the program is very small and there are not enough files or functions to populate the top five bar charts, the ARM Profiler fills the charts with as much data as possible and increases the thickness of the bars to fill the space.

————————

5.3.6 The Timeline

In addition to aggregate data, the summary report also shows you graphs that break down the performance of your code by time. The line graphs depict your code’s behavior in terms of instructions, exceptions, and memory accesses. Hovering over an interval in these charts activates a vertical line in the graph and the Top 5 bar charts to the right of timeline.

To pan across the timelines and see intervals outside of the currently visible values move the mouse cursor to the bottom portion of the graph. This highlights the bottom section of the graph and changes your cursor to a hand. Hold your mouse button down and drag left or right to move the timeline forwards and backwards.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

In the upper left hand corner of all three of the timeline graphs is a numeric key. This number represents the highest value shown in the graph and varies depending on the upper limit of the data collected during the execution.

You can specify the interval length by using the controls in the upper right hand corner of the **Timeline** section. Click on the 1000 ms or 100 ms links to toggle between these values.

The first graph in the **Timeline** section is the Instructions Executed graph, pictured in Figure 5-6:

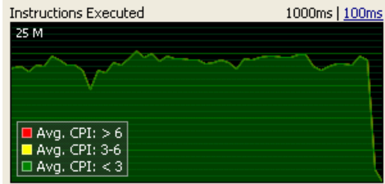


Figure 5-6 The Instructions Executed timeline graph

The instructions bar chart shows you the total number of instructions and breaks them down by their average CPI. Red indicates an average CPI of six cycles or greater, yellow represents an average CPI between three and six, and green means an average CPI of three or less.

The Exceptions line graph is located underneath the Instructions Executed graph and is pictured in Figure 5-7

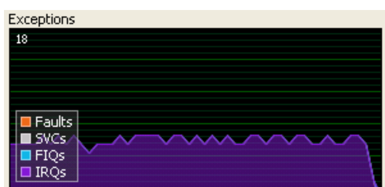


Figure 5-7 The Exceptions timeline graph

The highest line in the exceptions graph represents the total number of exceptions that occurred during each interval. These are further broken down by type with each type represented by a different color:

- Faults - All exceptions that are not FIQs, IRQs, or SVCs fall into this category. This includes prefetch aborts, data aborts, and, undefined instructions.
- SVCs - Supervisor calls, normally used to request privileged operations or access to system resources from an operating system.

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- FIQs - Fast interrupt requests
- IRQs - Interrupt requests

The Memory Accesses timeline graph is just below Exceptions and is pictured in Figure 5-8:

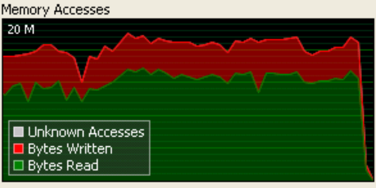


Figure 5-8 The Memory Accesses timeline graph

The top of the Memory Accesses timeline graph represents the total accesses to external memory, in bytes, triggered by the profiled code during execution. Like the other timeline graphs, it tracks this value per interval. The red area of the graph represents the number of bytes written, while the green area represents bytes read.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

5.4 Live update

In addition to the static analysis summary, the ARM Profiler shows you the data in real time, as the code executes, using the live update feature. It provides a graphical overview of the compiled data and second-by-second snapshots of where your code is spending its time and resources.

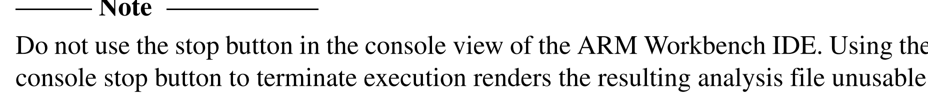
5.4.1 Progress section

The **Progress** area of the live update window displays reference information during your code's execution, as shown in Figure 5-9:

Figure 5-9 The Progress section

- Progress bar- The value above the bar is how much data has already been written to disk. The amount of data waiting to be processed in cases where your host computer cannot keep up with the amount of incoming trace data is shown below the bar. As the amount of backlog increases, so does the yellow fill in the progress bar. After you press the stop button, the backlog value will decrease. After it reaches zero, an analysis file is created. This process is visually represented by a green fill bar slowly replacing the yellow. Due to compression, backlog data size does not translate directly to disk usage amount. It will get smaller as the ARM Profiler processes it.
- Total Instructions - The sum of instructions executed across all threads and processes.
- Disk Space Remaining - The amount of disk space left on the host machine.

Use the red stop button, located to the left of the status bar, to stop the current execution and trigger the creation of an analysis file.



Note

Do not use the stop button in the console view of the ARM Workbench IDE. Using the console stop button to terminate execution renders the resulting analysis file unusable.

5.4.2 Activity section

The activity section is pictured in Figure 5-10 on page 5-14:

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

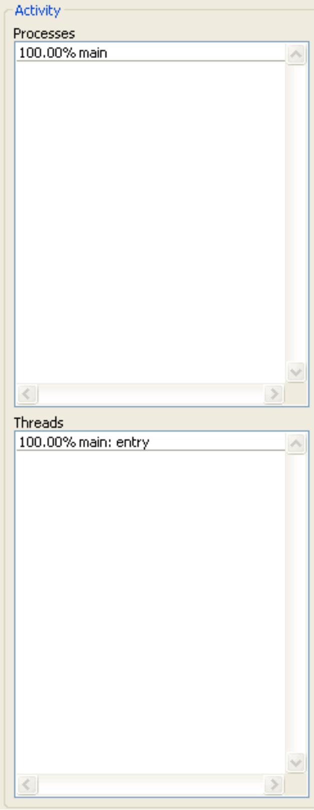


Figure 5-10 The Activity section

This section contains two scrollable lists revealing all currently running processes and threads. Next to each list item is the percentage of time spent in that process or thread during the last sample period.

5.4.3 Graphs

The graphs section of the live update window is very similar to the timeline section of the summary report, only the graphs here update every 1000 ms. The single column to the right of the line graphs tracks the values for the current interval while the line graphs track past values. These graphs are pictured in Figure 5-11 on page 5-15

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

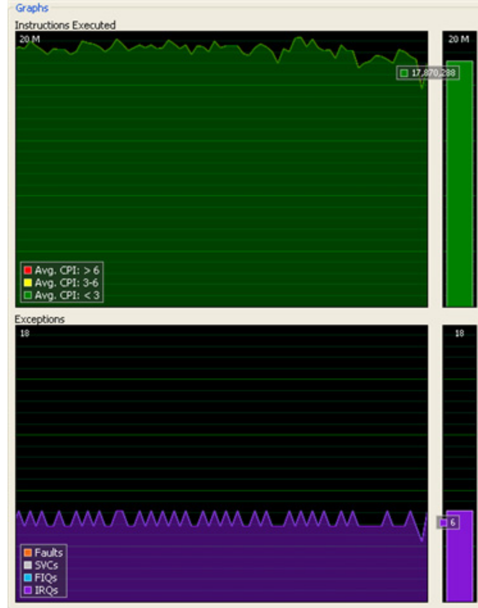


Figure 5-11 The live update graphs

For more information on how to navigate these graphs and an explanation of the data they contain, see *The Timeline* on page 5-10.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 6
The Table Reports: Functions, Files, Classes, and Call Chains

This chapter provides an in-depth look at the functions report, the files report, the classes report, and the call chains report.

It contains the following sections:

- *Table report basics* on page 6-2
- *Navigating to other reports* on page 6-13
- *The functions report* on page 6-17
- *The classes report* on page 6-18
- *The files report* on page 6-19
- *The call chains report* on page 6-20.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

6.1 Table report basics

The data contained in the analysis summary is just the beginning. All of the reports listed in the analysis summary's navigation section provide a more thorough look at your code's interaction with the hardware. This chapter focuses on the four reports that are laid out in tables:

- Functions report
- Classes report
- Files report
- Call chains report

Each of these report types are broken up by the unit type indicated in the report's title. Though statistical fields presented in the columns differ from report to report, there is a shared functionality in these report types. The toolbar, for example, contains a set of icons that, for the most part, are common to all report types and you can right-click on a row in any table report to open a contextual menu that enables you to navigate to the other report types. Exporting data from any of the table report types into a .txt, .tab, or .csv format is the same basic process for all table reports.

Before delving into the differences between the table reports, this chapter explores their commonalities.

6.1.1 Opening a table report

To open any of the table reports, click on its corresponding link in the summary report's navigation section, located in the upper left corner, above the code coverage pie charts, as shown in Figure 6-1.

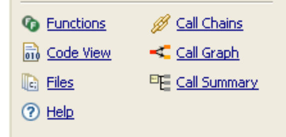
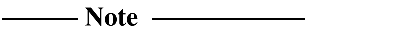


Figure 6-1 The Navigation Section of the Summary



Note

The classes report does not appear in the navigation section of the summary report for standard C code analysis files.

All of these report types are filled with data displayed in rows and columns, like a spreadsheet.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

6.1.2 Table report column headers

Column headers vary by report type, as the statistics represented in them are dependent on whether they are broken down by function, file or class. Here is a list of all of the column headers contained in the table reports:

- Name - The name of the function, file, or class, as described in the source code.
- Coverage - The percentage of code actually executed. The color-coded symbol next to the percentage gives a visual indication of code coverage. A red circle indicates that the code was not called. A yellow triangle represents partial coverage. If any of the function, class or file's instructions were executed, but not completely, it is considered partially covered. A green square indicates 100% coverage; every line of code was executed, including both paths of conditional instructions.
- Self Time - The values in this column indicate the amount of time that was spent executing instructions contained exclusively in this function, file or class. The units vary depending on the method of data collection. If you used the RealView ICE and RealView Trace 2 hardware to generate analysis data, time is measured in samples. If your source code was profiled against a Real-Time System Model, time is presented in estimated cycles. Samples, collected during runtime on actual hardware, are an accurate measure for time spent in a function when a large enough data sample is tested. RealView Trace 2 checks the target at an interval determined by the sampling rate and reports the instruction being executed. The ARM Profiler reports the number of times the application executed one of this function's instructions during the sampling process. Estimated cycles, used to measure performance on a Real-Time System Model, are an estimate of the number of cycles spent executing the function based on hardware configuration and instruction counts.
- Total Time - Total time is measured in the same manner as self time, but represents the amount of time used by the function *and* all of the functions it calls. While a function may have a very low value in the self time field, its total time can be much higher if the functions it calls use a lot of time. This statistical field is not applicable to the file or class reports, so it only appears in the function and call chain reports.
- Avg CPI - Cycles per instruction (CPI) measure how many clock cycles it takes to execute a single assembly instruction. Average CPI is the average number of cycles it takes to execute a single instruction within the function, class or file.
- Functions - The number of functions contained in the class or file.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Delay - Delay measures the delta between the function, class or file's expected execution time without interlocks or other delays and its actual execution time. other delays include cache misses, bus arbitration, and page misses, to name a few. The ARM Profiler does not provide insight into what, besides interlocking, might have caused the delay, only that the delay occurred. If the data gathered in the current analysis report is captured using the RealView ICE and RealView Trace 2 hardware, the data listed here is a calculated value based on samples collected during the run and this real data is measured against the expected cycles per instruction. If the data is collected using a Real-Time System Model, the delay is the total number of interlock cycles caused by this function's instructions multiplied by the call count for each instruction.
- Eff - The efficiency of the function, class or file. A ratio of delay to self time.
- Called - The number of times this function is called.
- Callers - The number of unique functions that called this function.
- Callees - The number of unique functions called by this function. To see all of the functions called by this function, right-click on the function and choose the 'Select in Call Summary Option'. Doing this opens up the call summary, a report that includes all of the called functions as branches to the right of the selected function. You can filter certain called functions and see time and coverage statistics based only on the remaining called functions.
- Read - The number of bytes that the instructions from this function, class or file read from memory.
- Write - The number of bytes that the instructions from this function, class or file wrote to memory.
- Accessed - The total number of memory interactions caused by the reads and writes of this function, class or file.
- Size - The total size of the instructions contained in this function, class or file, measured in bytes.
- Stack - The number of bytes used by the stack in this function. A question mark appears next to the total here if the function's stack usage is unknown.
- Call Stack - The maximum number of bytes used by the stack in any call chain in which this function resides. For example, if a function is called in two separate call chains, one with a maximum stack depth of 512 and another with a maximum stack depth of 1024, the value 1024 appears in the **Call Stack** column. A question mark appears next to the total in the call stack field if it includes any unknown function stack usage.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Order - Considering a variety of factors, the ARM Profiler determines which function should be looked at first for stack depth optimization. Optimizing the function with 1 in the order column yields the greatest impact on your program's overall stack depth.
- Location - This column reports the exact location of the function or class, listing both the file name and the exact line of the declaration.

6.1.3 The totals panel

At the bottom of any table report is a panel that presents the totals of all the selected rows in various statistical fields. This panel can be useful to quickly see how big an impact a range of items has on the performance of your source code. Select a range of rows by holding down the shift or control key and the totals panel immediately updates to show you the totals based on the new selection. The totals panel can feature the following fields:

- Functions/Classes/Files/Call Chain Links - Although the unit depends on the table report type, this field lets you know how many rows are currently selected.
- Self Time - The total of the self time values from all of the selected rows next to the percentage that sum represents of the total time used by the application.
- Size - The total size, in bytes, of all the selected rows is listed next to the percentage this size represents of the combined total size of all functions.
- Avg CPI - This field reports the average CPI for all of the selected rows.
- Coverage - The percentage of the selected row's code that was covered. This number is not a mean value of all the listed percentages, but an aggregate based on the percentage of all lines of code from the selected functions. As such, functions with more instructions have a larger impact on the coverage percentage listed here.
- Delay - The delay in all of the selected rows. This field lists the total delay next to this value's percentage of the whole.
- Read - The total size of all of the read operations caused by the selected rows.
- Write - The total size of all of the write operations caused by the selected rows.
- Called - The number of times the call chain links in the selected rows were called. The 'Called' and 'Calls' fields only appear in the totals panel of a call chain report.
- Calls - The number of times the call chain links in the selected rows called another function.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

6.1.4 The statistical type drop-down menu

Located to the right of the export table icon in every table report's toolbar, the statistical type drop-down menu enables you to set the type of data used to populate the table. By default, a table report presents the aggregate for fields like self time, total time and delay, but this menu can be used to toggle between the following possible options:

- Totals - The default option, selecting **Totals** from the drop-down menu updates the table to show the aggregate values for all the pertinent fields. Some fields, like efficiency and average CPI have fixed value types and do not update to match the current selection. Average CPI always measures the average CPI per call, while efficiency always reports a percentage.
- Percentages - Changes all of the pertinent fields to reflect their value as a percentage of the whole. For example, select **Percentages** and the functions report updates the self time field to show the percentage of the total time spent executing a function's instructions. The Percentages option does not affect the values shown in the average CPI fields.
- Averages - Changes the statistical value to an average value per call for each of the functions or classes. As you would expect, this menu option is not available in the files and classes report types as files and classes are never themselves invoked. The efficiency column continues to report values in terms of percentages.

6.1.5 Selecting columns to display

By default, the report types display every available report field and the table scrolls from left to right if there is too much data to fit horizontally. If you want to remove any of the columns, right-click on the header row in any report type to bring up the column selection contextual menu, as shown in Figure 6-2 on page 6-7:

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

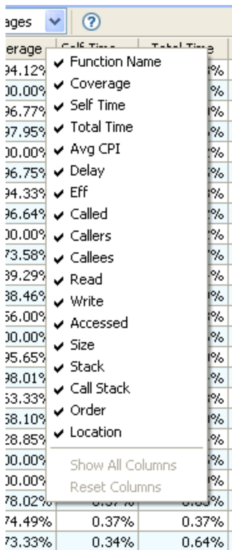


Figure 6-2 Choosing Report Columns

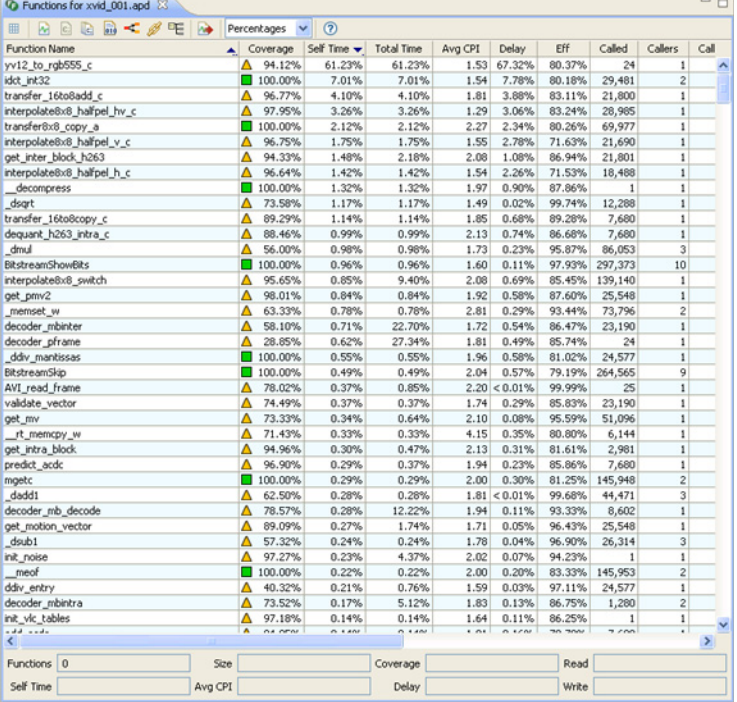
The ARM Profiler displays all of the checked report fields. To hide a report field and its associated column, select it in the contextual menu. To un-hide the column, re-open the contextual menu and select the report field again. Any number of columns can be hidden in this manner so that functions, classes, files and call chains reports display only the information that you find valuable.

6.1.6 Sorting data

The sort order varies from report to report, but it can be changed by clicking once on any of the column headers. The data in the table is reordered based on the data contained in that column. To reverse the sort order, click in the same column header again. The default numerical and alphabetical sorting behavior varies from column to column, but an upwards arrow in the column header always indicates an ascending sort, while a downward arrow indicates a descending sort.

You are not limited to a simple one-level sort, either. You can specify as many subordinate sort criteria as there are columns. To specify more levels in the sort hierarchy, hold down the shift key and click on other columns until you have achieved your desired sequence. Such a sort is best illustrated in an example. To first sort by code coverage and then by the function name, click twice on the **Coverage** column header, and then shift-click on the Function Name header. This updates the table's headers to resemble those in Figure 6-3 on page 6-8:

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential



Function Name	Coverage	Self Time	Total Time	Avg CPI	Delay	Eff	Called	Callers	Call
yv12_to_rgb555_c	94.12%	61.23%	61.23%	1.53	67.32%	80.37%	24	1	
idct_int32	100.00%	7.01%	7.01%	1.54	7.78%	80.18%	29,481	2	
transfer_16to8add_c	96.77%	4.10%	4.10%	1.81	3.88%	83.11%	21,800	1	
interpolate8x8_halfpel_hv_c	97.95%	3.26%	3.26%	1.29	3.06%	83.24%	28,985	1	
transfer8x8_copy_a	100.00%	2.12%	2.12%	2.27	2.34%	80.26%	69,977	1	
interpolate8x8_halfpel_v_c	96.75%	1.75%	1.75%	1.55	2.78%	71.63%	21,690	1	
get_inter_block_h263	94.33%	1.48%	2.18%	2.08	1.08%	86.94%	21,801	1	
interpolate8x8_halfpel_h_c	96.64%	1.42%	1.42%	1.54	2.26%	71.53%	18,488	1	
__decompress	100.00%	1.32%	1.32%	1.97	0.90%	87.86%	1	1	
_dsqrt	73.58%	1.17%	1.17%	1.49	0.02%	99.74%	12,288	1	
transfer_16to8copy_c	89.29%	1.14%	1.14%	1.85	0.68%	89.28%	7,680	1	
dequant_h263_intra_c	88.46%	0.99%	0.99%	2.13	0.74%	86.68%	7,680	1	
_dmul	56.00%	0.98%	0.98%	1.73	0.23%	95.87%	86,053	3	
BitstreamShowBits	100.00%	0.96%	0.96%	1.60	0.11%	97.93%	297,373	10	
interpolate8x8_switch	95.65%	0.85%	9.40%	2.08	0.69%	85.45%	139,140	1	
get_pmv2	98.01%	0.84%	0.84%	1.92	0.58%	87.60%	25,548	1	
_memset_w	63.33%	0.78%	0.78%	2.81	0.29%	93.44%	73,796	2	
decoder_mbinter	58.10%	0.71%	22.70%	1.72	0.54%	86.47%	23,190	1	
decoder_pframe	28.85%	0.62%	27.34%	1.81	0.49%	85.74%	24	1	
_ddiv_mantissas	100.00%	0.55%	0.55%	1.96	0.58%	81.02%	24,577	1	
BitstreamSkip	100.00%	0.49%	0.49%	2.04	0.57%	79.19%	264,565	9	
AVI_read_frame	78.02%	0.37%	0.85%	2.20	< 0.01%	99.99%	25	1	
validate_vector	74.49%	0.37%	0.37%	1.74	0.29%	85.83%	23,190	1	
get_mv	73.33%	0.34%	0.64%	2.10	0.08%	95.59%	51,096	1	
__rt_memcpy_w	71.43%	0.33%	0.33%	4.15	0.35%	80.80%	6,144	1	
get_intra_block	94.96%	0.30%	0.47%	2.13	0.31%	81.61%	2,981	1	
predict_acdc	96.90%	0.29%	0.37%	1.94	0.23%	85.86%	7,680	1	
mgetc	100.00%	0.29%	0.29%	2.00	0.30%	81.25%	145,948	2	
_dadd1	62.50%	0.28%	0.28%	1.81	< 0.01%	99.68%	44,471	3	
decoder_mb_decode	78.57%	0.28%	12.22%	1.94	0.11%	93.33%	8,602	1	
get_motion_vector	89.09%	0.27%	1.74%	1.71	0.05%	96.43%	25,548	1	
_dsub1	57.32%	0.24%	0.24%	1.78	0.04%	96.90%	26,314	3	
init_noise	97.27%	0.23%	4.37%	2.02	0.07%	94.23%	1	1	
__meof	100.00%	0.22%	0.22%	2.00	0.20%	83.33%	145,953	2	
ddiv_entry	40.32%	0.21%	0.76%	1.59	0.03%	97.11%	24,577	1	
decoder_mbintra	73.52%	0.17%	5.12%	1.83	0.13%	86.75%	1,280	2	
init_vlc_tables	97.18%	0.14%	0.14%	1.64	0.11%	86.25%	1	1	

Figure 6-3 A Multi-level Sort

The sort triangles show the direction of sort for each field, and the dots in the lower right of the column headers indicate ordering. In this case, 'Coverage' has one dot, indicating that it is the primary sort criteria, and 'Function Name' has two dots because it is the secondary sort criteria. There is no limit to the number of sort criteria you can specify.

If an element is selected in the table, the table scrolls to keep the selected element in view after a re-sort.

Note

Sorting in the call chains report works somewhat differently than in the other report types. You can still click various columns to add sort criteria and change the direction of the sorts, but only the order in which the children of a particular function appear are subject to the sort criteria.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

6.1.7 Exporting table data

If you want to save the data displayed in the functions report to disk in a format easily read by other applications, use the export data icon, located to the right of the navigation options in the toolbar. The export command can be used to create a text, tab-delimited, or comma-delimited file on disk which contains data from the selected table. The amount of data exported is up to you.

When you have selected the export command, the ARM Profiler presents you with an export table dialog similar to the one shown in Figure 6-4:

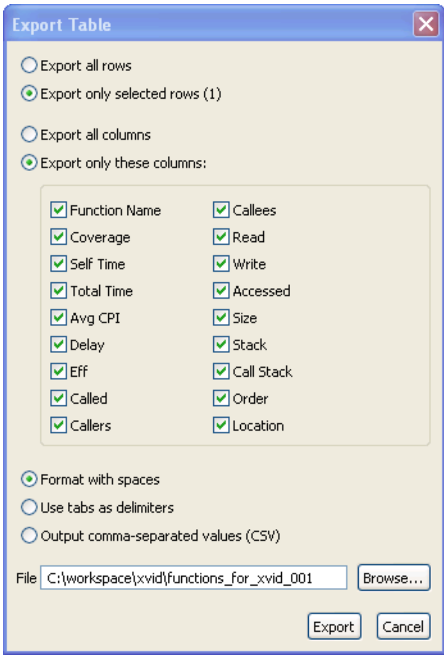


Figure 6-4 The Export File Dialog

If you have something currently selected in the table report, you can use the two radio buttons at the top of the export table dialog to set the row parameters of the exported data. Use the **Export all rows** button to export the entire contents of the table or choose the **Export only selected rows** button to export only the range of rows that are currently selected in the table. The number in parentheses next to the **Export only selected rows button** indicates the number of rows that are currently selected. If nothing is currently selected in the table report, these export row options do not appear.

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

When you have chosen the rows you want to export, you can use the next set of radio buttons to set the same options for columns. The first option exports all of the columns, while selecting the second option enables the column header check boxes to manually choose the columns you want to export.

The three radio buttons near the bottom of the export dialog enable you to save the exported data in different formats. Choose the **Format with spaces** option to export the resulting data file with spaces so that it is legible in a monospaced font. Selecting the **Use tabs as delimiters** button generates a row where the values are separated by tabs, making the file more easily interpreted by common spreadsheet applications. Choosing the final option, **Output comma-separated values**, outputs a file with the values separated by commas, another format easily understood by spreadsheet applications.

Enter a valid file location in the **To file** field. The ARM Profiler automatically assigns a file extension based on the output type chosen using the above buttons. The 'Format with spaces' option yields a .txt file, a tab-delimited file is saved as a .tab file, and the comma-separated values option produces a .csv file.

When you have chosen all of your preferred options and entered a file name, click the **Export** button. A new file appears in the specified location.

6.1.8 The outline view

In the default layout of the ARM Profiler perspective, the outline view appears on the far right side of the ARM Workbench window. The outline view can be opened and closed independently via the **Window** → **Show View** menu option. A sample outline view is shown in Figure 6-5 on page 6-11.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

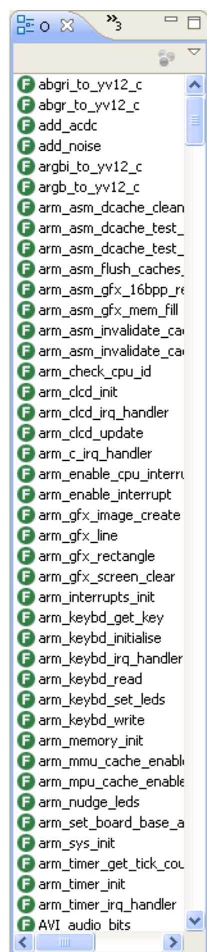


Figure 6-5 The outline view

When any report besides the summary report is active, the outline view contains an alphabetical list of every function, class or file, depending on the active report type.

You can use the outline view to easily find a specific function without changing the sort options you currently have set in the data table. For example, if you want to know where a particular function is in relation to call count, but do not want to re-sort by name in the report, use the alphabetically sorted outline view to find the function you are looking for. Clicking on a function in the outline view brings the function to view in the report and highlights it.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

The outline view in the call chains report lists every function in alphabetical order, but here the single function icon from the functions report has been replaced by the multiple function icon. Clicking on any function in the outline view while the call chain report is active in the ARM Workbench editor selects every call chain link of the selected function and expands the necessary functions to show them. The number in parentheses next to the function name indicates the number of call chain links where the function is present.

Just like in the table reports themselves, you can right-click on any item in the outline view and navigate to other report types using the available contextual menu options:

- Select in Functions
- Select in Files
- Select in Code View
- Select in Call Graph
- Select in Call Summary

For more information on how to use these contextual menu options to navigate to other report types, see *Contextual menus* on page 6-13.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

6.2 Navigating to other reports

The ARM Profiler provides two ways to access other report types from within a table report. You can open a new report using that report's corresponding icon in the toolbar, or open a contextual menu by right-clicking on a row in the table and selecting one of the report types. In both cases, context is important. The newly opened report selects and highlights data based on which rows are selected in the table report.

6.2.1 Contextual menus

To open a contextual menu, right-click on a row in any of the table reports. Although the exact list of options is dependent on the current selection and table report type, the table reports' contextual menu options include:

- Filter - This menu option filters the current selection's data from all reports. To remove a filter, right-click on and select the **Remove Filter** option from the contextual menu. All filtered functions are grayed out.
- Filter, Including Children - This option filters the statistics of the current selection and the statistics of all of its children from all reports. In the call chains report, the filtered call chain links are grayed out. To remove this filter, right-click on any filtered function or call chain link and select the **Remove Filter, Including Children** contextual menu option. This option is disabled in the file report and in cases where the selection does not contain any children.
- Filter Others - This menu option is the inverse of the **Filter** menu option. Instead of filtering the selection's data from all report types, it filters out everything but the selected items.
- Filter Others, Not Including Children - This options works similarly to the **Filter Others**, menu option, only it filters out everything not currently selected and the children of the unselected functions. This option is disabled in the file report and in cases where the selection does not contain any children.
- Select in Functions - This menu option opens the functions report with the function or functions related to the current selection highlighted. For example, if you have two files selected in the file report and press the function report icon in the toolbar, the functions report opens with all of the functions contained in both of those files selected. This contextual menu item is not available from within the functions report itself.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Select in Files - This option opens the files report with the file that contains the chosen function or class highlighted. The location column header in the functions, classes, and call chains table reports lets you know which file contains the function or class, but the files report provides detailed coverage and timing statistics for the file as a whole.
- Select in Classes - From the functions report, this option opens the classes report with the chosen function's class selected. From the files report, this option opens the classes report with all of that file's classes selected. This option is only available if C++ code is present.
- Select in Code View - This option opens the code view for the selected function, file, or class. The code view report gives insight into your image's performance broken down by individual instruction in the assembly code and source line in the original C or C++ source. In this way, you can take a deeper look at the performance of a critical section of code.
- Select in Call Graph - This option opens the call graph with the functions associated with the current selection highlighted. The call graph shows visually where the function fits in the grand scheme of your code and is color coded based on timing or code coverage.
- Select in Call Chains - This option opens the call chain report with every instance of the chosen function selected in the hierarchical call chain report. All hierarchical call chain links leading to the selected function are disclosed to ensure every instance appears in a call chain report opened in this manner. This contextual menu option is not available from the call chain report itself and if the selected class, file or function was not in the execution captured by the current analysis file, the **Select in Call Chains** option is not available.
- Select in Call Summary - This contextual menu option opens the call summary with the chosen function as its primary function. The call summary lets you filter out calling and called functions to narrow the scope of the timing and coverage data.
- Edit Source - The final option in the contextual menu is **Edit Source**, a menu option that opens the source file that contains the selected item. This menu item is disabled if there is no selection.

6.2.2 Toolbar navigation options

On the left side of the toolbar of every one of the report types, there is a group of icons, each representing a different report type that enables you to open a new report. The buttons are context sensitive and the selections in the new report are dependent on the selections in the report in which you chose the toolbar command. With no rows selected

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

in the currently active report, the **Edit Source** option is disabled. To open the source code based on the context of the current selection, you must first select one or more items.

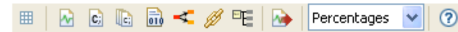


Figure 6-6 A Functions Report's Toolbar Navigation

Just like the context sensitive menu options, the menu buttons are dependent on the currently open report. For example, the functions report navigation option does not appear if that is the report you currently have open.

Each report type is represented by its corresponding icon, listed in the order that they appear:

- - The Analysis Summary
- - The source file
- - The functions report
- - The classes report
- - The files report
- - The code view
- - The call graph
- - The call chains report
- - The call summary

Selecting any one of these options has the same effect as choosing the contextual menu option that matches the report type. For a description of each of these navigation options, see *Contextual menus* on page 6-13.

Note

The classes report icon only appears in a toolbar if the associated source file contains C++. If the code you are using is written in C, the classes report icon does not appear in the toolbar at all.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

In addition to the report type buttons, the toolbar also contains an export table button and a help button. The export table button opens the export dialog while the help button opens the help view and displays a list of topics specific to the currently active table report. Using this toolbar button has the same effect as pressing F1 in Windows or Shift + F1 in Red Hat Linux.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

6.3 The functions report

The functions report shows every function defined in your source files in addition to the following performance statistics:

- Coverage
- Self Time
- Total Time
- Avg CPI (average cycles per instruction)
- Delay
- Eff (efficiency)
- Called
- Callers
- Callees
- Read
- Write
- Accessed
- Size
- Stack
- Call Stack
- Order
- Location

For a description of each of these column headers, see *Table report column headers* on page 6-3.

Double-clicking on any item in the functions report opens the code view with every line of source and every disassembly instruction pertinent to the function selected.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

6.4 The classes report

The ARM Profiler provides the following statistical categories in class reports for C++ analysis files only.

- Coverage
- Self Time
- Delay
- Avg CPI
- Functions
- Read
- Write
- Accesses
- Size

For a description of each of these column headers, see *Table report column headers* on page 6-3.

Double-clicking on any item in the classes report opens the functions report, with each of the functions contained in the class selected.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

6.5 The files report

The files report breaks data down by the source files used to build the image file. Compiler and language libraries and other instructions with no direct or obvious link to source code are included in this report under the name of the binary image they came from. It contains the following column headers:

- Coverage
- Self Time
- Delay
- Avg CPI
- Functions
- Read
- Write
- Accesses
- Size

For a description of each of these column headers, see *Table report column headers* on page 6-3

Double-clicking on a file in this table report opens the code view for the file, with all of the pertinent code lines and disassembly instructions selected.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

6.6 The call chains report

The call chains report is laid out in a table format like the other reports discussed in this chapter, but here the data is presented hierarchically. The call chains report is intended to show you the exact code hierarchy as captured during execution. If a function is called in multiple places in the hierarchy, each instance appears as an individual ’call chain link' in the call chains report. The call chains report's set of column headers is similar to those in the functions report:

- Total Time
- Self Time
- Called
- Calls
- Stack
- Location

For a description of each of these column headers, see *Table report column headers* on page 6-3.

In the call chains report, the data is broken down by instance, so that you can get a clear view of how different functions performed at every place they occur in the call chain.

Double-clicking on a row in the call chains report opens up that function in the call summary, a graphical view of a function's callee and caller relationships, overlaid with statistics.

6.6.1 Navigating the call chains report's hierarchical table

The data in the call chains report is categorized hierarchically, meaning that, to see a function's subordinate functions, you have to expand it by clicking on the disclosure controls to the left of the function name, or pressing the right arrow key on your keyboard. To hide a link's subordinate functions, click the disclosure button again or press the left arrow key. This functionality is shown in Figure 6-7.

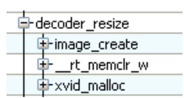


Figure 6-7 The hierarchical call chains display

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

If you take a look at the call chain shown in Figure 6-7 on page 6-20, you can see that the link `decoder_resize` has been expanded to show three of its child functions. Exploring the links in the call chains report in this manner is a good way to pinpoint exactly where the time is spent in your code.

Note

Due to memory constraints, the call chain's maximum depth is 511. If a call chain exceeds this depth, it is trimmed to keep within this number and the ARM Profiler adds a note to the affected call chain link.

The call chains report's toolbar contains all of the standard table report buttons and six other buttons, not found in the other report types. The button, located just to the right of the standard toolbar options, hides all the children of all links, while the button to its right, expands it to show every subordinate branch of all links. The filtering buttons are located to the right of these disclosure control buttons. For more information on their functionality, see *Call chains report filtering* on page 6-22. To the right of the filtering buttons are the call chain link note buttons. For more information on using the next and previous call chain link buttons, see *Call chains link notes*.

The call chains report also contains two unique contextual menu item, **Expand Selection To All Matching Call Chain Links** and **Collapse Unselected Call Chain Links**. Use the **Expand Selection To All Matching Call Chain Links** option, only present if the function appears in the call chains report more than once, to select all of this function's call chain links. The hierarchical table expands to show each of the newly selected links. The **Collapse Unselected Call Chain Links** option collapses every unselected call link subordinate to the selected function.

6.6.2 Call chains link notes

In certain cases, a note icon appears to the right of the link in the **Function Name** field. Hover over the note and a tooltip appears that lists every special consideration that is applicable to this link. Here is a list of all of the possible notes that can be found in a call chains report:

- The point at which execution for this profiling run began
- The point at which execution for this profiling run ended
- Function call data was merged into the ancestor link of this function due to unbounded recursion at this point
- Function returned to an unexpected point
- Execution resumed here after a return from an unexpected point

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Synchronization was lost at this point during the run, due to trace or ETM overflows.
- Thread Call Chain Root: *Name of Thread*
- Exception Call Chain Root: *Name of Exception*

The call chains report includes two buttons in its toolbar to help you navigate to each call chain link with an attached note. Use the button to go to the next call chain link with an attached note and the button to go to the previous one. This selects the annotated call chain link and expands the call chain to the level necessary to expose it.

6.6.3 Call chains report filtering

The call chains report includes filtering buttons in its toolbar that are not available in the other table report types. Use the to filter the currently selected call chain links and all of their children. The button removes filtering for the selected links and all of their children. Filtering removes the data from every report type.

The call chain report contextual menu contains additional filtering options, including **Filter**, **Filter, Including Children**, **Filter Others**, and **Filter Others, Not Including Children**. For information on the functionality of these menu options, see *Contextual menus* on page 6-13.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 7
The Code and Replay Views

This chapter gives you an in-depth look at the code view. It includes an overview, a description of how to navigate to the code view, a detailed look at the source panel and a description of the data provided by the disassembly and replay views.

It contains the following sections:

- *Overview* on page 7-2
- *Navigating to the code view* on page 7-3
- *Basic code view functionality* on page 7-5
- *The source panel* on page 7-7
- *The disassembly panel* on page 7-11
- *The replay view* on page 7-15.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.1 Overview

The code view enables you to analyze functions line-by-line to see where you can optimize your code to improve the way it interacts with hardware. Scan your source code to see how each line performed in terms of code coverage, execution counts, and time, and then look to the disassembly panel to see how each instruction performed against expectations.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.2 Navigating to the code view

To open the code view, click on the **Code View** link in the overall analysis section of the summary report. This opens the code view for the first function in the first call chain:

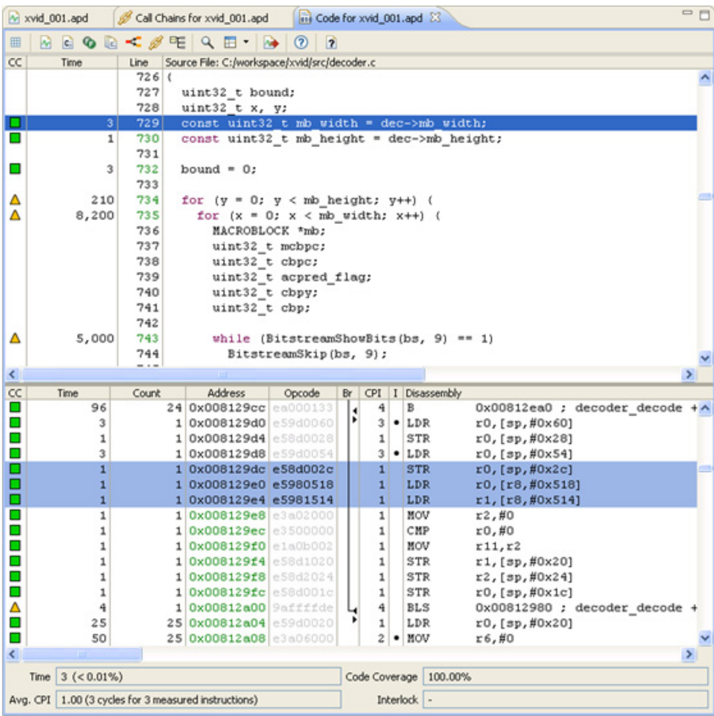


Figure 7-1 The Code View

To select the code for a specific function, you can use the **Select in Code View** contextual menu option, available in most instances by right-clicking on items in all of the ARM Profiler reports.

7.2.1 Navigating to the code view using the bar charts

To gain quick access to a code view report for one of the top five functions presented in the summary report's bar charts, double-click on any of the bars or bar titles. All of the chosen function's pertinent lines of code are highlighted in the code view as well as each line of its associated disassembly. Right-clicking a bar or bar title to open a contextual menu and choosing the **Select in Code View** option from the list has the same effect.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.2.2 Navigating to the code view from the other reports

While the analysis summary allows you quick access to the code view via the link in the overall analysis section and the bar charts' contextual menus, you can also access the code view for a specific function or class from within the following report types:

- The functions report
- The files report
- The classes report
- The call graph
- The call chains report
- The call summary

Open the code view from the functions, classes and call chains reports by right-clicking on a row and choosing the **Select in Code View** navigation option from the resulting contextual menu. Navigation from the call graph and call summary work in much the same way. Right-click on a function icon in either of these graphs and choose the **Select in Code View** menu option.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.3 Basic code view functionality

Although the two primary sections of the code view show different versions of the code alongside a different set of profiling data, these panels do possess some commonalities. This section explores features that are common to all of the panels of the code view.

7.3.1 Selection behavior

Selecting code in either the source or disassembly panels synchronizes both panels and highlights related code. This feature ignores coding comments.

Double-clicking an instruction in the disassembly panel selects all of the instructions that relate to a single line of source code, and double-clicking on a function label in the disassembly panel selects all of the instructions that make up that function. Both are excellent ways to select a series of related instructions.

To select multiple rows, hold down the mouse button and drag it across a range of rows. Selection behavior available in other applications is also present here. Hold down the shift key and select the first and last row of the series to select the entire sequence of rows. Hold down the control key if you would like to select additional rows without selecting all of the rows in between.

If selected source lines or disassembly instructions contain too many rows to fit in the bounds of the ARM Workbench editor, small selection indicators appear on the right hand side of the code view. If there are more selected rows than can fit in the view, the indicators show you how many more are present off screen.

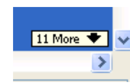


Figure 7-2 A Selection Indicator

Scroll up or down to see all of the selected rows.

7.3.2 Adjusting views

By default, the source panel and the disassembly panel each take up half the space of the ARM Workbench editor, but you can adjust this proportion by clicking and dragging the line that divides the two panels.

To see either the source or the disassembly panel exclusively, use the button, located in the menu bar of the code view By default, this is set to **Source & Disassembly**, but you can press the button to cycle to the **Source Only** or **Disassembly Only** to see one of the views exclusively. Use the small arrow to the right of the button to open a drop-down menu and select any of these options directly.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.3.3 The find command

To find a specific function in your code, press the magnifying glass icon in the toolbar or press **Ctrl+F**. A new **Find** field appears in the totals panel that enables you to search your code and instructions for a function name or a hexadecimal instruction address. Enter a string and the field on the right hand side updates to show the current match, if there is one. Pressing the enter key takes you to the first match in the code and subsequent presses of the enter key cycles you through all of the available matches. Hit the icon again or press **Ctrl+F** to remove the field.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.4 The source panel

The top section of the code view is the source panel, where you can see your original source code next to line-by-line profiling data:

Figure 7-3 The Source Panel

All of the pertinent lines of the function used to navigate to the code view are highlighted in the source panel when the code view is opened. Comment lines and lines of code that are not directly linked to assembly code instructions, such as variable declarations, are not selected and have no profiling data associated with them.

7.4.1 Locating source files

If you have not moved your source files used in the creation of the current analysis file, the ARM Profiler automatically locates and displays the source code in the source panel. If, however, the source files are not located in the same directory location they were in during compilation, the source code view is not populated. Instead, an empty source panel in the code view tells the user that the location of the source file is missing, as shown in Figure 7-4 on page 7-8:

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

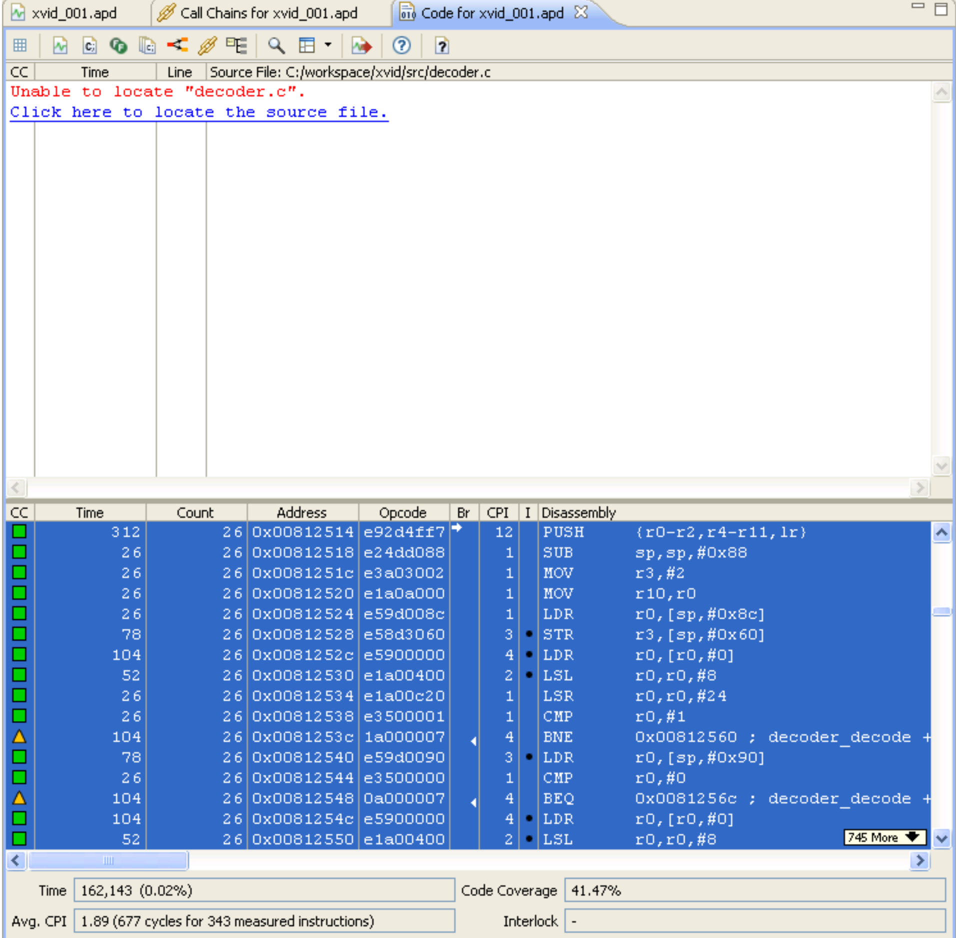


Figure 7-4 An Empty Source Panel

To populate the source panel, you must locate the exact version of the source file used to create the analysis report. Clicking the link in the source panel opens a file navigation dialog like the one shown in Figure 7-5 on page 7-9:

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

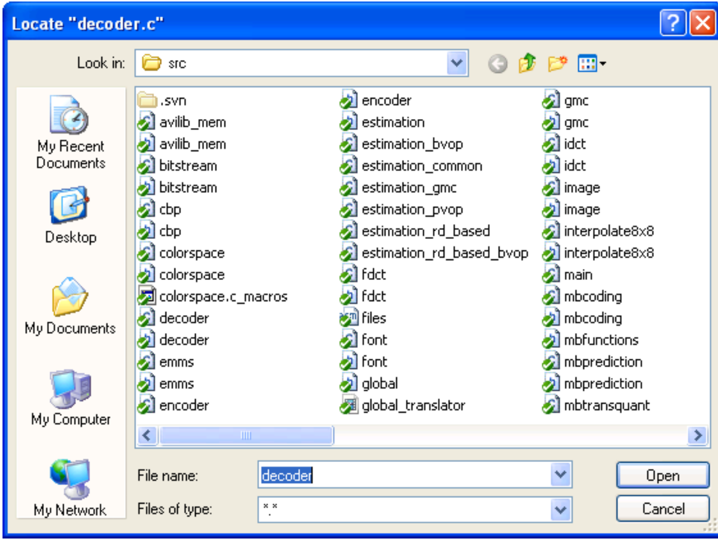


Figure 7-5 The Locate Source Dialog

The locate source dialog is a standard file navigation window, and varies depending on your operating system. Locate the source file, select it and press the **Open** button. When you return to the code view, the source panel has now populated the source code and profiling data columns with data.

7.4.2 Source panel column headers

In addition to providing a look at your original source code, the source panel also gives you line-by-line statistics for all pertinent lines of code. Each of these statistical categories are explained here:

- CC - CC stands for code coverage and the color and shape of the icon indicates whether the code was covered during the run captured in this analysis file. A green box indicates the code was 100% covered, a yellow triangle means partial coverage, while a red circle means the line of code was not executed at all during the run.
- Time - Time is shown in the second column of the source panel and its displayed unit of measurement depends on the method used to gather profiling data. If the code was tested against actual hardware with the help of RealView ICE and RealView Trace 2, the data presented here is measured in samples. Depending on the user-defined sampling rate, RealView Trace 2 captures which instruction is being executed periodically and records a sample for that instruction, thus giving

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

you a general idea of where your code is spending most of its time. If, however, you generated profiling data on a host machine running a Real-Time System Model, the code view reports the time in estimated cycles.

- Line - The line number for every line of source code in the file is reported here. If the ARM Profiler inlined the instruction due to the optimization level, the number will appear in green. Hover over the number and a tooltip appears that lets you know to what location the line of code was inlined.
- Source - This column of the source panel displays the actual source code, while the column header shows the source file's name and directory location.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.5 The disassembly panel

The disassembly panel shows you the disassembly code from the image file used in the profiling run next to statistics relevant to that instruction. A sample disassembly panel is shown in Figure 7-6:

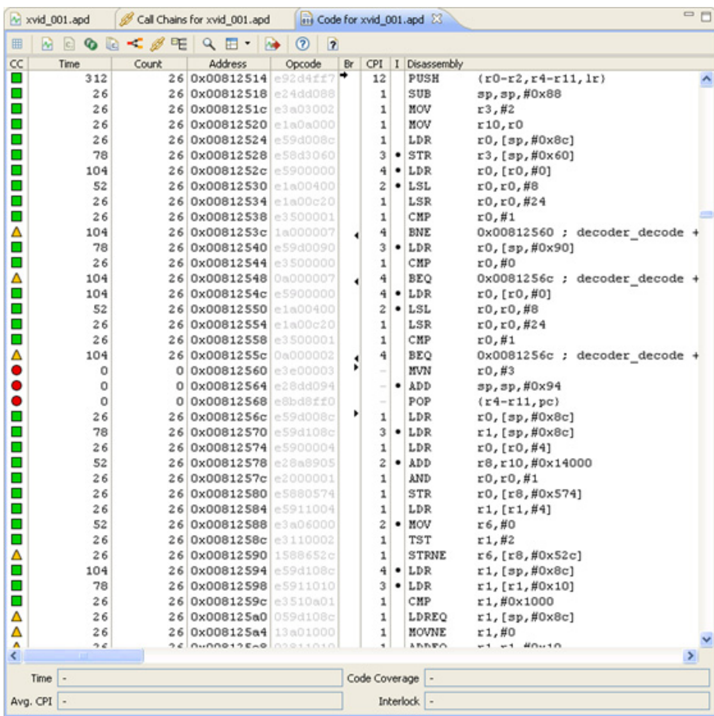


Figure 7-6 The Disassembly Panel

You can use the data presented here to see which instructions are taking more time than you expected due to an interlock and see the cycle per instruction efficiency of each line of disassembly code.

 Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.5.1 Disassembly panel column headers

This section provides an explanation of each of the column headers present in the disassembly panel.

- CC - Just like in the source panel, CC is shorthand for code coverage. A green box indicates the instruction was executed during the profiling, a red circles means it was not, while a yellow triangle indicates it was only partially executed. An instruction is said to be partially covered when it is conditionally executed and only one of the conditions is ever executed. For instance, an ADDEQ instruction is partially covered when the equality condition is always true or always false.
- Time - Time, as shown in the ARM Profiler, is a relative measure and is not measured in actual execution time. The displayed unit of measure depends on the method used to gather profiling data. For more information on time, samples and estimated cycles, see *Chapter 5 The Table Reports: Function, File, Class, and Call Chain.*
- Count - The number of times the instruction was executed.
- Address - The memory address where this instruction is located. This number appears in green if the instruction was inlined due to code optimization. Inlining is most likely to occur when using the highest level of compiler optimizations (-o3). You can easily identify the instructions that were inlined as a result of this process by their green addresses in the Address column. Hover over the address column to get a tooltip, or scroll to the Disassembly column for details about the function's inline destination.
- Opcode - The machine code instruction as it is executed by the microprocessor.
- Branches - The branches column shows you where the instruction was called from and shows all its calling locations. It is essentially split into two halves. The left side of the branches column shows every instruction that calls or is called by another function. What follows is a brief description of each of the arrow types that appear in the left half of this column:
  - ➜ - This arrow appears if the instruction is called by an instruction contained in another function. Hover over the arrow to show a tooltip that lists each of the calling instructions and the call percentage of each. Right-click on the instruction to open a contextual menu that contains the same information. Each calling location here is a menu option that takes you to the calling instruction in the disassembly panel.
  - ← - This arrow appears if the instruction calls an instruction contained in another function. The tooltip and contextual menu behavior is the same as the left arrow, only they list the call destinations for the instruction.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

The right hand side of the column shows branching behavior within the function. Here, too, there are two types of arrows that can appear:

— ◂ - This arrow shows that the instruction performs one or more local branches within the function. To see which instructions a branch comes from or leads to, hover the mouse over the small arrow icon in the right side of the column and a connecting line appears. The line stays after you have moved the cursor off the icon, so that you can follow its path to its connecting locations. Hovering over a branch arrow produces a tooltip that displays each of the local and cross-function calls to and from this line. To select any of these call destinations, right click on the branch item and choose the instruction you want from the resulting contextual menu.

— ▸ - This arrow shows that the instruction is a local branch destination. Its functionality is identical to the ◂ arrow, only the tooltip and contextual menu lists the instructions from which it branches.

It is important to note that a single instruction can be both a branching instruction and a branching destination, and, if this is true, both arrows show up in the right hand side of the branches column.

- CPI - CPI stands for cycles per instruction and it measures how many cycles, on average, it took the instruction to execute. If data was collected using hardware, the CPI column only reports a value if the sample rate for an instruction is less than the amount of pass/fails. In cases where insufficient executions were reported to make the CPI meaningful, this column shows a '-'. This number is highlighted in pink if the CPI value is higher than expected. An instruction's expected execution time is observed under ideal conditions. This assumes the instruction opcode is located in L1 cache and any data accessed by the instruction is also fetched directly from the cache.

- Interlock - A mark in the interlock column indicates the associated instruction caused an interlock because of resource contention in the processor's pipeline. Hovering over the Interlock bullet produces a tooltip that tells you which register caused the interlock and highlights the references to the register that led to the interlock in the Disassembly column.

- Disassembly - The disassembly column shows you the opcode mnemonic side-by-side with the registers and any immediates used in the instruction. If the instruction calls another function, the function name is given here along with its hexadecimal address. Calls within a function reports their target address along with a function name and offset. For example, `ICON SUBNE pc,r7,r3` indicates that the subtraction never took place because the zero flag was not set.

- File - Gives the file name and row location of the code that contains this instruction.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.5.2 Code view totals panel

The totals panel in the code view is similar to the totals panel you find in the various table reports in that it shows you the accumulated data for a range of selected rows. It gives you aggregate values for time, average CPI and code coverage. The code view totals panel fields are:

- Time - The total amount of time used by the selected instructions.
- Code Coverage - The Code Coverage field shows you the percentage of the selected code that was executed during the run captured by the current analysis file. If a single conditional instruction is selected, a parenthetical is listed after the code coverage number, telling you exactly how many times the instructions reported a pass versus how many times it reported a failure (pass/fail).
- Avg. CPI - The average of all the CPI values for the selected instructions. The number in parentheses next to the average CPI value is the expected average CPI value, based on instruction type. If you accumulate your data using a Real-Time System Model, the delta between CPI and expected CPI is based solely on the detected interlocks. If data was collected using RealView Trace 2, the CPI values are determined by multiplying the samples by the sampling rate and dividing by the instruction count. Disparity between the CPI and expected CPI, when based on actual trace data, can have a number of causes, including accesses to slow external memory or cache effects.
- Interlock - This field lists the register that caused the interlock by the selected instruction. Nothing is displayed if multiple rows are selected.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.6 The replay view

The replay view is only available if you activated the **Collect Instruction Trace Replay** checkbox for the captured analysis file. For more information on how to do this, see *Enabling instruction trace replay* on page 3-14.

If you turn trace data collection on, the ARM Profiler produces a second file in addition to the normal `.apd` file. The trace replay view is pictured in Figure 7-7.

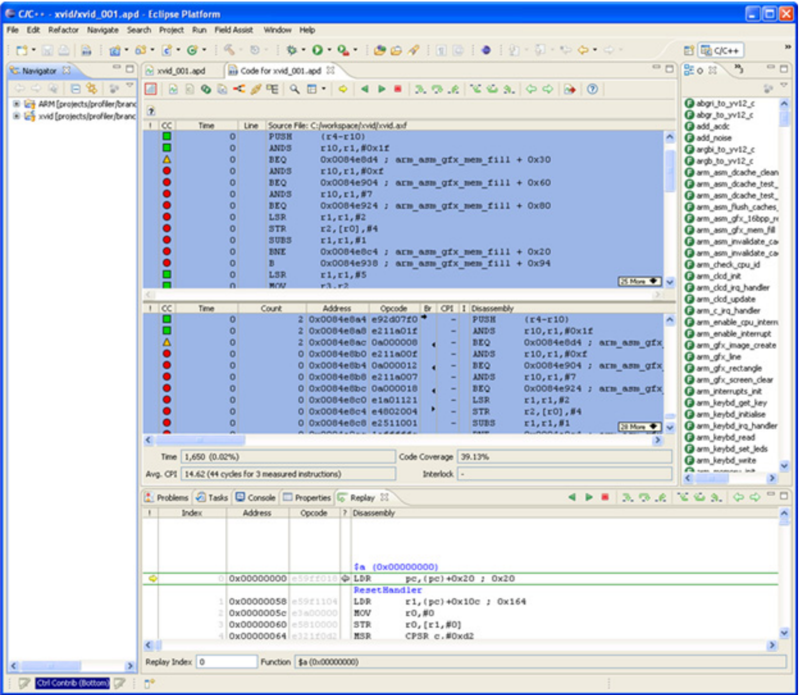


Figure 7-7 The replay view below the code view

Enabling trace replay collection adds the replay view, which appears, by default, in the bottom section of the workbench. In addition to adding the replay view, there are important differences in the code view when trace replay collection has been turned on. Trace-specific navigation and trace replay options enable you to:

- display the current breakpoint in all of the open views
- step, step over, or step out forward and backward through the trace
- set and move through breakpoints
- jump between instances of the selected instruction.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

The yellow trace marker, also a feature specific to trace-enabled analysis reports, marks the current trace position in all three views.

7.6.1 Replay view basics

While the disassembly panel provides an overview of each instruction's performance by providing execution counts and timing statistics, the replay view is a sequential list of all executed instructions. Here, every instance of every instruction is listed in the order of execution. If a loop is called 100,000 times, the instructions that make up that loop will appear 100,000 times in the replay view.

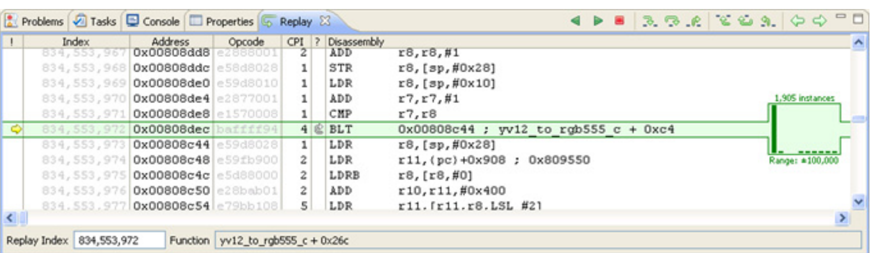


Figure 7-8 The replay view with a cycle accurate capture

The centered light green highlight and the yellow arrow show the current trace position. The green box on the right of the highlighted instruction is a CPI histogram and it only appears in the replay view if the sampling rate was set to cycle accurate for the captured execution. The small graph shows the incidence rates of CPI values caused by this instruction and any instances of the instructions within a range of 100,000 instructions above or below it. For example, if there are 10 instances of a given instruction within the +/- 100,000 range of the current trace position and 5 had a CPI value between 0 and 9 and 5 with a value between 41 and 50 due to cache effects, two equal bars would appear in this chart, one on the far left that represents the five values between 0 and 9 and one in the middle that represents the values between 41 and 50. No other bars would appear in the chart, as no other CPI values exist in the given +- 100, 000 range. Click on any bar for a tooltip that tells you the CPI range the bar represents and how many instances of the instruction fall in that range. Hold the mouse button down and drag across all bars to see information on each.

7.6.2 Replay view column headers

This section provides an explanation of each of the column headers present in the replay view.

- ! - Displays the ⇨ trace position and ● breakpoint markers, if applicable.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Note

This column is added to the source and disassembly panel when instruction trace replay data is present, so that the current trace position and break points are also visible in these views.

- Index - Shows a sequential label for every instruction.
- Address - The memory address where the instruction is located.
- Opcode - The machine code instruction as it is executed by the microprocessor.
- ? - Displays various informative symbols about the instruction. The list of possible symbols include:
  - — - Appears if this instruction calls a new function.
  - — - Appears if instruction branches to an instruction at a lower memory address. The arrow indicates an above position because instructions with a lower memory address value will appear higher in the disassembly panel.
  - — - Appears if instruction branches to an instruction at a higher memory address.
  - — - Appears if the instruction performs no actions because the conditions that trigger the instruction's actions were not met or if the processor partially executed the instruction, then cancelled its execution.
- Disassembly - Lists the opcode mnemonic with the any registers and immediates used in the instruction. There is no difference between this column and **Disassembly** column in the disassembly panel.

Replay view totals panel

The replay view contains the following fields:

- Replay Index - Displays the sequential index number of the trace position. Enter a value in this field and the trace position will jump to the instruction with the corresponding index number.
- Replay Time - If the data captured in the analysis file is cycle accurate, the Time field in the totals panel shows how many cycles the current instruction used.
- Function - Displays the current instruction's function name and offset within that function.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

7.6.3 Replay view menu options

In addition to the replay view itself, the addition of trace collection adds a number of new menu options to help you navigate to the data that interests you. These options appear atop the replay view and the code view, if you enabled the **Collect Instruction Trace Replay** checkbox.

Figure 7-9 Trace-specific menu options

The trace replay menubar options like **Run**, **Run Backward**, **Stop**, and the step options work like common debugger options in that they allow you to run your code to user-defined breakpoints and step through instructions in a variety of ways. The ARM Profiler is not a debugger, however, and the **Run Forward** and **Run Backward** options do not trigger any actual execution of code. They enable you to cycle through recorded trace data to get to areas of interest. This guide refers to these debugger-like tools as trace replay options.

The menubar includes the following trace specific menu options:

- - Sets all views so that the current trace position is highlighted and visible. In the source panel, this option will highlight the line of code associated with the current instruction. This option is specific to the code view. It does not appear in the replay view.
- & - Cycles backward and forward through the instructions until it hits either a breakpoint or the first/last instruction in the trace. You can use the stop button during this process to halt the search.
- - Stops the current progress of the two previous menu options, **Run Forward** and **Run Backward**. The trace indicator will halt at its current location.
- & - Steps forward and backward through your code. Clicking this button moves the current position in the replay view up or down one instruction and updates the source and disassembly panels accordingly. The order of execution is followed exactly.
- & - Steps over forward and backward. The step over trace playback options work similarly to the step forward and backward menu options, only they will skip child functions when they are called and move the trace position to the next function call at the same or higher level as the current function. In this manner it steps over functions lower in the hierarchy.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- & - Steps out forward and backward. The Step Out Forward and Step Out Backward commands enable you to travel up in the call chain hierarchy. Rather than preceding to the next or previous instructions in the trace, these commands will take you to the next or previous function that is above it in the call hierarchy.
- & - Jumps the trace position to the next or previous instance of the currently active instruction in the trace. If there are no further instances, they will take you to the beginning or the end of the trace, depending on whether you selected forward or backward. For example, if you have program loop that has executed 100 times, there will be 100 instances of the current instruction. The jump buttons will take you through all the loop positions one at a time. While the trace index box changes value, the trace window itself does not appear to move because the instructions surrounding current trace position are the same. These jump options are purely for navigation and do not honor breakpoints.

7.6.4 Replay view contextual menu options

The enabling of instruction trace replay collection also affects the contextual menu commands. In all three views - source, disassembly, and trace - you now have the ability to set and remove breakpoints and jump forward and backward between instruction instances.

- Set Breakpoint - Sets a breakpoint at the current location. The Run Forward and Run Backward commands can then be used to move the trace position between breakpoints.
- Remove Breakpoint - If a breakpoint is currently set in the selected code, this menu option removes it.
- Disable Breakpoint - Disabling a breakpoint causes the ARM Profiler to ignore it without actually deleting it. This makes it easy to toggle a breakpoint on and off without having to find the location again.
- Run Forward to Selection - Runs forward to the next instance of the currently selected instructions, the next breakpoint, or the end of the program trace.
- Run Backward to Selection - Runs backward to the previous instance of the currently selected instructions, the previous breakpoint or the beginning of the program trace.
- Run Forward off Selection - Runs forward to the next unselected instruction, the next breakpoint, or the end of the program trace.
- Run Backward off Selection - Runs backward to the previous unselected instruction, the previous breakpoint, or the beginning of the program trace.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Jump to Previous Instance in Replay - Has the same functionality as the ⇦ menubar button. It moves the trace indicator to the previous instance of the current instruction, ignoring breakpoints.
- Jump to Next Instance in Replay - Has the same functionality as the ⇨ menubar button. It moves the indicator to the next instance of the instruction, ignoring breakpoints.

7.6.5 Breakpoint view

The breakpoint view lists all of the breakpoints that are currently set. If it is not already visible, follow these steps to open the breakpoint view:

1. Select **Window** → **Show View** → **Other**
2. Show the ARM Profiler views using the disclosure control.
3. Select **Breakpoints** and press the **OK** button.

Figure 7-10 The breakpoints view

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Double-click on any of the breakpoints to jump to that breakpoint in the source and disassembly panel. The yellow trace pointer marks the current location in the trace.

7.6.6 Stack view

The stack view lists the call chain hierarchy for the current trace location. The current function is listed first followed by the function that called it, and so on up the hierarchy to the originating function. Each function's stack depth value is listed in a column to the right of it's name.

To open the stack view, follow these instructions:

1. Select **Window** → **Show View** → **Other**
2. Show the ARM Profiler views using the disclosure control.
3. Select **Stack** and press the **OK** button.

Figure 7-11 The stack view

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Note

The stack view is related to the current trace position, not the current selection in the various report types. Selecting an item in any of the report types has no affect on the stack view. Move the trace location using the replay menu commands or keyboard shortcuts to update the stack window.

To open the call chains report with the chosen call chain link selected, double-click on a function in the stack view.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 8
The Call Graph

This chapter explores the call graph and the information it provides. It includes a general overview and sections on how to open a call graph, how the call graph is laid out, selection, color-coding, navigating using the mini-map, and contextual menu options.

It contains the following sections:

- *Overview* on page 8-2
- *Opening a call graph* on page 8-3
- *Call graph layout* on page 8-4
- *The mini-map* on page 8-6
- *Color coding* on page 8-7
- *Selection behavior* on page 8-8
- *Contextual menu options* on page 8-9
- *The toolbar* on page 8-12
- *The outline view* on page 8-14.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.1 Overview

While the table reports lay out every detail of your functions' performance, the call graph's purpose is to show you visually which functions call what and where the hot spots are in the hierarchy. The call graph gives you a visual representation of each functions' timing performance. To find the function that caused the biggest performance bottleneck, hold down the spacebar and drag the screen to the function box colored the darkest shade of red.

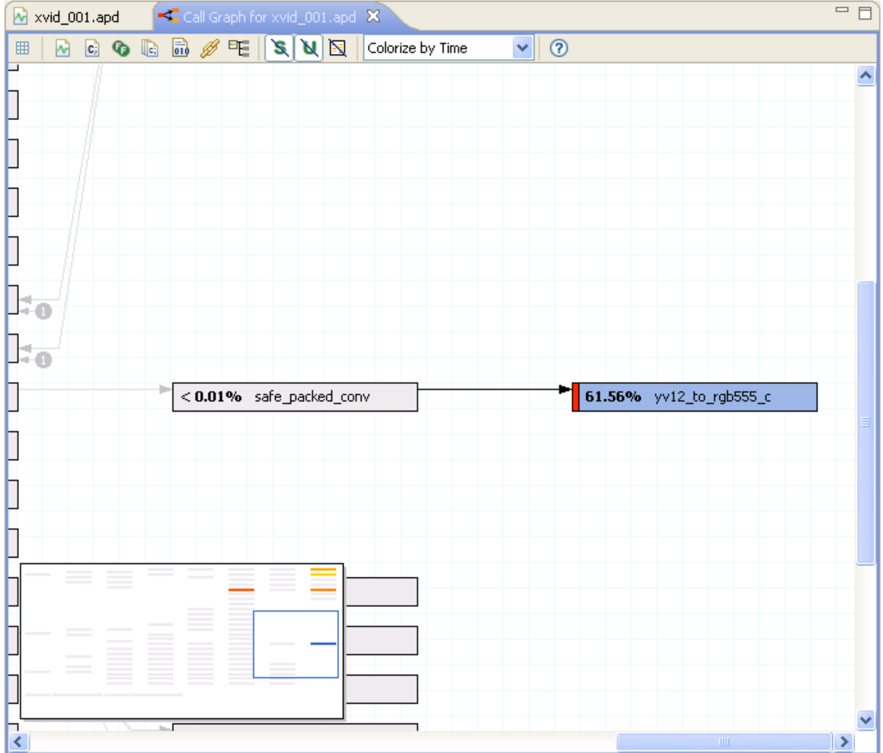


Figure 8-1 The Call Graph

The call graph is built to remain comprehensible even when the code example is complex. The use of callee and caller bullets eliminates many of the intersecting wires necessary to describe the complex relationships between the functions in your source code. Right-click on the bullet tab to see which functions called, or were called by, the attached function. For more information on callee and caller bullets, see on page 8-5*Caller and callee bullets* on page 8-5.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.2 Opening a call graph

The most direct route to an analysis file's call graph is through the Analysis Summary's overall analysis section. The call graph link, presented next to the icon is used to open that analysis file's call graph. In addition to using the navigator link in the summary report, you can also access the call graph from any of the other report types via the contextual menu or one of the other report's toolbar. Right-clicking on a function, or a group of functions, and choosing the **Select in Call Graph** menu option opens the call graph with the chosen function or functions selected.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.3 Call graph layout

The call graph provides you with a visual representation of your code hierarchy, laying out each function according to where it is called and using arrows to connect calling functions. The direction of the arrow indicates which function was the calling function. An arrow pointing to a function tells you that function is the callee and the function from which the line originates is the calling function. This section describes the layout of the call graph in more depth, providing a quick overview of how the hierarchy is built and what the bullets to the right and left of the call graph's functions represent.

8.3.1 How the hierarchy is built

The hierarchy of functions, as presented in the call graph, is built based on the call chain captured during execution. The originating function is placed in the far left column and functions it calls are placed in the column to its right. Functions that these functions call are placed in a column to the right of that and so on down the line, until all of the functions have been placed. There is a caveat to this placing behavior. If a function is called at multiple levels of the hierarchy, it is placed as far left as possible in the call graph.

To illustrate, if the function `main` calls function `a` which in turn calls function `b`, it appears as shown in Figure 8-2.

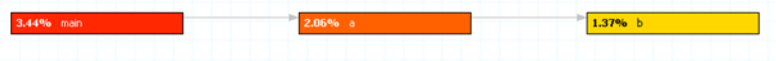


Figure 8-2 A Simple Call Hierarchy

If, in addition to function `a`, `main` also calls function `b`, function `b` is put in a higher place in the hierarchy, nearer to main. This is shown in Figure 8-3.

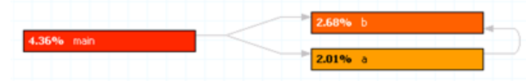


Figure 8-3 A Call Hierarchy with Multiple Call

The call graph presents a simple call hierarchy, but real-world algorithms describe hierarchies far more complex than those shown in Figure 8-2 and Figure 8-3. Rather than present the call graph with all of these connections visually represented as a spider web of call arrows, the call graph uses a simple method to determine whether or not to draw a call line.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.3.2 Caller and callee bullets

In cases where the calling function is in the same column or in a column just to the left or right of the called function, a call arrow is drawn from the caller to the callee. If, however, the called function appears in a column more than one column to the left of the calling function, a bullet is added to the left of the calling function and to the right of the called function. The number contained in the bullet represents how many calling or called functions are being shown this way.

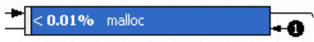


Figure 8-4 Caller Bullets

Right-click on a caller or callee bullet to see all of the functions contained in it. Choose a function in the contextual menu to select and center that function in the call graph. In this way, all of the calling and called functions are still easily accessible, but call arrows are not used to cross many layers of the hierarchy.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.4 The mini-map

In the bottom left hand corner of the call graph is a mini-map that can be used to easily navigate around the call graph when the hierarchy is too large to fit in the editor section of the ARM Workbench.

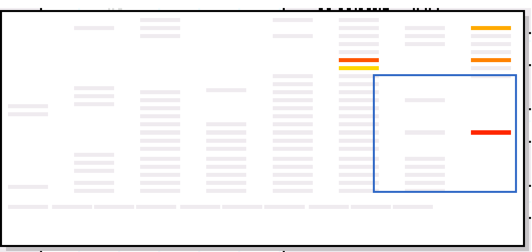


Figure 8-5 The Call Graph Mini-map

When inside the mini-map, but outside the draggable view area, the cursor changes into a crosshair. Click on a location in the mini-map to center on that section. If you hover over the draggable area box within the mini-map, the cursor changes into a hand that allows you to click and drag the view area. Panning the view area in this way enables you to quickly scan sections of the hierarchy without using the scroll bars.

The objects in the mini-map have the same color coding as the functions in the call graph itself. The bright red function in the hierarchy appears as bright red in the mini-map so that you can use the mini-map to quickly zoom to a bottleneck. Selected functions appear dark blue in the mini-map.

You can hide the mini-map by using the button, located just to the left of the drop-down menu in the toolbar.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.5 Color coding

You can use the drop-down menu in the toolbar to choose one of the two following options for color-coding the function boxes:

- Colorize by Time - This option is the default. It color codes the functions with the top five self time values so they are quickly identifiable in the call graph. These colors match the colors in the 'Top Five Functions' bar charts in the summary report and they range from bright red to bright yellow, red being the highest value, yellow the lowest, and the shades of orange representing functions two, three and four. These colors are easily identifiable in the mini-map so you can scroll quickly to these critical functions.
- Colorize by Coverage - This option colors the functions based on code coverage. Green indicates that all of the instructions were executed, while yellow represents partial coverage, and red indicates no coverage. Note that no red functions appear unless the 'Show Uncalled Functions' option is activated.

——— **Note** ———

The percentage found in each function label does not change when changing to either **Colorize by Time** or **Colorize by Coverage**. The percentages displayed in the function boxes are always a reference to the time spent as a fraction of the total execution time.

———————————

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.6 Selection behavior

Left-clicking on any function in the hierarchy selects it. In addition to coloring the rectangle dark blue, it changes the color of all of the arrows from gray to black, clearly showing you to what functions the selection function is connected. This behavior is shown in Figure 8-6:

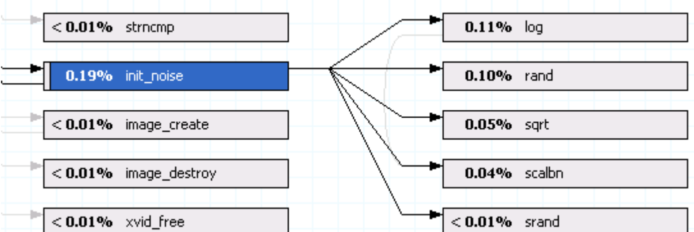


Figure 8-6 A Selected Function

Selecting a function also changes any connected caller or callee bullets from gray to black.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.7 Contextual menu options

Right-click on any function in the call graph to open a contextual menu, which presents you with a list of selection and navigation options so that you have easy access to that function's calling and called functions, as well as detailed statistics for that function via the table reports. Right-clicking on the call graph without a function selected opens a contextual menu that contains only two options, **Show System Functions** and **Show Uncalled Functions**, which have the same functionality as the and menu bar buttons. This section contains a description of this and every other option contained in a call graph contextual menu.

8.7.1 Showing system and uncalled functions

You can use contextual menu items to show or hide the system and the uncalled functions. These are only available if you right-click on an area outside of the function rectangles.

The ARM Profiler classifies all functions that begin with either an underscore or 'std::' as system functions, and hides them by default. You can, however, show them by choosing the **Show System Functions** drop-down menu option. All orphaned functions that are no longer connected to the tree when the system functions are hidden are shown as unconnected boxes at the bottom of the call graph. If this option is active, the contextual menu option changes to **Hide System Functions**.

The **Show Uncalled Functions** drop-down menu option works in much the same way as the **Show/Hide System Functions** option. Select **Show Uncalled Functions** and all of the functions contained in your code that were not called during the captured execution appears as disconnected boxes in the bottom of the hierarchy.

8.7.2 The caller and callee menu options

The two menu options that appear at the top of any call graph contextual menu are the caller and callee functions:

- Callers - Use the arrow to the right of this contextual menu option for a complete list of all of the functions that called the selected function. Choosing a function from this list selects it in the hierarchy. If the function is the root function, this option is grayed out.
- Callees - This menu option works identically to the 'Callers' menu option, only it contains a selectable list of functions that are called by the selected function. If the selected function does not call any functions, this menu option is grayed out.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.7.3 Contextual menu selection options

After the **Callers** and **Callees** options in a function's contextual menu are the options that allow you to select a group of functions relating to the selected functions:

- Select Callers - Use the **Select Callers** option to select all of the function's callers.
- Select Callees - This menu option selects all of the function's callees in the hierarchy.
- Select Caller Tree - The **Select Caller Tree** option selects more than just the caller functions, it selects every function and its parents all the way back to the originating function.
- Select Callee Tree - Similar to **Select Caller Tree**, this selects all of the functions the selected function called and all the called functions' children, as well.
- Select Callers and Callees - This option selects every function that the selected function was called by, as well as every function the selected function called.
- Select Caller Tree and Callee Tree - The **Select Caller Tree and Callee Tree**, option selects every function in the hierarchy that exists on the same call chain, from the originating function to all of the descendants of its called functions.

8.7.4 Contextual menu replay options

The function contextual menu also enables you to look at the selected functions in other report types:

- Set Breakpoint - Sets a breakpoint at the current location. The Run Forward and Run Backward commands can then be used to move the trace position between breakpoints.
- Run Forward to Selection - Runs forward to the next instance of the currently selected instructions, the next breakpoint, or the end of the program trace.
- Run Backward to Selection - Runs backward to the previous instance of the currently selected instructions, the previous breakpoint or the beginning of the program trace.
- Run Forward off Selection - Runs forward to the next unselected instruction, the next breakpoint, or the end of the program trace.
- Run Backward off Selection - Runs backward to the previous unselected instruction, the previous breakpoint, or the beginning of the program trace.
- Jump to Next Instance in Replay - Moves the indicator to the next instance of the instruction, ignoring breakpoints.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Jump to Previous Instance in Replay - Moves the trace indicator to the previous instance of the current instruction, ignoring breakpoints.

8.7.5 Contextual menu navigation options

The function contextual menu also enables you to look at the selected functions in other report types:

- Filter - This menu option filters the current selection's data from the report.
- Filter, Including Children - This option filters the statistics of the current selection and the statistics of all of its children from the report.
- Filter Others - The inverse of the **Filter** option, this option filters out everything but the current selection.
- Filter Others, Not Including Children - The inverse of **Filter, Including Children**, this option filters the statistics of everything but the current selection and its children.
- Select in Functions - Use this option to open the functions report with the chosen function highlighted in the table, so that you can see its call count and time statistics.
- Select in Files - **Select in Files** opens the files report and selects the file that contains the selected function.
- Select in Code View - This contextual menu option opens the code view for the selected function with all pertinent lines of code highlighted in the source panel and all of its associated instructions selected in the disassembly panel.
- Select in Call Chains - This menu option opens the call chains hierarchical table report with every instance of the selected function highlighted and every necessary calling function disclosed to show the function.
- Select in Classes - Use this option to select the chosen function's class in the classes report. This option only appears when the function is in a C++ file.
- Select in Call Summary - This menu option opens the call summary report, with the selected function as the primary function. Double-clicking any function in the call graph has the same effect as choosing this contextual menu option.
- Edit Source - This option opens the source file that contains the selected function in default source editor defined in the ARM Workbench. If no source code exists for this function, this option does not appear.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.8 The toolbar

The toolbar in the call graph contains the same set of navigation options that appear in other report types and adds the **Show System Functions** and **Show Uncalled Functions**. A button is disabled if the current selection makes their use invalid. Here is a description of each button in the toolbar:

- Remove Filters - Removes all filters from the call graph. The call graph disables this button if no filters are present.
- Show the Analysis Summary - Opens the analysis summary.
- Edit the source - Opens the source file that contains the selected function.
- Select in Functions - Opens the functions report with the chosen function selected.
- Select in Classes - Opens the classes report with the class of the chosen function selected. This is only present when working with C++ files.
- Select in Files - Opens the files report and highlights the file that contains the current selection.
- Select in Code View - Opens the code view for the selected function.
- Select in Call Chains - Opens the call chains report with all instances of the currently selected function highlighted and their parent functions expanded to expose them. This option is not enabled for any of the uncalled functions listed at the bottom of the call graph.
- Select in Call Summary - This button opens the call summary with the selected function as the highlighted function.
- Toggle System Functions - This button has the same effect as the contextual menu item. It shows all of the system functions in the call graph if they are currently hidden, and hides them if they are shown. This button is highlighted if system function visibility is currently off.
- Toggle Uncalled Functions - Toggles uncalled function visibility on and off. If the button is highlighted, uncalled functions are currently hidden.
- Toggle Mini-map - Toggles mini-map visibility on and off.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Show Help - This button opens the help view with a list of topics relevant to the call graph.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

8.9 The outline view

The outline view in the call graph works in much the same way as the outline view in the table reports. It presents every function from the source code, in an alphabetical list. Select any function from the outline view and the call graph moves the focus of the editor to the position of that function.

To open the outline view, select **Window** → **Show View** → **Outline** from the ARM Workbench menu. By default, the outline view appears to the right of the editor section. Right-clicking on any selection in the outline view will open a contextual menu with navigation options to every other report type.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 9
The Call Summary

This chapter explores the functionality of the call summary. It includes a section that breaks down the items in the call summary window, a section that explains call summary navigation, a section on filtering, and a section that describes each of the toolbar options and contextual menu options.

It contains the following sections:

- *Call summary breakdown* on page 9-2
- *Function box statistics* on page 9-3
- *Filtering instances* on page 9-6
- *Call summary navigation* on page 9-8
- *Navigating to other report types* on page 9-9.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

9.1 Call summary breakdown

This section guides you through the call summary and explains each of the statistics found in the function box. The ARM Profiler shows you a call summary window taken from the xvid sample that you can find by selecting **All Programs → ARM → ARM Profiler v2.1 → Examples**. This sample call summary is shown in Figure 9-1:

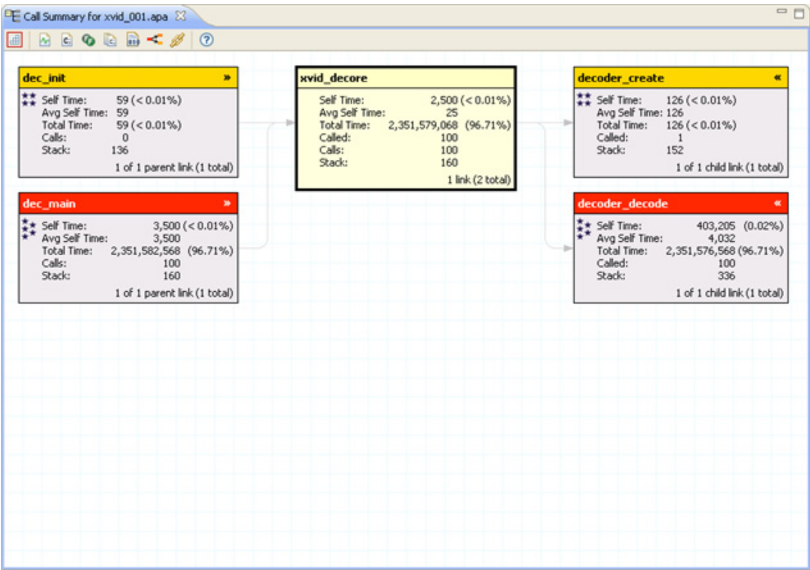


Figure 9-1 The Call Summary

The box in the center, `xvid_decore` is referred to as the primary function. The system functions on the left, `dec_init` and `dec_main`, are the calling functions while the functions on the right, `decoder_create` and `decoder_decode`, are the called functions.

Like the call graph, the → arrows indicate the direction of the call, but unlike the call graph, calling functions are always on the left and called functions are always on the right. Everything in the call summary is shown as it relates to the primary function, including the statistics shown in the function boxes. Clicking on the navigation icon in the upper right of any of the calling or called functions makes that function the new primary function, and reveals all of its calling and called functions.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

9.2 Function box statistics

The statistics listed in each of the function boxes are a subset of the statistical fields provided in the table reports, and, when the call summary is first opened, the numbers in the primary function box match up to the data presented in the functions report. One of the benefits of the call summary is that the numbers are updated based on the current filters. You can see a function's numbers as they relate to specific instances.

The ARM Profiler also color-codes the title bar of the calling and called function boxes based on the self time value reported in the function box. The given color-coding is identical to the colors used in the summary report, with red representing the highest self time value. Each of the color coded boxes are also marked with stars to help differentiate between the colors. Five stars are listed below the title in the function box with the highest self time value and one star is listed in the function box with the fifth highest.

9.2.1 Basic statistical fields

Call summary statistics are heavily dependent on the scope of filters applied in this view and others. The call summary view supports a wide assortment of filtering techniques. When a filter is applied on a function, statistics up and down the call chain are updated. Only statistics for unfiltered instances are shown in the Call Summary. For more information on filtering in the call summary, see *Filtering instances* on page 9-6. Here is a description of each of the statistical fields:

- Self Time - This value measures how much time the unfiltered instances use. Self time does not include the value of its subordinate branches, it is called self time because it represents the amount of time executing the instructions from this function alone. The units of time given in the call summary are variable based on the method of data collection. Data collected using an RTSM is measured in estimated cycles, while data collected using hardware and RealView Trace 2 is measured in samples. Every so often, based on the given sampling rate, RealView Trace 2 reports which function is currently being executed. That sample is then attributed to the self time for the function observed.
- Avg Self Time - This value shows you the average amount of self time consumed each time this function was called.
- Total Time - This value is measured the same way as self time, but the time spent in subordinate branches is included in the total.
- Called - In the case of the primary function, this number shows you how many times the primary function was called by the unfiltered calling function. For the called functions, this value shows how many times the function was called by the unfiltered instances of the primary function only. Like all the data in the call summary, the provided data is in relation to the primary function.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Calls - In the case of the primary function, this unit shows how many times the function called its unfiltered subordinate functions. For the calling functions, this number shows how many times the functions called the unfiltered instances of the primary function.
- Stack - The number of bytes used by the stack in this function.

9.2.2 Instances: the x of y (z total) notation

The instance notation, listed in the bottom right of every function, gives you the complete instances picture for the function. It tells you how many total instances of the function exist in the entire call chain as well as how many instances are currently filtered out. The number listed in the primary function is different than the notation in the calling and called functions. The ARM Profiler provides a closer look at the `xvid_decore` primary function:

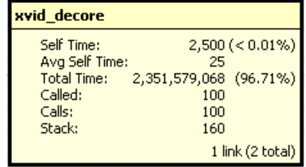


Figure 9-2 The `xvid_decore` Primary Function

In the primary function, only two numbers are listed. In the case of `fputc`, the notation provided is '1 links (2 total)'. This shows:

- There are two instances of the function in the call chain as a whole, two unique places that this function exists in the call chain. This number is listed as a parenthetical.
- There is one currently active, unfiltered instance of the function. Any filtering of calling and called functions reduces this number.

Now have a look at the calling function, `dec_init`:

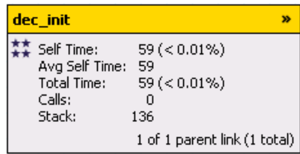


Figure 9-3 The `dec_init` Calling Function

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

The notation in the bottom corner says '1 of 1 parent links (1 total)', providing an additional number. The notation in the calling and called functions tells us:

- The parenthetical means the same thing as in the primary functions. It tells you how many total instances of the function exist in the call chain.
- The x of y notation tells us that, out of the y possible instances that call or are called by this function x are currently unfiltered. If you filter out `P_ChangeSector`, this number changes to '0 of 1 parent links'. This function calls `xvid_decore`, but, because this instance is filtered out, '0' are currently active.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

9.3 Filtering instances

You can use the contextual menu options, **Filter**, **Filter, Including Children**, **Filter Others**, and **Filter Others, Not Including Children** to filter out instances. Right-click on a function and select one of these options to filter statistics from the call summary. The **Filter** and **Filter, Including Others** options will filter out data from the selected function, while the **Filter Others** and **Filter Others, Not Including Children** options will filter out everything but the selected function. Functions filtered in this manner are not removed from the call summary, but they are grayed out and all statistics relating to this function are removed.

The xvid code used in the earlier figures again serves as the example, this time to illustrate the filtering behavior of the call summary. Right-clicking on the `dec_init` function box and choosing the **Filter, Including Children** menu option, updates the call summary to look like the one shown in Figure 9-4.

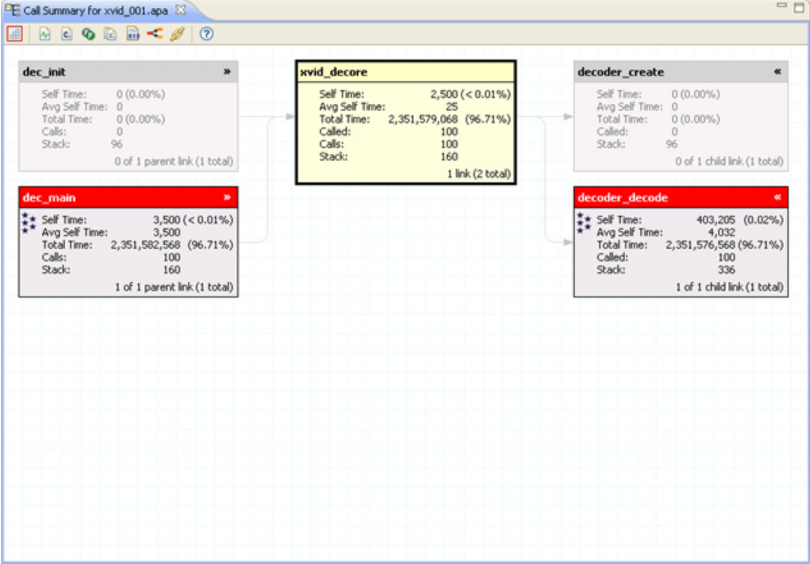


Figure 9-4 Filter, Including Children

Notice that the statistics from both `dec_init` and its subordinate links are removed, and the call summary grays out the function boxes for `dec_init` and `decoder_create`. The total number of links in the primary function, `xvid_decore`, is reduced by one.

Right-clicking on a function and selecting the **Filter** menu command only removes the selected function. It does not filter out subordinate functions. Right-clicking on `dec_init` and choosing the Filter menu option updates the call summary to look like the one shown in Figure 9-5 on page 9-7:

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

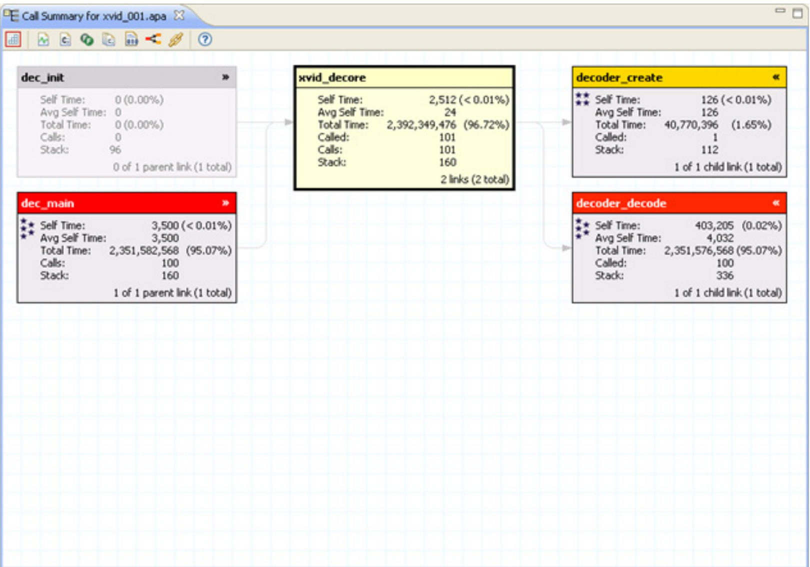


Figure 9-5 A single filter

To remove all filters from the call summary use the button located on the far left side of the toolbar. If there are any active filters, the call summary changes the appearance of this button and highlights it in red.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

9.4 Call summary navigation

To explore your call chain in the call summary, click on the navigation buttons in the upper right any of the caller or callee functions. This shifts the call summary, making the selected function the new primary function.

9.4.1 The outline view

You also use the outline view to navigate to a specific function. When the call summary is active, clicking on a function in the outline view makes that function the primary primary function. If the outline view is not already there, you can open it using the **Window** → **Show View** → **Outline** menu option.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

9.5 Navigating to other report types

Just like in the other report types, the call summary enables you to quickly navigate to any other report in the ARM Profiler, both through the use of toolbar buttons and contextual menu items. Double-clicking any of the functions in the call summary opens the call chain, with every instance of the function selected and exposed.

9.5.1 Navigating using the toolbar

You can use the toolbar buttons to open another report type based on the current selection in the call summary. Select a function and press the function report button, and the ARM Profiler opens the function report with that function selected and in view. Here is a list of each of the buttons in the toolbar of the call summary:

- Show the Analysis Summary - Opens the analysis summary.
- Edit the source - Opens the source code that contains the selected function.
- Select in Functions - Opens the functions report with the call summary function selected in the newly opened report.
- Select in Classes - Opens the classes report with the chosen function's class selected in the newly opened report.
- Select in Files - Opens the files report with the file that includes the chosen function selected in the files report.
- Select in Code View - Opens the code view with all of the chosen function's code and instructions selected.
- Select in Call Graph - Opens the call graph, another graphical view of the call hierarchy, with the chosen function selected and centered in view.
- \- Select in Call Chains - Opens the call chains report with all unfiltered instances of the chosen function selected and all necessary links collapsed to disclose them. The call chains report and call summary work in tandem - all active instances in the call summary are selected in the call chains report when that report is opened. Similarly, any selection removed in the call summary is removed from the call summary's active instances when you re-open the call summary. Switch between the two reports to see how all of the call summary's active instances fit in the call chain. If the **Select in Call Chains** button is pressed when a calling or called function is selected, the call chains report opens with only the included instances that relate to the primary function selected. This option has the same behavior as double-clicking on a function box.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Show Help- Opens the help view, with a list of help topics relevant to the Call Summary. This button has the same effect as pressing the **F1** key (or **Shift + F1** in Red Hat Linux) when a call summary is active.

9.5.2 Navigating using the contextual menus

To open a contextual menu option, right-click on any of the function boxes in the call summary. The navigation options presented here have identical behavior to their corresponding toolbar options:

- Filter
- Filter, Including Children
- Filter Others
- Filter Others, Not Including Children
- Select in Functions
- Select in Classes
- Select in Files
- Select in Code View
- Select in Call Graph
- Select in Call Chains
- Edit Source

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 10
Merging Analysis Files

This chapter explains the process of merging analysis files. It provides a description of the benefits of merging analysis files, an explanation of analysis file compatibility, and a description of the process.

It contains the following sections:

- *Reasons to merge analysis files* on page 10-2
- *Analysis file compatibility* on page 10-3
- *How to merge analysis files* on page 10-4.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

10.1 Reasons to merge analysis files

The ARM Profiler provides you with the capability to merge multiple analysis files into a single file, provided the analysis files are produced using the same source code and processor. Doing this can help you streamline your code, while also giving a more accurate view of code coverage.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

10.2 Analysis file compatibility

To merge analysis files, they must fulfill the following compatibility requirements:

- Analysis files must be produced using the exact same version of the image, compiled using the same compilation options.
- Analysis files must be created using the same data collection mode. Data collected using a Real-Time System Model is not compatible with data collected using RealView Trace 2.
- Analysis files must be created using the same model or model file. You can not combine analysis files if they are created on different hardware.
- If the analysis data is captured using RealView Trace 2, the sampling rates used for the analysis files must match.

Trying to combine incompatible analysis files produces an error dialog with the text, `The specified analysis data cannot be merged`.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

10.3 How to merge analysis files

To merge multiple analysis files into a single file, first select the compatible analysis files you would like to merge in the ARM Profiler Data view, then right-click and choose the **Merge** option from the resulting contextual menu.

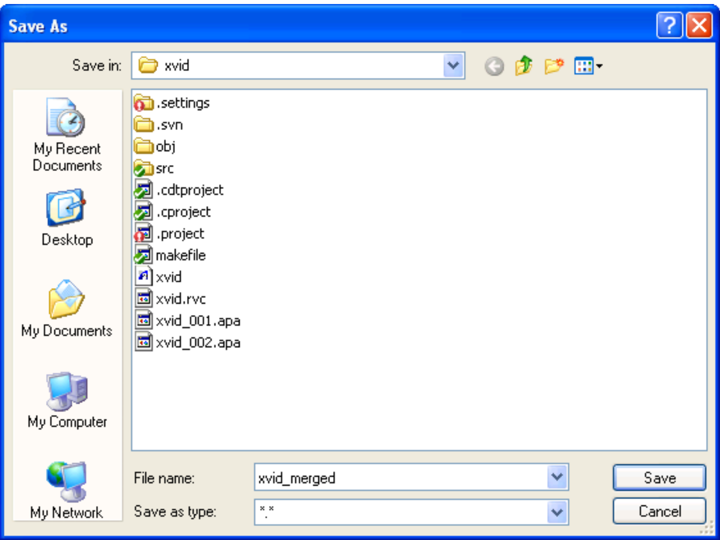


Figure 10-1 Merging Analysis Data

If successful, a dialog appears, asking you to name the newly merged file.

You can also use the `-m` option with `armprorep` to merge files on the command line. For more information on the ARM Profiler command line functionality, see *Command line options* on page A-3.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 11
Preferences

This chapter explores every ARM Profiler option available within the ARM Workbench preferences. It includes a section on how to access preferences and a description of every option available in the colors and fonts folder.

It contains the following sections:

- *Accessing the ARM Profiler color preferences* on page 11-2
- *Color preference descriptions* on page 11-3.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

11.1 Accessing the ARM Profiler color preferences

The ARM Profiler interface allows for a degree of flexibility in how the data is presented within the ARM Workbench. Using the ARM Profiler folder within the colors and fonts preferences page, you can change appearance options to match your taste.

To access the ARM Profiler-specific color options, open the ARM Workbench preferences using the **Window** → **Preferences...** menu option, and use the hierarchical control to expand the 'General' preference list. Expand the 'Appearance' preference in the same manner and click on 'Colors and Fonts' from the list that appears beneath it.

Now open the ARM Profiler folder to begin modifying your color scheme. Figure 11-1 shows the updated Preferences window:

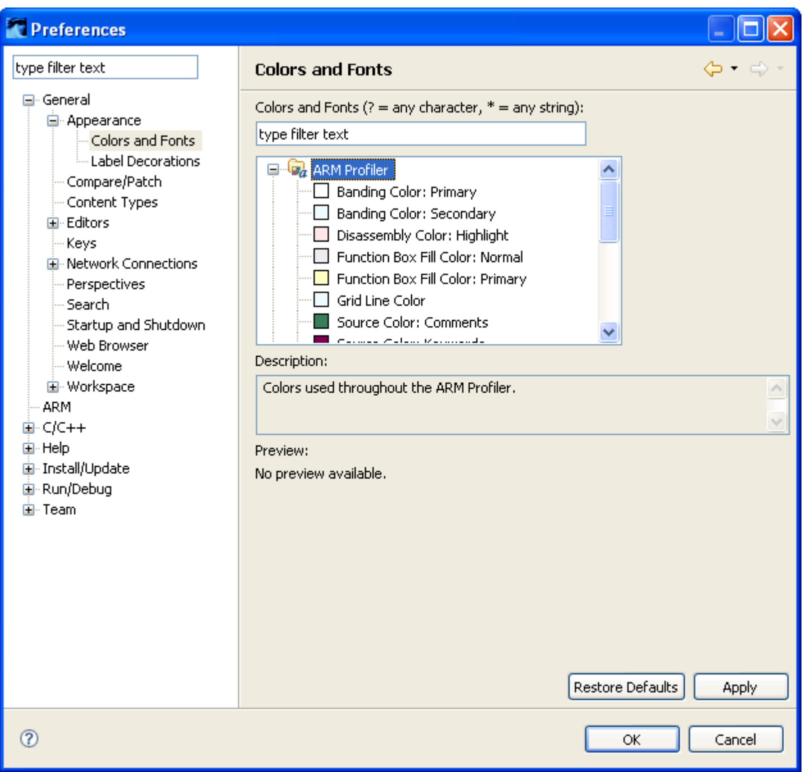


Figure 11-1 Color Preferences Window

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

11.2 Color preference descriptions

Within the ARM Profiler color preferences folder, there are the following options that change the way the plug-in looks and feels:

- Banding Color: Primary - The table reports in the ARM Profiler use alternating colors to make data easier to track as you follow a row from left to right. This color option changes the color of all of the odd rows in the table reports.
- Banding Colors: Secondary - The 'Banding Colors: Secondary' option changes the color of all the even rows. If you do not like the banding effect, use this option to change the secondary color to match the primary banding color.
- Disassembly Color: Highlight - Changes the color used to mark strings within the disassembly code.
- Function Box Fill Color: Normal - This option affects the colors of the function boxes in both the call graph and the call summary. In the call graph, this color option changes the standard function box color, so that all of the functions not color-coded red, yellow, or orange due to a high self time are updated to match the color you define here. It is best to avoid shades of red, yellow and orange, as these color choices conflict with the call graph's color-coding. All of the calling and called functions in the call summary are also updated to the chosen color. It does not affect the primary color function.
- Function Box Fill Color: Primary - Unlike the other function box fill color option, this option only affects the call summary. It changes the background color of the primary function, the bold-bordered function normally colored yellow in the call summary.
- Grid Line Color - You can use this option to change the color of the grid that is used to create the graph paper effect in the call graph and the call summary. If you do not like the grid effect, change this color to white.
- Quality Color: Unknown - Sets the color used to draw unknown values in the replay view.
- Replay Color: Focus Background - Sets the color used to draw the background of the focus area in the replay view.
- Replay Color: Focus Lines - Changes the color used to draw the separator lines in the focus area of the replay view.
- Source Color: Comments - This option changes the color of comments in the source code area of the code view.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- Source Color: Keywords - This option changes the color of keywords in the source code that appears in the ARM Profiler code view.
- Source Color: Strings - This option changes the color of strings that appear in the source section of the code view.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 12
Profiling Applications Running on the Symbian OS

This chapter describes the process of profiling Symbian OS applications, from building the Symbian OS with the profiler kernel extension to defining your application in the launcher panel.

Note

The instructions in this chapter are for building and running a Techview image.

It contains the following sections:

- *Building the ARM Profiler Symbian OS kernel extension* on page 12-2
- *Building Symbian OS applications* on page 12-4
- *Profiling your Symbian OS application* on page 12-5.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

12.1 Building the ARM Profiler Symbian OS kernel extension

To build the Symbian OS kernel with profiling enabled:

1. Install the Symbian build system on your host machine.
2. Build the Symbian OS according to your normal build process. For more information on building the Symbian OS, see the Symbian Developer Library at:

   `http://developer.symbian.com/main/documentation/sdl/`
3. Create a directory called `armproske` in `\src\CEDAR\GENERIC\base\e32\drivers\`.
4. Copy all files located in *ARM install directory*`\Profiler\Contents\`*version number*`\`*build number*`\target_os_support\symbian` directory to the `armposke` directory you created in step 2.
5. In the `\src\CEDAR\GENERIC\base\e32\drivers\armproske directory,` build the required build files by invoking the following command:

   ```
   bldmake bldfiles ARMV6
   ```
6. Build the kernel extension with the following command:

   ```
   abld.bat -v build ARMV6
   ```
7. In the `\bin\TechView\epoc32\rom\`*PLATFORM*`\kernel.iby` directory, open `kernel.iby` with any text editor. For example:

   ```
   edit kernel.iby
   ```
8. Add the following two lines before the first kernel extension:

   ```
   // Profiler Kernel Extension
   extension[VARID]= /
   \Epoc32\Release\##KMAIN##\##BUILD##\armproske.dll \Sys\Bin\armproske.dll
   ```

   Note

   Inserting the ARM Profiler extension before any other ensures that the ARM Profiler is aware of every extension loaded after it.
9. To use TechView, you also need to edit the `base_`*platform*`.iby` file:

   ```
   edit base_platform.iby
   ```

   The *platform* variable in the filename represents the processor family of your hardware. Here are three filename examples:

   - `base_ct1136.iby`
   - `base_cortex.iby`
   - `base_rvvp926.iby`

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

10. Insert the following line into the `base_platform.iby` file:

```
extension[VARID]= /
KERNEL_DIR\DEBUG_DIR\armproske.dll              \sys\bin\armproske.dll
```

11. Build the ROMs according to your normal process.

12. Depending on how you build your Symbian OS image, you may need to use a .etm script file to load the image before the application. For more information on using script files, see *Setting up scripts* on page 3-5.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

12.2 Building Symbian OS applications

To build a Symbian OS application so that it is ready for profiling:

1. When compiling with RVCT, make sure the `-g` and the or `--dwarf3` option is enabled in the `.mmp` file for the application. The `--dwarf2` option also works, but the `--dwarf1` option does not.

   Note

   The `-g` option turns on debug information so that the ARM Profiler can relate assembly code to source code and the `--dwarf2` and `--dwarf3` options dictate that the compiler uses `dwarf2` or `dwarf3` specifically.

2. Build your application for the target ARM processor.

   Note

   Building a Symbian OS application using RVCT creates two executables, both with the .exe file extension. The ELF executable, located in the *Symbian install directory*\bin\TechView\apoc32\release*platform**build type* directory, must be used for profiling, not the e32 executable located in the *Symbian install directory*\bin\TechView\apoc32\build\src\cedar\generic\base\e32test\group\exec name*platform**build type* directory. If you are unsure if the executable file is an ELF file, open it in a text editor and check to see if three of the first four characters are ELF, or use the fromelf.exe utility.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

12.3 Profiling your Symbian OS application

After you have built the Symbian OS and at least one Symbian OS application. To do this:

1. From the ARM Workbench IDE, select **Run** → **Open Run Dialog** to open the run configuration window.
2. Select **RealView Trace 2** or **Real-Time System Model** in the **Connection** tab.
3. Disable semihosting using the checkbox in the **Connection** tab.
4. In addition to all of the standard steps detailed in chapters 3 and 4, enter the elf OS image file in the **Images** tab. If your Symbian OS is already loaded on the target, or is loaded by the **Before Transfer** or **After Transfer** scripts, skip to step 6.

   ——— **Note** ———

   If you use the **Browse** button to locate the `.exe` file, make sure you use the Files of Type drop-down menu at the bottom of the dialog to specify the **Executables** option. Otherwise, the Symbian `.exe` file is not shown in the list of available files.

   ————————————

5. Select **Load Image** from the drop-down menu.
6. Click the plus button to add another image file.
7. Enter the location of the Symbian OS application to profile.
8. In the **Profiling** tab of the launcher, set the Symbian OS application to **Loaded by OS**.
9. Ensure that the **Generate ETM Context IDs** checkbox is selected.
10. Click **Run**.
11. After the Symbian OS starts, navigate through the interface to the application you want to profile.
12. Run the application in the Symbian OS Techview.

   ——— **Note** ———

   Because the ARM Profiler can not recognize an application that has already started, you must connect the profiler before you launch the application.

   ————————————

13. Click **End Capture** in the Live Update panel after the application has terminated.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Note

The ARM Profiler does not profile the Symbian OS itself. Any calls to the kernel will show up in reports as non-profiled code.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 13
Profiling Applications Running on Linux OS

This chapter describes the process of profiling Linux applications, from building the Linux kernel with the appropriate ARM patches to defining your application in the launcher panel.

It contains the following sections:

- *Installing and patching the Linux kernel extension* on page 13-2
- *Profiling your Linux application* on page 13-4

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

13.1 Installing and patching the Linux kernel extension

To build an ARM Profiler-enabled kernel:

1. Download and unpack version 2.6.28-1 of the Linux kernel. Right now, this is the only version of Linux that the ARM Profiler supports.

2. If you are using ARM Embedded Linux, you can get this patch file from http://www.arm.com/products/os/linux_download.html. If you are using ARM Linux from an alternate vendor, please contact your vendor.

   - `patch-2.6.28-arm1.gz`

     Note

     The `patch-2.6.28-arm1.gz` patch is only required if you are intending to profile Linux applications running on ARM boards. In any other case, only the patch listed below is necessary.

   - `patch-2.6.28-armpro1`

     Note

     This patch can be found in *ARM Installation Folder*/Profiler/Contents/2.1/*build number*/target_os_support/linux/

3. Unzip and install the ARM patch by entering the following command from within the top level of the kernel source tree.

   ```
   gz -cd < ../patch-2.6.28-arm1.gz | patch -p1 -
   ```

   In the above command, the kernel patch is located one level above the directory where the command is executed. The `../patch-2.6.28-arm1.gz` path should be replaced with the path to your patch.

4. Apply the ARM1 patch by invoking the following command from the Linux install directory:

   ```
   patch -p1 < patch-2.6.28-arm3
   ```

5. Apply the ARM Profiler patch:

   ```
   patch -p1 < patch-2.6.28-armpro1
   ```

6. Build the Linux kernel according to your normal process.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Note

There are many ways to boot the Linux kernel and mount the root file system that is required by the kernel. One method is to load your system using `.cmm` scripts defined in the ARM Profiler launch configuration dialog. For more information on using script files, see *Setting up scripts* on page 3-5.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

13.2 Profiling your Linux application

Once you have enabled ARM Profiler support in your kernel and you have built your image without stripping its symbols, you are ready to profile. You need two copies of the image that you want to profile:

- one for the target side
- one for the host side.

Make sure that the host side ELF file has its symbols. Debugging symbols are not required, but they give you the source code to disassembly correlation in the analysis report.

On the target side, the ELF file does not require any symbols and you can strip them to keep the image size small. To start profiling, follow these steps:

1. Select **Run → Open Run Dialog** from within the ARM Workbench IDE to open the run configuration window.
2. Select either **RealView Trace 2** or **Real-Time System Model** in the **Connection** tab, depending on whether you are profiling on Linux using hardware connected to a RealView Trace 2 unit or on an RTSM.
3. Disable semihosting using the checkbox in the **Connection** tab.
4. Enter the location of the Linux application to profile in the **Images** tab.

   ——— **Note** ———

   Make sure the specified image file matches the application loaded on the Linux image. An image file that does not match can cause runtime errors.

   ————————————

5. Select **Loaded by OS**.
6. Ensure that the **Generate ETM Context IDs** checkbox is selected.
7. Click **Run**.
8. After Linux booting completes, launch the application that you want to profile.

   ——— **Note** ———

   Because the ARM Profiler can not recognize an application that has already started, you must connect the profiler before you launch the application.

   ————————————

9. Click **End Capture** in the Live Update panel after the application has terminated.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

——— **Note** ———

The ARM Profiler does not profile the Linux kernel itself. Any calls to the kernel will show up in reports as non-profiled code.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Appendix A
Using the Command Line

Appendix A is a reference that includes descriptions of each of the command line options in the ARM Profiler.

It contains the following sections:

- *Configuring your system for running the ARM Profiler on the command line* on page A-2
- *Command line options* on page A-3.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

A.1 Configuring your system for running the ARM Profiler on the command line

While it is easier to navigate the analysis reports in the ARM Workbench, you can also use the command line to generate reports. Using the `armprorep` command, you can merge analysis files and generate table reports that you can view in your shell or output to tab-delimited or .csv files. In this way, you can automate the creation of multiple reports for the sake of comparison without ever opening the ARM Workbench.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

A.2 Command line options

Every one of the table reports available in the ARM Workbench IDE is also available from the command line. This mode of operation makes it easy to script the analysis process, so you can create fully automated comparisons of reports for a variety of ARM microprocessors. You can also test against a variety of data samples and generate any of the resulting reports.

All of the options shown can be called using the short one-letter call, or using the more verbose '--' call. A '/' also works in calling single letter options in DOS in place of the '-'. Every option is called using the same basic syntax: `armprorep [options] [file_list]`

- `-h, -?, --help`- Provides the version information and a short description of each command line option available with `armprorep`.
- `-v, --version` - Provides copyright information and the current version number of the ARM Profiler.
- `-o, --output` - Sends all output text into the specified file.
- `-m, --merge` - Merges multiple analysis files into a single file. For this option, the first name in the `[file_list]` is the name of the newly merged output file. For example: `armprorep -m output_file analysis_file_1 analysis_file_2 analysis_file_3`. The `--merge` takes precedence over the other options in this list. If `--merge` is called on the command line, all other options are ignored. See chapter ten for more information on merging analysis files.
- `-a, --all` - Produces every table report and the code view report and outputs it to your shell. The syntax for this and all of the reporting command line options is as follows: `armprorep -a analysis_file`. If more than one analysis file is listed, all of the reports are provided for each analysis file given.
- `-n, --callchain` - Provides the call chains report for the specified analysis files.
- `-s, --codeview` - Provides the disassembly section of the code view report for the specified analysis files.
- `-c, --classes` - Provides the classes report for the specified analysis files. As a standard C file does not have an associated classes report, this option is only available if the analysis file was created using C++.
- `-f, --files` - Provides the files report for the specified analysis files.
- `-p, --functions` - Provides the functions report for the specified analysis files.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

- `-t, --tabs` - To be used in conjunction with the command line reporting options, this command changes the formatting of the command line output to tab-delimited. By default, the command line reports are output to your shell using spaces so that the columns line up neatly. If you want to output data to a file easily digested by a spreadsheet application, use the -t option and the > command line output command to produce a tab-delimited file, for example: `armprorep -p -t analysis_file > output_file.tab`
- `-e, --csv` - This option, used in the same manner as `--tabs`, produces output separated by commas. Use in conjunction with the output command to create csv files from the command line.
- `--call_chain` - This option enables you to specify clock speed if you have the sample rate set to **Estimated Cycles**. This option is only available for hardware profiling. Do no use this option when profiling with an RTSM.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Appendix B
Keyboard shortcuts

Appendix B is a reference that includes a list of every keyboard shortcut for every report available in the ARM Profiler and a brief description of the contextual behavior of each shortcut.

It contains the following sections:

- *Table report keyboard shortcuts* on page B-2
- *Code view keyboard shortcuts* on page B-3
- *Call graph keyboard shortcuts* on page B-5
- *Call summary keyboard shortcuts* on page B-6.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

B.1 Table report keyboard shortcuts

While you can navigate every report in the ARM Profiler using the mouse, you can also use keyboard shortcuts. Here is a list of the keyboard shortcuts available for the functions, files, classes, and call chains table reports:

- Up arrow- Moves the current selection up one row.
- Shift + up arrow- Adds the row above to the current selection.
- Down arrow- Moves the current selection down one row.
- Enter- Has the same effect as double-clicking on the active row. The new report type depends on the currently active report.
- Shift + down arrow- Adds the row below to the current selection.
- Home- Selects the first row in the active table report.
- End- Selects the last row in the active table report.
- Page Up- Moves up in the current report one page. A page is defined by the range of rows currently displayed in the table report.
- Page Down- Moves down one page.

The call chains report has some unique keyboard shortcuts, not available in the other table report types:

- Right arrow- Discloses the subordinate rows for the currently selected call chain links. Has the same effect as clicking on the disclosure control to the left of the call chain link's title.
- Left arrow- Hides the subordinate rows for the currently selected call chain links.
- Shift + right arrow- Discloses all of the subordinate rows for the currently selected call chain links. The entire hierarchy below the selected links is revealed.
- Shift + left arrow- Hides all of the subordinate call chain links. On the surface, it has the same effect as just pressing the left arrow by itself, but when the subordinate functions are once again revealed, this command ensures that only the immediately subordinate functions appear.
- n- Finds and selects the next call chain link with a note attached to it. The call chain hierarchy is collapsed to reveal the selected link.
- p- Finds and selects the previous call chain link with a note attached to it.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

B.2 Code view keyboard shortcuts

Here is a list of the available keyboard shortcuts for the code view reports:

- Up arrow - Moves the current selection up one row.
- Shift + up arrow - Adds the row above to the current selection.
- Control + up arrow - Scrolls the current view up without changing the selection.
- Down arrow - Moves the current selection down one row.
- Shift + down arrow - Adds the row below to the current selection.
- Control + down arrow- Scrolls the current view down without changing the selection.
- Right arrow- If the currently selected instruction is a branching instruction, the right arrow key takes you to its call destination. Pressing the right arrow key when on a non-branching instruction selects the next instruction.
- Left arrow- If the selection is a branching instruction, the left arrow key takes you to its call destination. Pressing the left arrow key when on a non-branching instruction selects the instruction above it. The right and left arrow keys have no effect in the source panel.
- Enter - Highlights all of the associated instructions. Pressing enter on a function title in the disassembly panel highlights every instruction in the function. This keyboard shortcut is only available in the disassembly panel.
- Home - The home key takes you to the top of the function that contains the currently selected row. If a line of code is selected in the source panel that does not have any instructions associated with it, the **Home** key takes you top of the source file.
- End - The end key takes you to the bottom of the function that contains the currently selected row. Like the home key, if the selected line of source does not have any instructions associated with it, the end key takes you to the bottom of the file.
- Page Up - Moves up one page. A page is defined by the range of rows currently displayed in either the source or disassembly panel.
- Page Down - Moves down one page.
- Ctrl+F - Pressing Ctrl+F activates the find feature in the code view.
- Esc - Removes the find field from the bottom of the code view, if applicable.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

If trace replay is enabled, the following keyboard shortcuts can be used when the replay view is active:

- Up arrow - moves to the previous replay instruction.
- Down arrow - moves to the next replay instruction.
- Right arrow - Jumps to the next instance of the current instruction in the trace replay.
- Left arrow - Jumps to the previous instance of the current instruction in the replay view.
- F8 - Runs to the next breakpoint or the end of the trace replay, whichever comes first.
- Shift + F8 - Runs to the previous breakpoint or the beginning of the trace replay.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

B.3 Call graph keyboard shortcuts

Here is a list of the keyboard shortcuts that help you navigate the call graph:

- Up arrow- Moves the current selection up one function box.
- Down arrow- Moves the current selection down one function box. The down arrow does not move the selection to the uncalled and disconnected functions. These must be selected using the mouse.
- Right arrow- Moves the current selection to the right. If no function is to the immediate right of the current function box, the ARM Profiler chooses the closest available function in the row to the right of the currently selected function.
- Left arrow- Moves the current selection to a function box to the left of the currently selected function. Works in the same manner as the right arrow command.
- Home- Selects the top function box in the current call graph row.
- End- Selects the bottom function box in the current call graph row.
- SPACEBAR- Holding down the space bar turns the mouse cursor into a hand and enables you to click and drag the viewable area.
- TAB- Cycles the selection to the next highest self time value.
- Shift + TAB- Cycles the selection to the function one above the current selection in terms of its self time value. For example, if the function with the third highest self time value is selected, pressing shift + TAB selects the second highest function.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

B.4 Call summary keyboard shortcuts

Here is a list of the keyboard shortcuts that you can use in the call summary:

- Up arrow- Moves the current selection up one function box, if possible.
- Down arrow- Moves the current selection down one function box.
- Right arrow- Selects the function box to the right of the currently selected function box.
- Left arrow- Selects the function box to the left of the currently selected function box.
- Home- Selects the top function box in the current call summary row.
- End- Selects the bottom function box in the current call summary row.
- Spacebar- Holding down the space bar turns the mouse cursor into a hand and enables you to click and drag the viewable area.
- TAB- If a calling or called function box is selected, hitting the TAB key cycles the selection to the next highest self time value amongst the calling or called functions.
- Shift + TAB- If a calling or called function box is selected, pressing shift + TAB selects the function one above it in terms of self time value.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Appendix C
Troubleshooting guide

Appendix C is an important resource if you are having difficulty profiling your code using RealView Trace 2.

It contains the following section:

- *Troubleshooting steps* on page C-2.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

C.1 Troubleshooting steps

If you configure your hardware according to the process detailed in chapter two of this User Guide and the image still fails to execute when running through the ARM Profiler, consider the following common causes:

1. The "`ARM Profiler: WARNING: Trace buffer overflow`" error message may indicate that the hardware target is running too fast relative to the CPU and disk performance of your development machine. The full live update feature, configurable in the ARM target run configuration window, can slow down the host. Try turning off this option if you receive too many "`ARM Profiler: WARNING: Trace buffer overflow`" error messages.

   Note

   "`ARM Profiler: WARNING: Trace buffer overflow`" warnings and “`ARM Profiler: WARNING: ETM buffer overflow`” warnings can lead to bad output like analysis files that cannot be loaded and invalid samples count.

2. The "`ARM Profiler: WARNING: ETM buffer overflow`" error message indicates that too much data is trying to pass through the ETM port of the target for the ARM Profiler to handle. Try increasing the port size of the ETM/TPIU, or, for supported targets, use estimated cycles to remove cycle information from the ETM trace. To profile on hardware using estimated cycles, use the **Sample Rate** drop-down menu to set the sample rate to zero.

   ETM overflows during an ARM Profiler run usually indicate that your target is running so fast that it is swamping the ETM's output buffer with data, causing your target to fail to collect that data. This creates holes in your profile, the severity of which depend on how many overflows occur. Normally these overflows can be resolved by reducing the clock speed of your target processor, but there are cases where reducing the processor speed also reduces the speed of the ETM. In these cases, the reduction of clock speed has no impact on the amount of ETM overflows. Solutions to this problem vary widely from target to target. Contact ARM support for more information.

3. If the ARM Profiler terminates with the error message that reads "`Exiting application due to unrecoverable errors please refer to the ARM Profiler user guide Appendix C hardware troubleshooting guide`", one of the following issues is likely the cause.

   - The image file specified in the ARM Profiler launch configuration does not match the image on the target hardware.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

If the image specified in the launch configuration does not match the one loaded on the development board, the image can not run on the target with profiling turned on. This can often happen when using the **Symbols Only** option, but no specific error is generated in this case. Make sure the image file matches the one on the target and try hardware profiling again.

- Self-modifying code was used

  The ARM Profiler may fail with the above error if you have self-modifying code that attempts to branch to an address that does not match the behavior in the axf file. Remove any self-modifying code and try hardware profiling again.

  Note

  There is an exception to this. The ARM Profiler does allow self-modifying code that installs exception handlers at run time at address offsets 0x0000 to 0x001C.

  The ARM Profiler does not support profiling through certain abort handlers. Some handlers, like the prefetch abort handler, cause failure with the above error message.

- The hardware target is running too fast

  Try reducing the speed of the hardware for the purpose of profiling and try again.

- Code contained in the image file is defective

  It is important that you verify your code is stable before running it through the ARM Profiler.

- If your hardware has delays on the trace port pins, use the Auto-Calibrate feature to center the clock against the data.

4. Either the RVI or RealView Trace 2 unit may be in a bad state if the ARM Profiler fails with one of the following error messages:

   - `"Session handle invalid"`
   - `"Device state requested on connection could not be achieved"`
   - `"RVI timed out waiting for a response from the device"`

   The RVI or RealView Trace 2 unit may be in a bad state. To fix this, try power cycling the RVI and RealView Trace 2 units. First, power down the development board, followed by the RVI and RealView Trace 2 units. Then, turn the development board back on. Next, restart the RVI and RealView Trace 2 units and wait until the STAT LED on the RVI unit stays on and the Power On LED indicator on the RealView Trace 2 unit is lit. When all of the hardware is up and running again, power cycle the development board again and try another profiling run.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

5. If the ARM Profiler fails with a "`No connection to the device`" error message, try the following:
   - Make sure that your RVConfig file matches your development board.
   - Check the connections to and from the RVI and RealView Trace 2 units.
   - Make sure both the RVI and hardware target are powered on.

   If all of the above are correct and you still see the "`No connection to the device`" error, it is possible that the device has not yet obtained an IP address. Try power cycling the RVI unit as described in step two.

6. The "`Corrupted trace`" warning message might indicate that your target has imperfect timing characteristics. Try turning on the **Auto-Calibrate Timings** option in the **Connections** tab of the launch configuration dialog. For more information on the Auto-Calibrate Timings option, see *Auto-calibrate timings and using a TPIU pattern generator* on page 3-7.

   Note

   You must provide an executable to run Auto-Calibrate. If you choose to run the Dhrystone example, set the iteration count to one billion. This gives the ARM Profiler enough time to complete the auto-calibration of your hardware.

7. If the ARM Profiler fails with the "`Device in use`" error message, it may be due to an active RealView Debugger connection to the unit. Only one connection can be used with the RVI and RealView Trace 2 units at a time, so if RVD is already connected, try closing RVD and running the ARM Profiler again. This error can also occur if another instance of the ARM Profiler is already connected to the device. Check to make sure you only have a single instance of the ARM Profiler open.

8. The "`Bad plugin`" error message indicates the specified '.rvc' configuration file is not present. Check to make sure the configuration is in the directory location specified in the RealView Trace 2 launch configuration dialog.

9. The "`Failed to parse the specified configuration file`" indicates a bad or invalid '.rvc' configuration file. Create a new, valid '.rvc' configuration file and try running hardware profiling again.

   Note

   RVConfig files must be created using RVI version 3.3 or greater. RVConfig files created using older versions of RVI are not compatible with the ARM Profiler

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

10. The "`Buffer overflow`" error message may also indicate that the hardware target is running too fast or one of the units is in a bad state. See items one and two for potential fixes for these issues.

11. If the ARM Profiler fails with the error message "`Device not powered or has been disconnected`", check the connections and power status of the RVI and RealView Trace 2 units.

12. The "`Error in Loading the Object Image <path> execution terminated`" indicates a missing or invalid image file. Make sure the launch configuration dialog points to a valid image file and try hardware profiling again.

13. The error message "`read/write memory failure`" might be caused by a bad configuration of the target hardware's memory map. Configure the memory map per your hardware specification to correct this issue.

14. If your `.apd` analysis files are being created with zero instructions reported and you are running on a board that requires the use of an inverted clock to capture trace at high speeds, try one of the following fixes:

    - Check to make sure the probe is securely connected to the right slot.
    - If your hardware uses an inverted clock, make sure the Inverted Clock setting is used in your RVConfig file.

    Note

    Many settings in the RVConfig, including trace timing settings, are not honored by the ARM Profiler, but the Inverted Clock setting is an exception to this rule.

15. Some indirect branch errors are caused by an application writing the vector table as part of the bootstrap. Due to limitations in some ARM breakpoint units, only a subset of writes to the vector table can be caught. If your program writes the vector table as part of the setup, make sure that the final write to the vector table occurs at one of the following locations:

    - 0x08 (SVC)
    - 0x18 (IRQ)

    Note

    If your application writes the vector table, it is advised that two NOPs are appended after the vector table write to avoid misreporting links in the call chain for certain targets.

If you are having an issue with hardware profiling not listed here, try power cycling the RVI and RealView Trace 2 units as explained above and re-starting hardware profiling.

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007- 2010 ARM Limited. All rights reserved.
Non-Confidential