RealView® Development Suite

Version 4.1

Getting Started Guide

ARM®

ARM DUI 0255K

RealView Development Suite
Getting Started Guide

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
September 2003	A	Non-Confidential	ARM® RealView® Developer Suite v2.0 Release
January 2004	B	Non-Confidential	RealView Developer Suite v2.1 Release
December 2004	C	Non-Confidential	RealView Developer Suite v2.2 Release
May 2005	D	Non-Confidential	RealView Developer Suite v2.2SP1 Release
March 2006	E	Non-Confidential	RealView Development Suite v3.0 Release
March 2007	F	Non-Confidential	RealView Development Suite v3.1 Release
February 2008	G	Non-Confidential	RealView Development Suite v3.1 Professional Release
September 2008	H	Non-Confidential	RealView Development Suite v4.0 Release
June 2009	I	Non-Confidential	RealView Development Suite v4.0 SP2 Release
6 November 2009	J	Non-Confidential	RealView Development Suite v4.0 SP3 Release
28 May 2010	K	Non-Confidential	RealView Development Suite v4.1 Release

Proprietary Notice

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.

Product Status

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

Web Address

`http://www.arm.com`

Contents
RealView Development Suite Getting Started Guide

Preface

This preface introduces the ARM® *RealView*® *Development Suite Getting Started Guide*, that shows you how to start using ARM *RealView Development Suite* (RVDS) to manage software projects and to debug your application programs. It contains the following sections:

About this book

RVDS provides tools for building, debugging, and managing software development projects targeting ARM architecture-based processors. This book contains:

- an introduction to the software components that make up RVDS
- a summary of the differences between RVDS v4.1 and previous versions of RVDS
- a glossary of terms for users that are new to RVDS.

See also:

- *Intended audience*
- *Using this book*
- *Typographical conventions* on page ix
- *Further reading* on page ix.

Intended audience

This book has been written for developers who are using RVDS to manage development projects for ARM architecture-based processors. It assumes that you are an experienced software developer, but might not be familiar with the ARM development tools.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for an introduction to RVDS components, licensing, and documentation.

Chapter 2 ***Getting Started with RealView Development Suite***

Read this chapter for an overview of the main tasks that you can do with the RVDS tools. It also describes the example projects provided with RVDS.

Chapter 3 ***Changes to RealView Development Suite***

Read this chapter for a description of the changes between RVDS v4.1 and RVDS v4.0 SP3.

Appendix A ***Using the armenv Tool***

Read this appendix for a description of how to use the `armenv` tool.

Appendix B ***About Previous Releases***

Read this chapter for a description of previous RVDS versions.

Glossary An alphabetically arranged definition of terms used in the RVDS documentation.

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.
`mono`space	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
`...`	At the end of a path name `...` denotes that the directories of interest are below the last-specified directory name. The unspecified path names are usually those that are different between operating systems. For example: *`install_dir`*`\ARM\RVDS\Examples\...` In the middle of a path name `...` denotes that additional directories exist between the directory names specified. The unspecified path names are usually version and build numbers and platform-specific directory names. For example: *`install_dir`*`\ARM\RVD\Core\...\etc`

Further reading

This section lists publications from both ARM and third parties.

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

ARM publications

See the following document for information on the FLEX*net* license management system that controls the use of ARM applications:

- *FLEXnet for ARM® Tools License Management Guide* (ARM DUI 0209).

Note

Make sure that you use version 4.3 or later of this document for information on license management in RVDS v4.1.

Note

The FLEX*net* license management system is owned by Flexera Software Inc.

This book is part of the RVDS documentation set. Other books in this suite include:

- *ARM® Workbench IDE User Guide* (ARM DUI 0330)
- *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 Using the Assembler* (ARM DUI 0473)
- *ARM® Compiler toolchain Assembler Reference* (ARM DUI 0489)
- *ARM® Compiler toolchain Using the Compiler* (ARM DUI 0472)
- *ARM® Compiler toolchain Building Linux Applications with ARM® Compiler toolchain and GNU Libraries* (ARM DUI 0483)
- *ARM® Compiler toolchain Compiler Reference* (ARM DUI 0491)
- *ARM® Compiler toolchain Using ARM® C and C++ Libraries and Floating-Point Support* (ARM DUI 0475)
- *ARM® Compiler toolchain ARM® C and C++ Libraries and Floating-Point Support Reference* (ARM DUI 0492)
- *ARM® Compiler toolchain Using the Linker* (ARM DUI 0474)
- *ARM® Compiler toolchain Linker Reference* (ARM DUI 0493)
- *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® 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)
- *RealView® Development Suite Real-Time System Model User Guide* (ARM DUI 0424)
- *RealView® ARMulator® ISS User Guide* (ARM DUI 0207).

For full information about the software interfaces and standards supported by ARM, see *install_dir*\Documentation\Specifications\....

In addition, see the following documentation for specific information relating to ARM products:

- *DSTREAM Setting Up the Hardware* (ARM DUI 0481)
- *DSTREAM System and Interface Design Reference* (ARM DUI 0499)
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities* (ARM DUI 0498)
- *RealView® ICE and RealView Trace Setting Up the Hardware* (ARM DUI 0515)
- *RealView® ICE and RealView Trace System and Interface Design Reference* (ARM DUI 0517)
- *ARM® Architecture Reference Manual, ARMv7-A™ and ARMv7-R™ edition* (ARM DDI 0406)
- *ARM7-M™ Architecture Reference Manual* (ARM DDI 0403)
- *ARM6-M™ Architecture Reference Manual* (ARM DDI 0419)
- *ARM® Reference Peripheral Specification* (ARM DDI 0062)

- ARM datasheet or technical reference manual for your hardware device.

Other publications

For an introduction to ARM architecture, see Andrew N. Sloss, Dominic Symes and Chris Wright, *ARM System Developer's Guide: Designing and Optimizing System Software* (2004). Morgan Kaufmann, ISBN 1-558-60874-5.

For an essential handbook for system-on-chip designers using ARM processors and engineers working with the ARM architecture, see Steve Furber, *ARM system-on-chip architecture* (2nd edition, 2000). Addison Wesley, ISBN 0-201-67519-6.

Feedback

ARM welcomes feedback on both RVDS and its documentation.

See also:

- *Feedback on RealView Development Suite*
- *Feedback on this book*.

Feedback on RealView Development Suite

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

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

You can also use the ARM Support Wizard to report any problems with RVDS to ARM Support. See *ARM Support Wizard* on page 1-11 for more information.

Note

If you have any problems with RealView Debugger, it is recommended that you create a Software Problem Report. To do this, select **Send a Problem Report...** from the RealView Debugger **Help** menu. See the RealView Debugger online help for more information.

Feedback on this book

If you have any comments on this book, send email to `errata@arm.com` giving:

- the title
- the number, ARM DUI 0255K
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter describes the ARM® *RealView® Development Suite* (RVDS) v4.1 component applications, the additional licenses you can purchase to extend the features of RVDS, and gives an overview of the documentation suite. It contains the following sections:

1.1 RealView Development Suite components

RVDS provides a coordinated development environment for embedded systems applications running on the ARM family of processors. It consists of a suite of tools, together with supporting documentation and examples. The tools enable you to write, build, and debug your applications, either on target hardware or software simulators.

There are two versions of RVDS:

- Standard edition includes:
  - ARM Workbench IDE
  - ARM Compiler toolchain
  - RealView Debugger
  - DSTREAM and RealView ICE host software.
- Professional edition includes:
  - all the components in the Standard edition
  - ARM Profiler, and a license for its use
  - a license for NEON vectorizing compiler support
  - Real-Time System Models.

See also:

- *RVDS installation, examples, and documentation directories*
- *Host platform support* on page 1-3
- *Processor support* on page 1-4
- *Simulator support* on page 1-5
- *ARM Workbench IDE* on page 1-7
- *ARM Compiler toolchain* on page 1-7
- *RealView Debugger* on page 1-8
- *ARM Profiler* on page 1-9
- *DSTREAM and RealView ICE host software* on page 1-10
- *ARM Support Wizard* on page 1-11.

1.1.1 RVDS installation, examples, and documentation directories

Various directories are installed with RVDS that contain example code and documentation. The RVDS documentation refers to these directories where required.

The main installation, examples, and documentation directories are identified in Table 1-1. The *install_dir* shown is the default installation directory If you specify a different installation directory, then the path names are relative to your chosen directory.

Table 1-1 RealView Development Suite directories

Directory	Windows default path	Red Hat Linux default path
install_dir	C:\Program Files\ARM	~/arm
Examples	*install_dir*\RVDS\Examples\...	*install_dir*/RVDS/Examples/...
ARM Profiler examples (RVDS Professional edition only)	*install_dir*\Profiler\...\examples\...	*install_dir*/Profiler/.../examples/...
Real-Time System Model path	*install_dir*\RVDS\Models\...\lib\...	*install_dir*\RVDS\Models\...\lib\...
Project templates	*install_dir*\project_templates\...	*install_dir*/project_templates/...
RealView Debugger Flash examples	*install_dir*\RVD\Flash\...	*install_dir*/RVD/Flash/...
Documentation	*install_dir*\Documentation\... and *install_dir*\Documentation_*component_N.n*\...	*install_dir*/Documentation/... and *install_dir*/Documentation_*component_N.n*/...

See also

- *RealView Development Suite documentation* on page 1-15 for more information on accessing the documentation.
- the following for a summary of the examples provided:
  - *RealView Development Suite examples* on page 1-18
  - *ARM Profiler examples (RVDS Professional edition only)* on page 1-21.

1.1.2 Host platform support

RVDS supports the following OS platforms:

- Windows 7 Enterprise Edition
- Windows 7 Professional Edition
- Windows XP Professional SP3
- Windows Vista Business Edition SP2

- Windows Vista Enterprise Edition SP2
- Windows Server 2003 (ARM Compiler toolchain only)
- Solaris 10 (ARM Compiler toolchain only)
- Red Hat Enterprise Linux WS version 4 for x86 using GNOME Window Manager and Bash Shell
- Red Hat Enterprise Linux WS version 5 for x86 using GNOME Window Manager and Bash Shell.

All tools support both 32-bit and 64-bit versions of these operating systems, where available, with the following exceptions:

- RealView ICE does not support 64-bit versions of Red Hat Linux hosts, nor installing 64-bit USB drivers on Windows Vista.
- ARM Profiler does not support profiling on hardware when running on Red Hat Linux hosts. You can use the Real-Time System Models to profile on Red Hat Linux hosts.

See also

- *Simulator support* on page 1-5
- *ARM Profiler* on page 1-9
- *DSTREAM and RealView ICE host software* on page 1-10.

1.1.3 Processor support

RVDS supports the following processors:

- ARM7™, ARM9™, and ARM11™ processor families

  Note

  The ARM10™ processor family is deprecated.

- ARM11 MPCore™ multicore processor
- Cortex™ family of processors

  Note

  A license is provided with RVDS Professional edition to support the Cortex-A9 processor.

- SecurCore® SC100™ and SC200™ processors in RealView Debugger

- SecurCore SC300™ processor in ARM Compiler toolchain
- Faraday FA526, FA626, and FA626TE processors are supported in RealView Debugger
- Marvell Feroceon 88FR101 and 88FR111 processors (added to RealView Debugger, in addition to ARM Compiler toolchain)
- Marvell Sheeva 88SV581x-v7 PJ4.

1.1.4 Simulator support

RVDS supports the following simulators:

- *RealView ARMulator® Instruction Set Simulator* (RVISS)
- *Instruction Set System Model* (ISSM)
- *Real-Time System Model* (RTSM)
- SoC Designer.

RealView ARMulator Instruction Set Simulator

RVISS simulates the instruction sets and architectures of ARM7, ARM9, ARM10, and ARM11 processor families, together with a memory system and peripherals.

Note

The ARM10 processor family is deprecated.

RVISS enables you to begin developing and debugging your embedded applications without target hardware. This is useful where hardware is still being developed, or if there is a limited number of development boards available.

Instruction Set System Model

ISSM simulates the instruction sets and architecture of the Cortex family of ARM processors:

- Cortex-A8
- Cortex-M0
- Cortex-M1
- Cortex-M3
- Cortex-R4.

Real-Time System Models

The following RTSMs are provided with RVDS Professional edition:

- RTSM *Emulation Baseboard* (EB) with ARM926EJ-S™
- RTSM EB with ARM1136JF-S™
- RTSM EB with ARM1176JZF-S™
- RTSM EB with Cortex-A5_MPx1
- RTSM EB with Cortex-A5_MPx2
- RTSM EB with Cortex-A8
- RTSM EB with Cortex-A9_MPx1
- RTSM EB with Cortex-A9_MPx2
- RTSM EB with Cortex-R4
- RTSM *Microcontroller Prototyping System* (MPS) with Cortex-M3.

These models can be used with ARM Profiler and RealView Debugger.

Note

Be aware that EB RTSMs are not intended to be software implementations of particular revisions of EB hardware.

Note

To create your own RTSMs, you must purchase Fast Model Tools from ARM.

SoC Designer

RealView Debugger provides a SoC Designer Debug Interface. This interface enables you to configure connections to SoC Designer models when they are opened in Carbon SoC Designer Simulator. When you attempt to connect to a target processor in a SoC Designer model, RealView Debugger can automatically open Carbon SoC Designer Simulator with the model containing that target processor.

No SoC Designer models are provided with RVDS.

Note

You must purchase the Carbon SoC Designer software separately.

See also

- *RealView® ARMulator® ISS User Guide*
- *RealView® Debugger Target Configuration Guide*.

- *RealView® Development Suite Real-Time System Model User Guide*
- *ARM® Workbench IDE User Guide*
- *ARM® Profiler User Guide*
- Carbon SoC Designer Plus, `http://carbondesignsystems.com/Products/SoCDesigner.aspx`.

1.1.5 ARM Workbench IDE

ARM Workbench is an *Integrated Development Environment* (IDE) that combines software development with the compilation and profiling technology of the RealView tools. You can use it as a project manager to create, build, and manage projects for ARM targets. It uses a single folder called a workspace to store files and folders related to specific projects.

See also

- *ARM® Workbench IDE User Guide*.

1.1.6 ARM Compiler toolchain

You can use the ARM Compiler toolchain to build programs from C, C++, or ARM assembly language source. The ARM Compiler toolchain comprises the following:

- ARM and Thumb™ C and C++ compiler, `armcc`
- NEON™ vectorizing compiler, invoked using the command `armcc --vectorize`.
- ARM and Thumb assembler, `armasm`
- linker, `armlink`
- ARM librarian, `armar`
- ARM image conversion utility, `fromelf`
- supporting libraries.

Note

A license is provided with RVDS Professional edition to enable you to use the NEON vectorizing compiler.

See also

- *RealView Development Suite licensing* on page 1-12
- *RealView Development Suite documentation* on page 1-15 for more information on accessing the documentation

- *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain* for more information on the tools and features available
- the ARM web site for updates and patches to the ARM Compiler toolchain as they become available.

1.1.7 RealView Debugger

RealView Debugger together with a supported debug target, enables you to debug your application programs and have complete control over the flow of the program execution so that you can quickly isolate and correct errors. See *Debug Interface support in RealView Debugger* on page 1-22 for more information.

Note

For information specific to using RealView Debugger on Red Hat Linux see the appendix that describes RealView Debugger on Red Hat Linux. You can find this appendix in the *RealView® Debugger User Guide*.

RealView Debugger includes support for:

- multiprocessor debugging
- trace, analysis and profiling
- *Operating System* (OS) awareness.

The default license for RealView Debugger enables you to debug applications that run on single or multiple ARM architecture-based processors.

RealView Debugger downloads

Downloads are available from the ARM web site that enable you to use supported plug-ins to debug your OS aware applications, and obtain software updates and utilities.

To access the RealView Debugger downloads, from RealView Debugger select:

Help → ARM on the Web → Goto RTOS Awareness Downloads

This displays the *OS-Aware and Middleware Debug* web page on the ARM web site. From here you can locate and download the OS plug-in of your choice.

Help → ARM on the Web → Goto Update and Utility Downloads

This displays the *ARM Technical Support - Downloads* web page on the ARM web site. From here you can locate and download any ARM software updates and utilities.

See also

- *RealView Development Suite licensing* on page 1-12 for more information on licensing
- *RealView Development Suite documentation* on page 1-15 for more information on accessing the documentation
- the *RealView® Debugger Essentials Guide* for more information on the features available in RealView Debugger.

1.1.8 ARM Profiler

ARM Profiler is a plug-in to the ARM Workbench IDE. It enables you to see how your code performs on a target system, either:

- by observing your code on target hardware using RealView ICE in combination with RealView Trace 2
- by testing code against an RTSM.

Note

Profiling with DSTREAM is not supported.

When execution of your application stops, ARM Profiler produces an analysis file containing detailed information on the executed code, such as call sequences for various functions, timing characteristics, cycle counts, and instruction counts.

If you have RVDS Professional edition, then ARM Profiler is installed with the **Full** product selection. A license to use ARM Profiler is also included.

If you have RVDS Standard edition, then you can purchase ARM Profiler separately. You must also obtain an ARM Profiler license.

See also

- *DSTREAM and RealView ICE host software* on page 1-10
- *ARM® Workbench IDE User Guide*
- *ARM® Profiler User Guide*
- *DSTREAM and RealView® ICE Using the Debug Hardware Configuration Utilities*.

1.1.9 DSTREAM and RealView ICE host software

A DSTREAM or RealView ICE run control unit provides the interface between your target hardware and the debug and analysis tools provided with RVDS:

- Using DSTREAM and RealView Debugger you can:
  - debug your applications running on your target hardware
  - analyze trace captured in an *Embedded Trace Buffer*™ (ETB™), if present.

  Note

  RealView Debugger does not support tracing from the external trace port of a SoC with DSTREAM.

- Using RealView ICE and RealView Debugger you can:
  - debug your applications running on your target hardware
  - analyze trace captured in an *Embedded Trace Buffer*™ (ETB™), if present.

  With the addition of a RealView Trace or a RealView Trace 2 data capture unit, you can also capture and analyze trace directly from an *Embedded Trace Macrocell*™ (ETM™).

- To perform hardware profiling with ARM Profiler you require a RealView ICE run control unit connected to the host using TCP/IP or USB, together with a RealView Trace 2 data capture unit connected to the host using USB.

  Note

  Profiling with DSTREAM is not supported.

The latest version of the DSTREAM and RealView ICE host software available at the time of this RVDS release is installed with the **Full** product selection.

A GDB plug-in is also provided for the ARM Workbench IDE. It enables you to configure DSTREAM or RealView ICE for use with GDB.

See also

- *RealView Debugger* on page 1-8
- *ARM Profiler* on page 1-9
- *DSTREAM and RealView® ICE Using the Debug Hardware Configuration Utilities*
- *DSTREAM Setting Up the Hardware*
- *RealView® ICE and RealView Trace Setting Up the Hardware*
- *ARM® Profiler User Guide*

- *ARM® Workbench IDE User Guide*
- *RealView® Debugger User Guide*
- *RealView® Debugger Target Configuration Guide*.

1.1.10 ARM Support Wizard

The ARM Support Wizard is available from:

- **Start → All Programs → ARM → Support Wizard** menu on Windows
- **Start Menu → Programs → ARM → Support Wizard** menu on Red Hat Linux.

The wizard gathers information about your installed ARM products, and enables you to save the report to a file or e-mail the report to ARM Support. You must include a description of the problem with the report before sending it to ARM Support.

1.2 RealView Development Suite licensing

All licensing for RVDS is controlled by the FLEX*net* license management system. RVDS supports both floating and node-locked licenses. The type of license you can use depends on the type of license you purchased. You can obtain your license from the *ARM Web Licensing Portal*, `https://license.arm.com`. See the *FLEXnet for ARM® Tools License Management Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/index.html` for information on how to request, install, and use your license.

See also:

- *ARM Profiler license*
- *Profiler-guided optimization license*
- *NEON vectorizing compiler license* on page 1-13
- *Installing a node-locked license with the ARM License Wizard* on page 1-13
- *FLEXnet files and documentation provided with RVDS* on page 1-14
- *Changes to FLEXnet licensing in RVDS 4.1* on page 3-2.

1.2.1 ARM Profiler license

The ARM Profiler license enables you to use the ARM Profiler to analyze the performance of your code through runtime profiling.

Note

A license to use the ARM Profiler is provided with RVDS Professional edition.

See also

- *FLEXnet for ARM® Tools License Management Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/index.html`
- *ARM® Profiler User Guide*

1.2.2 Profiler-guided optimization license

The Profiler-guided optimization license enables the compiler to perform optimizations baesd on a data file produced by the ARM Profiler.

Note

A license to use the Profiler-guided optimizations is provided with RVDS Professional edition.

See also

- *FLEXnet for ARM® Tools License Management Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/index.html`
- *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain*
- *ARM® Compiler toolchain Using the Compiler*.

1.2.3 NEON vectorizing compiler license

The NEON vectorizing compiler license enables the compiler to generate NEON instructions whenever appropriate to target ARM processors with a NEON unit, for example Cortex-A8 or Cortex-A9.

Note

A license to use the NEON vectorizing compiler is provided with RVDS Professional edition.

See also

- *FLEXnet for ARM® Tools License Management Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/index.html`
- *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain*
- *ARM® Compiler toolchain Using the Compiler*
- *ARM® Compiler toolchain Compiler Reference*.

1.2.4 Installing a node-locked license with the ARM License Wizard

To use the ARM License Wizard to install a node-locked license:

1. Start the ARM License Wizard from the Start menu:

   Start → Programs → ARM → License Wizard v4.x

2. Follow the on-screen instructions. See the *FLEXnet for ARM® Tools License Management Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/index.html` for more details.

1.2.5 FLEXnet files and documentation provided with RVDS

RVDS supports floating licensing and provides the software for all supported license servers. The FLEX*net* files can typically be found in the following places:

- On the RVDS DVD-ROM or in the downloaded package:

  Windows `Utilities\FLEXnet\version\release\win_32-pentium`

  Solaris `Utilities/FLEXnet/version/release/solaris-sparc`

  Linux `Utilities/FLEXnet/version/release/linux-pentium`

  Linux `Utilities/FLEXnet/version/release/linux-pentium-rh72`

- In the a folder of your installation:

  Windows *`install_dir`*`\Utilities\FLEXnet\version\release\win_32-pentium`

  Solaris *`install_dir`*`/Utilities/FLEXnet/version/release/solaris-sparc`

  Linux *`install_dir`*`/Utilities/FLEXnet/version/release/linux-pentium`

  Linux *`install_dir`*`/Utilities/FLEXnet/version/release/linux-pentium-rh72`

If you cannot locate the ARM license server utilities, either:

- download the files from the ARM support site at:
  `http://www.arm.com/support/downloads/flexnet.html`
- contact ARM License Support by email at:
  `license.support@arm.com`

Other FLEXnet documentation

The *FLEXnet End User Guide* is supplied as a PDF. A link to the document is provided in the ARM product grouping of the Windows Start menu. If this link is not present, the PDF is normally located at one of the following locations:

- *`install_dir`*`\Utilities\FLEXlm\version\release\`
  `flexnet_licensing_end_user_guide.pdf`
- *`install_dir`*`\Utilities\FLEXlm\version\release\enduser.pdf`
- *`install_dir`*`\Documentation_FLEXlm_version\PDF\enduser.pdf`
- *`install_dir`*`\licensing\doc\FLEXlm_EndUserGuide.pdf`.

On Unix or Linux platforms, reverse the direction of the slash character.

1.3 RealView Development Suite documentation

The following documentation is provided with RVDS:

- *RealView® Development Suite Installation Guide*
- *RealView® Development Suite Getting Started Guide* (this document)
- *ARM® Workbench IDE User Guide*
- *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain*
- *ARM® Compiler toolchain Developing Software for ARM® Processors*
- *ARM® Compiler toolchain Building Linux Applications with ARM® Compiler toolchain and GNU Libraries*
- *ARM® Compiler toolchain Using the Assembler*
- *ARM® Compiler toolchain Assembler Reference*
- *ARM® Compiler toolchain Using the Compiler*
- *ARM® Compiler toolchain Compiler Reference*
- *ARM® Compiler toolchain Using C and C++ Libraries and Floating-Point Support*
- *ARM® Compiler toolchain C and C++ Libraries and Floating-Point Support Reference*
- *ARM® Compiler toolchain Using the Linker*
- *ARM® Compiler toolchain Linker Reference*
- *ARM® Compiler toolchain Creating Static Software Libraries with armar*
- *ARM® Compiler toolchain Using the fromelf Image Converter*
- *ARM® Compiler toolchain Errors and Warnings Reference*
- *ARM® Compiler toolchain Migration and Compatibility*
- *ARM® Profiler User Guide*
- *RealView® Debugger Essentials Guide*
- *RealView® Debugger User Guide*
- *RealView® Debugger Target Configuration Guide*
- *RealView® Debugger Trace User Guide*
- *RealView® Debugger RTOS Guide*
- *RealView® Debugger Command Line Reference Guide*
- *DSTREAM Setting Up the Hardware*
- *DSTREAM System and Interface Design Reference*
- *DSTREAM and RealView® ICE Using the Debug Hardware Configuration Utilities*
- *RealView® ICE and RealView Trace Setting Up the Hardware*
- *RealView® ICE and RealView Trace System and Interface Design Reference*

- *RealView® Development Suite Real-Time System Models User Guide*
- *RealView® ARMulator® ISS User Guide*.

A glossary of ARM terms used in the RVDS documentation is provided. See *Glossary* on page Glossary-1 at the end of this book.

Note

To view the PDF documentation, you must install and use Adobe Acrobat Reader.

See also:

- *Getting more information online*
- the *Additional reading* sections in each book for related publications.

1.3.1 Getting more information online

Depending on your installation, the full documentation suite is available in HTML and PDF format:

- View the HTML documentation on the ARM Information Center at *RealView Development Suite*, `http://infocenter.arm.com/help/topic/com.arm.doc.subset.swdev.rvds`.
- View the HTML documentation from within the ARM Workbench IDE:
  - on Windows, select:
    Start → All Programs → ARM → ARM Workbench IDE v4.0
  - on Red Hat Linux, select:
    Start Menu → Programs → ARM → ARM Workbench IDE v4.0.

  In the ARM Workbench IDE, select **Help Contents** from the **Help** menu.

  This displays Help browser where you can:
  - view the RVDS documentation in HTML format
  - perform text searches on all documents or a subset of documents
  - access the corresponding PDF file for each document.

  Note

  You cannot search all PDF documentation when viewing PDF documents from the ARM Workbench IDE Help browser.

- Depending on your platform, to view the PDF documentation:
  - on Windows, select:

Start → All Programs → ARM → RealView Development Suite v4.1 → RVDS v4.1 Documentation Suite

— on Red Hat Linux, select:

Start Menu → Programs → ARM → RealView Development Suite v4.1 → RVDS v4.1 Documentation Suite.

This displays a PDF document containing links to the RVDS documentation in PDF format. You can also perform text searches on the PDF documentation.

See also

- *RVDS installation, examples, and documentation directories* on page 1-2.

1.4 RealView Development Suite examples

The code for many of the examples in the RVDS documentation is located in the main examples directory. See *RVDS installation, examples, and documentation directories* on page 1-2 for more information.

In addition, the directory contains example code that is not described in the documentation. Read the `readme.txt` in each example directory for more information. The examples are installed in the following subdirectories:

`asm` Some examples of ARM assembly language programming. The examples are used in *ARM® Compiler toolchain Using the Assembler* and *ARM® Compiler toolchain Assembler Reference*.

`cached_dhry` Examples of routines to initialize cache and tightly coupled memory on various ARM processors, built around the Dhrystone example. The supported processors include:

- ARM9xx processors
- ARM11xx processors
- Cortex-A5
- Cortex-A8
- Cortex-A9
- Cortex-R4.

`Cortex-M0` Examples for the ARM Cortex-M0 processor, that include example scatter files and build scripts.

`Cortex-M1` Examples for the ARM Cortex-M1 processor, that include example scatter files and build scripts.

`Cortex-M3` Examples for the ARM Cortex-M3 processor, that include example scatter files and build scripts.

`cpp` Some basic C++ examples.

`databort` Design documentation and example code for a standard Data Abort handler.

`dcc` Example code that demonstrates how to use the Debug Communications Channel. The code for this example is described in *ARM® Compiler toolchain Developing Software for ARM® Processors*.

`dhrystone` The Dhrystone Benchmark. The example is used in the RealView Debugger documentation.

`dsp` This example demonstrates the use of the *European Telecommunications Standard Institute* (ETSI) basic operations provided in `dspfns.h`.

`emb_sw_dev` The example projects referenced in the chapter that describes embedded software development in the *ARM® Compiler toolchain Developing Software for ARM® Processors*.

`fft_v5te` *Fast Fourier Transform* (FFT) code optimized for *ARM architecture v5TE* (ARMv5TE™).

`iMx31_RTC` An example for the Freescale Zoom i.Mx31 LiteKit, that demostrates the *Real Time Clock* (RTC) of the Freescale i.Mx31 processor.

`interwork` Examples that show how to interwork between ARM code and Thumb code. See the chapter that describes interworking ARM and Thumb in *ARM® Compiler toolchain Developing Software for ARM® Processors* for more information.

`linux_apps` Examples that demonstrate the interoperation between the ARM Compiler toolchain and the GNU toolchain and GNU libraries, for building applications and shared libraries to run on Linux. See *ARM® Compiler toolchain Building Linux Applications with ARM® Compiler toolchain and GNU Libraries*.

`mandelbrot` The Mandelbrot example `brot.c`, that is referenced from the *RealView® Development Suite Real-Time System Model User Guide*.

`mmugen` The source and documentation for the MMUgen utility. This utility can generate MMU pagetable data from a rules file that describes the virtual to physical address translation required.

`picpid` An example of how to write position-independent code.

`sorts` Example code that compares an insertion sort, shell sort, and the quick sort used in the ARM C libraries.

`svc` An example *Supervisor Call* (SVC) handler.

`trace` An example application `trace.c` that is used in the tracing tutorial described in the *RealView® Debugger Trace User Guide*. The application:

- simulates a small system that reads a set of input data samples and computes the sample average
- provides a framework for common instruction and data trace scenarios.

`unicode` Example code that enables you to evaluate multibyte character support.

`vfpsupport` Example code for enabling and carrying out VFP operations. Also included are various utility files for configuring the debug system when using VFP, and a PDF of *Application Note 133 Using VFP in RVDS.*

1.5 ARM Profiler examples (RVDS Professional edition only)

ARM Profiler is provided as an option during the RVDS Professional edition installation. If you installed ARM Profiler, the following examples are also installed:

`ARM` A common set of source files required to build the other Profiler projects. The `makefile` for each of the other projects includes the `common.make` file from this directory.

`doom` Example code that runs Doom.

Note

You must download an external shareware file before you can successfully compile and run this example. See the *ARM® Profiler User Guide* for more information.

`fireworks` Example code that produces a simulation of exploding fireworks.

`fft` Example code that runs a FFT.

`xvid` Example code that displays a video encoded in MPEG4.

See *RVDS installation, examples, and documentation directories* on page 1-2 for the location of the ARM Profiler examples directory.

1.6 Debug Interface support in RealView Debugger

The Debug Interfaces supported by RealView Debugger in RVDS are shown in Table 1-2.

Table 1-2 RealView Debugger Debug Interfaces supported in RVDS v4.0

Debug Interface	Description
DSTREAM	Includes support for connections to target hardware through a DSTREAM run control unit, and forwarding trace from an ETB to the RealView Debugger.
Instruction Set System Model (ISSM) (deprecated)	Connections to simulated Cortex targets.
Model Library	Connections to a *Cycle Accurate Debug Interface* (CADI) model defined in a model library file.
Model Process	Connections to a CADI model that is currently running.
Real-Time System Model (RTSM)	Connections to a selection of simulated ARM Versatile EB targets.
RealView ARMulator ISS (RVISS) (deprecated)	Connections to simulated ARM7, ARM9, and ARM11 targets.
RealView ICE	Includes support for: • connections to target hardware through a RealView ICE run control unit • tracing with RealView ICE in conjunction with either a RealView Trace or RealView Trace 2 data capture unit.
SoC Designer	Connections to simulated targets in single or multiprocessor systems created with the Carbon SoC Designer Plus software.

Be aware of the following:

- To create SoC Designer connections, you must purchase and install Carbon SoC Designer Plus.
- To trace using RealView Trace or RealView Trace 2, you must purchase the corresponding product.
- You can use RealMonitor with DSTREAM or RealView ICE in RealView Debugger.

See also:

- *RealView® Debugger Target Configuration Guide* for more information on using RealMonitor with DSTREAM or RealView ICE.

1.7 Fixing problems with your RVDS environment

If you are having problems running the component applications in RVDS, then make sure your RVDS environment is correctly configured:

- On Red Hat Linux, source the `RVDS41env.posh` script. This is the preferred method of setting up the RVDS environment on Red Hat Linux.
- On Windows, you can use the `armenv` utility to modify the RVDS environment after installation.

Note

You cannot use the `armenv` utility on custom installations. If you performed a custom installation on Windows, you must set the environment variables yourself. On Red Hat Linux, use the `RVDS41env.posh` script.

See also:

- *RVDS environment variables*
- Appendix A *Using the armenv Tool*
- *RealView® Development Suite Installation Guide* for more information on running the `RVDS41env.posh` script.

1.7.1 RVDS environment variables

Table 1-3 shows the main RVDS environment variables that must be set on Windows. Replace `...` with the path elements of your installation. Use the preferred methods described in *Fixing problems with your RVDS environment* to set these, if possible. Also, make sure that your `PATH` environment variable includes the locations of the various RVDS component application executables.

Table 1-3 Main RVDS environment variables on Windows

Environment variable	Setting
`ARMROOT`	Your installation directory root (*`install_dir`*). The default is `C:\Program Files\ARM`.
`ARMLMD_LICENSE_FILE`	The location of your ARM RealView license file. See the *FLEXnet for ARM® Tools License Management Guide* for information on this environment variable.
`ARM_PROFILER_RTSM_PATH`	The location of the RTSMs provided with RVDS, that are used by the ARM Profiler.

Table 1-3 Main RVDS environment variables on Windows (continued)

Environment variable	Setting
ARM_RVI_HELP_*N_n*	The online help files for the DSTREAM and RealView ICE utilities, where *N.n* is the version of host software installed: *install_dir*\Documentation\DebugHW\1.0\... *install_dir*\Documentation\DSTREAM\1.0\... *install_dir*\Documentation\RVI*N.n*\...
ARM_RVI_ROOT	The installation directory root for RealView ICE *install_dir*\RVI
ARM_RVI_TOOLS	The location of the executable files for the DSTREAM and RealView ICE utilities, where *N.n* is the version of host software installed: *install_dir*\RVI\Tools\N.n\...\win_32-pentium\rel
ARM Compiler toolchain environment variables	For a list of the environment variables used by ARM Compiler toolchain, see the *Introducing the ARM® Compiler toolchain*. **Note** The environment variables ARMCC41BIN, ARMCC41INC, and ARMCC41LIB must be set in RVDS 4.1.
RealView Debugger environment variables	For a list of the environment variables used by RealView Debugger, see the *RealView® Debugger Essentials Guide*.
RVDS_PROJECT	Identifies the project template file. You can override this with the --project command-line option of the ARM Compiler toolchain and RealView Debugger.
RVDS_PROJECT_WORKDIR	Identifies the project working directory. You can override this with the --workdir command-line option of the ARM Compiler toolchain and RealView Debugger.

See also

- *ARM® Workbench IDE User Guide*
- *FLEXnet for ARM® Tools License Management Guide*
- *RealView® ARMulator® ISS User Guide*
- *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain*
- *ARM® Compiler toolchain Developing Software for ARM Processors*
- *ARM® Compiler toolchain Building Linux Applications with ARM® Compiler toolchain and GNU Libraries*
- *ARM® Compiler toolchain Using the Assembler*
- *ARM® Compiler toolchain Assembler Reference*
- *ARM® Compiler toolchain Using the Compiler*

- *ARM® Compiler toolchain Compiler Reference*
- *ARM® Compiler toolchain Using C and C++ Libraries and Floating-Point Support*
- *ARM® Compiler toolchain C and C++ Libraries and Floating-Point Support Reference*
- *ARM® Compiler toolchain Using the Linker*
- *ARM® Compiler toolchain Linker Reference*
- *ARM® Compiler toolchain Creating Static Software Libraries with armar*
- *ARM® Compiler toolchain Using the fromelf Image Converter*
- *RealView® Debugger Essentials Guide*
- *RealView® Debugger Target Configuration Guide*
- *DSTREAM and RealView® ICE Using the Debug Hardware Configuration Utilities*.

Chapter 2
Getting Started with RealView Development Suite

This chapter introduces you to the basic tasks for building and debugging with the ARM® *RealView® Development Suite* (RVDS) tools. It contains the following sections:

- *Building and debugging task overview* on page 2-2
- *Using the example projects* on page 2-6
- *Getting started with ARM Profiler (RVDS Professional edition only)* on page 2-7.

2.1 Building and debugging task overview

Table 2-1 on page 2-3 is a high-level procedure showing the main tasks for building and debugging applications with the RVDS tools, and where to find the information.

The tasks referred to in the referenced documentation are not necessarily described in the order presented in Table 2-1 on page 2-3. If you are using the RealView tools for the first time, it is suggested that you work through the tasks in the order described in the referenced documents. The sequence presented in Table 2-1 on page 2-3 reflects the order in which the tasks might usually be performed.

Table 2-1 Main building and debugging tasks

Step	Description	Reference
1	The steps to follow depend on whether or not the image you want to debug already exists: • To debug an existing image, such as `dhrystone.axf` from the RVDS examples, continue at step 9. • To debug an image that has not yet been built, and you want to use the RVDS tools to build the image, continue at step 2. Alternatively, use the build tools of your choice to build your image, then continue at step 9 to debug that image.	*Using the example projects* on page 2-6
2	Choose the RVDS application you want to use to manage and build your projects: • To use the ARM Workbench IDE, continue at step 4. • To build from your system command-line using the ARM Compiler toolchain, continue at step 3.	

Table 2-1 Main building and debugging tasks (continued)

Step	Description	Reference
3	If you want to use the ARM Compiler toolchain directly, then create makefiles or command files for your platform that contain the required build commands. Continue at step 9 to load and debug your image in RealView Debugger.	*ARM® Compiler toolchain Introducing the ARM® Compiler toolchain* *ARM® Compiler toolchain Developing Software for ARM® Processors* *ARM® Compiler toolchain Building Linux Applications with ARM® Compiler toolchain and GNU Libraries* *ARM® Compiler toolchain Using the Compiler* *ARM® Compiler toolchain Compiler Reference* *ARM® Compiler toolchain Using ARM® C and C++ Libraries and Floating-Point Support* *ARM® Compiler toolchain ARM® C and C++ Libraries and Floating-Point Support Reference* *ARM® Compiler toolchain Using the Assembler* *ARM® Compiler toolchain Assembler Reference* *ARM® Compiler toolchain Using the Linker* *ARM® Compiler toolchain Linker Reference* *ARM® Compiler toolchain Creating Static Software Libraries with armar* *ARM® Compiler toolchain Using the fromelf Image Converter* *ARM® Compiler toolchain Errors and Warnings Reference*
4	Start the ARM Workbench IDE.	*ARM® Workbench IDE User Guide*
5	If an ARM Workbench IDE project already exists, continue at step 7. Otherwise, create an ARM Workbench IDE project for your application.	*ARM® Workbench IDE User Guide*
6	Set up the build configuration settings as required to build the image for your application. Continue at step 8.	*ARM® Workbench IDE User Guide*
7	Open the existing ARM Workbench IDE project.	*ARM® Workbench IDE User Guide*
8	Build the image for the ARM Workbench IDE project.	*ARM® Workbench IDE User Guide*
9	Start RealView Debugger.	*RealView® Debugger Essentials Guide*

Table 2-1 Main building and debugging tasks (continued)

Step	Description	Reference
10	Configure your debug target and connections as required.	*RealView® Debugger User Guide* *RealView® Debugger Target Configuration Guide* *DSTREAM and RealView® ICE Using the Debug Hardware Configuration Utilities*
11	Connect to your debug target.	*RealView® Debugger Essentials Guide* *RealView® Debugger User Guide*
12	Load the image ready for debugging.	*RealView® Debugger Essentials Guide* *RealView® Debugger User Guide*
13	Prepare any debugging facilities, such as breakpoints and tracepoints.	*RealView® Debugger Essentials Guide* *RealView® Debugger User Guide* *RealView® Debugger Trace User Guide* *RealView® Debugger RTOS Guide*
14	Run the image.	*RealView® Debugger Essentials Guide* *RealView® Debugger User Guide*
15	Perform the required debugging and monitoring tasks, such as stepping, and displaying contents of variables and memory. If using tracepoints, use the trace analysis facilities of RealView Debugger to analyze the trace output.	*RealView® Debugger Essentials Guide* *RealView® Debugger User Guide* *RealView® Debugger Trace User Guide* *RealView® Debugger RTOS Guide*
16	What is the result of the debugging session? • If there are problems, continue at step 17. • If there are no problems, rebuild your image for final release.	*ARM® Workbench IDE User Guide* *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain*
17	Decide how to fix any problems in your source code: • use the ARM Workbench IDE • use another source editor of your choice.	*ARM® Workbench IDE User Guide*
18	When you have fixed the problem, then you must rebuild, reload, and debug the image: • if you are using the ARM Workbench IDE, then return to step 8 • if you are using the ARM Compiler toolchain directly, then return to step 3.	

2.2 Using the example projects

The tasks described in the RVDS documentation use some of the example projects provided with RVDS.

Follow the instructions described in *Building and debugging task overview* on page 2-2 as a guide for building and debugging your applications until you are familiar with the steps involved. However, many tasks described in the user documentation require that you modify the files in the examples. Before you do this, make a backup copy of the example project files and directories.

See also, *RealView Development Suite examples* on page 1-18 for more information about the example projects provided with RVDS.

2.3 Getting started with ARM Profiler (RVDS Professional edition only)

ARM Profiler enables you to see how your code performs on a target system by:

- observing your code on target hardware using RealView ICE in conjunction with RealView Trace 2
- testing code against an ARM *Real-Time System Model* (RTSM).

Note

Profiling with DSTREAM is not supported.

When execution of your application stops, ARM Profiler produces an analysis file containing detailed information on the executed code, such as call sequences for various functions, timing characteristics, cycle counts, and instruction counts.

Table 2-2 is a high-level procedure showing the main tasks. The sequence presented in the table reflects the order that the tasks might usually be performed.

Table 2-2 Main profiling tasks

Step	Description	Reference
1	Build the image you want to analyze.	*Building and debugging task overview* on page 2-2
2	Start the ARM Workbench IDE.	*ARM® Workbench IDE User Guide*
3	If an ARM Workbench project already exists, continue at step 5.	
4	Otherwise, create an ARM Workbench project for your application and add your image file.	*ARM® Workbench IDE User Guide*
5	Select the image you want to analyze.	*ARM® Workbench IDE User Guide*
6	Decide what collection method you want to use: • If you want to use hardware or create your own run configuration, continue at step 7. • If you want to use a preconfigured RTSM, continue at step 8.	

Table 2-2 Main profiling tasks (continued)

Step	Description	Reference
7	Configure your target connections within ARM Workbench.	*ARM® Profiler User Guide* *DSTREAM and RealView® ICE Using the Debug Hardware Configuration Utilities* *RealView® ICE and RealView Trace Setting Up the Hardware* *ARM® Workbench IDE User Guide*
8	Run the image.	*ARM® Profiler User Guide* *ARM® Workbench IDE User Guide*
9	Perform the required profiling tasks, such as analyzing the summary report, code view, charts, and graphs. If there is no need for optimization, rebuild your image for final release.	*ARM® Profiler User Guide* *ARM® Workbench IDE User Guide* *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain*
10	Optimize your source code: • use the ARM Workbench IDE • use another source editor of your choice.	*ARM® Workbench IDE User Guide*
11	Return to step 1.	

Chapter 3
Changes to RealView Development Suite

This chapter describes the major changes between ARM® *RealView® Development Suite* (RVDS) v4.1 and the previous release, RVDS v4.0 SP3. It contains the following sections:

3.1 Changes to FLEXnet licensing in RVDS 4.1

RVDS 4.1 now supports a date as part of the version number in a license. This date is used to limit the period that a licensed feature is valid. The format is of a version number with a date is, *a.byyyymm*, where:

- *a.b* is the version number
- *yyyy* is a year
- *mm* is a month in numerical format.

This is typically used for *Service and Maintenance* (S&M) agreements.

See also:

- *FLEXnet for ARM® Tools License Management Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/index.html`.

3.2 Debug target support in RVDS v4.1

The following processors, models, and boards are supported:

Processor support

The following additional processors are supported:

- Cortex-A5
- Marvell Sheeva 88SV581x-v7 PJ4.

Model support

The following addtional *Real-Time System Models* (RTSMs) are supported:

- Cortex-A5
- Cortex-A9 Dual Core.

Board support

Board-Chip Definition (BCD) files and corresponding Flash methods, where appropriate, are supported for the following boards:

- Atmel AT91SAM9261-EK
- Atmel AT91SAM9263-EK
- Atmel AT91SAM9G45-EKES
- Atmel AT91SAM9RL-EK
- Icytecture iMX35 Starter board
- i.MX31
- i.MX31 LiteKit
- Freescale iMX25 PDK
- Freescale iMX27 LiteKit
- Texas Instruments Zoom OMAP34x-II Mobile Development Platform
- PBX-A9
- PHYTEC phyCORE-iMX35
- Samsung SMDK C100
- TMS320DM355
- `zoran4100.bcd`.

See also:

- Chapter 3 *Changes to RealView Development Suite*.

3.3 ARM Compiler toolchain support in RVDS v4.1

For changes to the ARM Compiler toolchain, see *Introducing the ARM® Compiler toolchain*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0529a/index.html`.

See also:

- Chapter 3 *Changes to RealView Development Suite*.

3.4 RealView Debugger support in RVDS v4.1

For changes to RealView Debugger, see *RealView® Debugger Essentials Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0181l/index.html`.

See also:

- Chapter 3 *Changes to RealView Development Suite*.

3.5 Documentation in RVDS v4.1

The following documentation changes have been made for RVDS 4.1:

- The ARM Compiler toolchain v4.1 documentation supersedes the RealView Compilation Tools v4.0 and earlier documentation. Details of the ARM Compiler toolchain v4.1 enhancements and features are included.
- The RealView Debugger documentation has been updated to describe the new features provided in v4.1.
- *FLEXnet for ARM® Tools License Management Guide* has been updated to v4.3 of the document.
- The following documentation is provided for RealView ICE v4.0 support:
  - *DSTREAM Setting Up the Hardware*, that describes the DSTREAM unit and how to connect it to your develpoment platform and workstation
  - *DSTREAM System and Interface Design Reference*, that describes the attributes of the system and interface connectors to enable you to design your hardware to communicate with a DSTREAM unit.
  - *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*, that describes how to use the RVI Config IP, RVConfig, and RVI Update utilities to setup your DSTREAM or RealView ICE hardware, and to configure connections to targets on your develpoment platform.
  - *RealView® ICE and RealView Trace Setting Up the Hardware*, that describes the RealView ICE, RealView Trace, and RealView Trace 2 hardware, and how to connect it to your develpoment platform and workstation.
  - *RealView® ICE and RealView Trace System and Interface Design Reference*, that describes the attributes of the system and interface connectors to enable you to design your hardware to communicate with RealView ICE, RealView Trace, and RealView Trace 2 hardware.

  These documents supersede the *RealView® ICE and RealView Trace User Guide* provided in previous releases.

See also:

- *RealView Development Suite documentation* on page 1-15
- *FLEXnet for ARM® Tools License Management Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/index.html`
- *Introducing the ARM® Compiler toolchain*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0529a/index.html`

- *RealView® Debugger Essentials Guide*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0181l/index.html
- *DSTREAM Setting Up the Hardware*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0481a/index.html
- *DSTREAM System and Interface Design Reference*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0499a/index.html
- *DSTREAM and RealView® ICE Using the Debug Hardware Configuration Utilities*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0498a/index.html
- *RealView® ICE and RealView Trace Setting Up the Hardware*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0515a/index.html
- *RealView® ICE and RealView Trace System and Interface Design Reference*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0517a/index.html

3.6 Deprecated features in RVDS v4.1

The following features are deprecated, and are to be removed in a future release of RVDS:

- *RealView Instruction Set Simulator* (RVISS) support
- *Instruction Set System Model* (ISSM) support
- ARM/Thumb synonyms in ARM Compiler toolchain
- The `RVCTnnBIN`, `RVCTnnINC`, and `RVCTnnLIB` environment variables are deprecated, and replaced by `ARMCCnnBIN`, `ARMCCnnINC`, and `ARMCCnnLIB`.

See also:

- Chapter 3 *Changes to RealView Development Suite*.

Appendix A
Using the armenv Tool

This appendix describes the `armenv` tool that you can use to manage your ARM® RealView® product installations.

It contains the following sections:

A.1 About the armenv tool

The armenv tool enables you to:

- set up the environment variables for ARM RealView products
- remove environment variables for ARM RealView products
- check for clashes between the ARM RealView products you have installed on a particular host machine
- set up different versions of the same product.

Note

You cannot use the armenv tool for Custom installations in this release of *RealView Development Suite* (RVDS).

You can find the armenv tool at:

install_dir/bin/*platform*

A.2 Using the armenv tool

This section describes the syntax of the `armenv` command, and shows some examples of how it can be used.

See also:

- *armenv command syntax*.

A.2.1 armenv command syntax

The command syntax of the `armenv` tool is:

```
armenv [-r root] [-u] -p product [[--and] -p product]...
[--user|--system|--proc] [--bat|--sh|--csh|--posh|--exec program [args]]
```

Table A-1 shows the command-line arguments that are available on all platforms.

Table A-1 Generic armenv arguments

Argument	Description
`--help`	Displays help on the command-line arguments.
`-r root`	The absolute path to the root of the product installation, *`install_dir`*. For example, on Windows the default root is: `C:\Program Files\ARM`
`-p product`	The ARM RealView product. See *Product syntax* on page A-4 for more information.
`--and`	Compute settings for all products before this argument, then do the same for those following it. The settings in the second group override those in the first.
`--proc`	Change the environment for the current process only. You cannot use this argument with `--system` or `--user` on Windows.
`--exec`	Execute a program in the new environment. You cannot use this argument with `--bat` on Windows, or with `--sh`, `--csh`, or `--posh` on Red Hat Linux.
`-u`	Attempts to undo the changes to the environment that were made when setting up the product.

Table A-2 shows the command-line arguments that are specific to Windows systems.

Table A-2 armenv arguments specific to Windows

Argument	Description
`--system`	Update the Windows SYSTEM area of the registry. This is the default.
`--user`	Update the Windows USER area of the registry.
`--bat`	Changes the environment for the current command prompt window. This is the default.

Table A-3 shows the command-line arguments that are specific to Red Hat Linux systems. You can specify only one of these.

Table A-3 armenv arguments specific to Red Hat Linux

Argument	Description
`--csh`	Generate a csh syntax shell script.
`--sh`	Generate a sh syntax shell script.
`--posh`	Generate a portable shell script. This is the default.

Product syntax

The syntax for specifying the product is:

`armenv -p` *`category`* `[`*`name`*`] [`*`version`* `[`*`revision`*`]] [-v` *`name value`*`]...`

where:

category is the product identifier, for example, `RVDS`.

name `armenv` uses the default name `Contents`. Do not use any other name for this option.

version is the version number of the product, for example, `3.1`. If you do not specify a version, the most recent version of the installed product is used.

revision is a specific build number for the product. If you do not specify a build number, the most recent build of the installed product is used.

-v name value

identifies a variant of the same product:

name The type of the variant, for example, `platform`. It is suggested that you use only the `platform` variant.

value The specific variant, for example, `linux-pentium`.

For example, you might have the Red Hat Linux variant of RVDS v3.1 installed.

Example A-1 How to use armenv

- To set up the Red Hat Linux environment variables for the csh shell, and for the most recent build of RVDS v4.1, enter:

  ```
  armenv -r ~/ -p RVDS 4.1 -v platform linux-pentium --csh
  ```

- To check for clashes between RVDS v4.1 and RVDS v3.1, enter:

  ```
  armenv -p RVDS 4.1 -p RVDS 3.1
  ```

- To override the RVDS v3.1 settings with the RVDS v4.1 settings, enter:

  ```
  armenv -p RVDS 3.1 --and -p RVDS 4.1
  ```

Appendix B
About Previous Releases

This chapter summarizes the major differences between previous releases of the ARM® *RealView® Development Suite* (RVDS). It contains the following sections:

For changes between RVDS v4.1 and RVDS v4.0 SP3, see Chapter 3 *Changes to RealView Development Suite*.

B.1 Changes between RVDS v4.0 SP3 and RVDS v4.0 SP2

This section describes the major changes between ARM® *RealView® Development Suite* (RVDS) v4.0 SP3 and the previous release, RVDS v4.0 SP2. It contains the following sections:

- *Processor support in RVDS v4.0 SP3*
- *Simulator support in RVDS v4.0 SP3*
- *RealView Compilation Tools support in RVDS v4.0 SP3*
- *RealView Debugger support in RVDS v4.0 SP3*
- *ARM Profiler support in RVDS v4.0 SP3* on page B-4
- *Documentation in RVDS v4.0 SP3* on page B-4
- *Deprecated features in RVDS v4.0 SP3* on page B-5
- *Obsolete features in RVDS v4.0 SP3* on page B-5.

B.1.1 Processor support in RVDS v4.0 SP3

Changes to processor support include support for Cortex™-A5.

See also:

- *Changes between RVDS v4.0 SP3 and RVDS v4.0 SP2*.

B.1.2 Simulator support in RVDS v4.0 SP3

Changes to the simulator support include support for the Cortex-A9 Dual Core RTSM.

See also:

- *Changes between RVDS v4.0 SP3 and RVDS v4.0 SP2*.

B.1.3 RealView Compilation Tools support in RVDS v4.0 SP3

Changes to *RealView Compilation Tools* (RVCT) include support for the Cortex-A5 processor.

See also:

- *Changes between RVDS v4.0 SP3 and RVDS v4.0 SP2*
- *RealView® Compilation Tools Essentials Guide*.

B.1.4 RealView Debugger support in RVDS v4.0 SP3

Changes to RealView Debugger include:

- Support for the Cortex-A5 processor.

- Support for tracing on Red Hat Linux.
- The ability to display diagnostic messages in the Diagnostic Log view.
- Additional platform support for:
  - Freescale iMX51 PDK: Extended Target Visibility and project templates
  - PHYTEC phyCORE-iMX35: Extended Target Visibility, Flash algorithms, and project templates
  - Icytecture iMX35: Extended Target Visibility, Flash algorithms, and project templates
  - Samsung SMDK C100: Extended Target Visibility, Flash algorithms, and project templates.
- Generic CoreSight device support. This enables you to examine and modify registers for a CoreSight device that you are developing, and that is not currently known to RealView Debugger.

See also:
- *Changes between RVDS v4.0 SP3 and RVDS v4.0 SP2* on page B-3
- *RealView® Debugger Essentials Guide*
- *RealView® Debugger User Guide*
- *RealView® Debugger Trace User Guide*.

B.1.5 ARM Profiler support in RVDS v4.0 SP3

The ARM Profiler plug-in and a license for its use are provided with RVDS Professional edition.

Changes to ARM Profiler include:
- support for profiling an application running on the Linux OS
- auto-calibrate in Profiler
- Cortex-M3 processor hardware support
- Cortex-A9 dual core model support.

See also:
- *Changes between RVDS v4.0 SP3 and RVDS v4.0 SP2* on page B-3
- *ARM® Profiler User Guide*.

B.1.6 Documentation in RVDS v4.0 SP3

The following documents have been updated in this release:
- *RealView® Development Suite Installation Guide*

- *RealView® Development Suite Getting Started Guide*
- *RealView® Development Suite Real-Time System Model User Guide*.

See also:

- *Changes between RVDS v4.0 SP3 and RVDS v4.0 SP2* on page B-3.

B.1.7 Deprecated features in RVDS v4.0 SP3

The following features are deprecated, and are to be removed in a future release of RVDS:

- *RealView ARMulator® ISS* (RVISS) support.
- *Instruction Set System Model* (ISSM) support.
- ARM/Thumb synonyms in RVCT.

See also:

- *Changes between RVDS v4.0 SP3 and RVDS v4.0 SP2* on page B-3.

B.1.8 Obsolete features in RVDS v4.0 SP3

In RealView Debugger, ETM Pairing is obsolete. The settings for ETM_Pairing are no longer available on the Configure ETM dialog box.

See also:

- *Changes between RVDS v4.0 SP3 and RVDS v4.0 SP2* on page B-3.

B.2 Changes between RVDS v4.0 SP2 and RVDS v4.0 SP1

This section describes the major changes between RVDS v4.0 SP2 and RVDS v4.0 SP1. It contains the following:

- *RealView Compilation Tools support in RVDS v4.0 SP2*
- *RealView Debugger support in RVDS v4.0 SP2*
- *RealView ICE, RealView Trace, and RealView Trace 2 support in RVDS v4.0 SP2*
- *Documentation in RVDS v4.0 SP2*.

B.2.1 RealView Compilation Tools support in RVDS v4.0 SP2

Changes to *RealView Compilation Tools* (RVCT) are described in the *RealView® Compilation Tools Essentials Guide*.

B.2.2 RealView Debugger support in RVDS v4.0 SP2

Changes to RealView Debugger are described in the *RealView® Debugger Essentials Guide*.

B.2.3 RealView ICE, RealView Trace, and RealView Trace 2 support in RVDS v4.0 SP2

The RealView ICE v3.4 host software is provided with all editions of RVDS v4.0 SP1.

RealView ICE includes support for the Cortex-M0 processor.

See also:

- *RealView® ICE and RealView Trace User Guide*.

B.2.4 Documentation in RVDS v4.0 SP2

Changes to the RVDS documentation suite include:

- Changes to RVCT documentation are described in the *RealView® Compilation Tools Essentials Guide*.
- Changes to the RealView Debugger documentation are described in the *RealView® Debugger Essentials Guide*.
- All documents reflect the feature changes in the component tools.

See also *Glossary* on page Glossary-1 for a list of ARM terms used in the RVDS documentation.

B.3 Changes between RVDS v4.0 SP1 and RVDS v4.0

This section describes the major changes between RVDS v4.0 SP1 and RVDS v4.0. It contains the following:

- *Processor support in RVDS v4.0 SP1*
- *RealView Compilation Tools support in RVDS v4.0 SP1*
- *RealView Debugger support in RVDS v4.0 SP1*
- *RealView ICE, RealView Trace, and RealView Trace 2 support in RVDS v4.0 SP1*
- *Documentation in RVDS v4.0 SP1*.

B.3.1 Processor support in RVDS v4.0 SP1

Changes to the processor support include Cortex™-M0 processor in RealView ICE.

B.3.2 RealView Compilation Tools support in RVDS v4.0 SP1

Changes to *RealView Compilation Tools* (RVCT) are described in the *RealView® Compilation Tools Essentials Guide*.

B.3.3 RealView Debugger support in RVDS v4.0 SP1

Changes to RealView Debugger are described in the *RealView® Debugger Essentials Guide*.

B.3.4 RealView ICE, RealView Trace, and RealView Trace 2 support in RVDS v4.0 SP1

The RealView ICE v3.4 host software is provided with all editions of RVDS v4.0 SP1.

RealView ICE includes support for the Cortex-M0 processor.

See also:

- *RealView® ICE and RealView Trace User Guide*.

B.3.5 Documentation in RVDS v4.0 SP1

Changes to the RVDS documentation suite include:

- Changes to RVCT documentation are described in the *RealView® Compilation Tools Essentials Guide*.
- Changes to the RealView Debugger documentation are described in the *RealView® Debugger Essentials Guide*.

- All documents reflect the feature changes in the component tools.

See also *Glossary* on page Glossary-1 for a list of ARM terms used in the RVDS documentation.

B.4 Changes between RVDS v4.0 and RVDS v3.1 Professional edition

This section describes the major changes between RVDS v4.0 and RVDS v3.1 Professional edition. It contains the following:

- *Host platform support in RVDS v4.0*
- *Processor support in RVDS v4.0*
- *Simulator support in RVDS v4.0* on page B-10
- *RealView Compilation Tools support in RVDS v4.0* on page B-10
- *RealView Debugger support in RVDS v4.0* on page B-10
- *RealView ICE, RealView Trace, and RealView Trace 2 support in RVDS v4.0* on page B-10
- *ARM Profiler support in RVDS v4.0* on page B-11
- *IDE support in RVDS v4.0* on page B-11
- *Documentation in RVDS v4.0* on page B-12
- *Miscellaneous changes in RVDS v4.0* on page B-12
- *Deprecated features in RVDS v4.0* on page B-13
- *Obsolete features in RVDS v4.0* on page B-13.

B.4.1 Host platform support in RVDS v4.0

Added support for:

- Windows Vista Business Service Pack 1
- Windows Vista Enterprise Service Pack 1
- Windows Server 2003 (Compiler only)
- Red Hat Enterprise Linux WS version 5 for x86 using GNOME Window Manager and Bash Shell.

Continued support for:

- Windows XP Professional Service Pack 2
- Red Hat Enterprise Linux WS version 4 for x86 using GNOME Window Manager and Bash Shell.

Removed support for Windows 2000.

B.4.2 Processor support in RVDS v4.0

The following additional processors are supported:

- Cortex-A9 processor

- Faraday FA526, FA626, and FA626TE processors.
- Marvell Feroceon 88FR101 and 88FR111 processors.

Note

A license is provided with RVDS Professional edition to support the Cortex-A9 processor.

B.4.3 Simulator support in RVDS v4.0

Changes to the simulator support include the following *Real-Time System Models* (RTSMs):

- Versatile *Emulation Baseboard* (EB) with ARM926EJ-S™
- Versatile EB with ARM1136JF-S™
- Versatile EB with ARM1176JZF-S™
- Versatile EB with Cortex-A8
- Versatile EB with Cortex-A9
- Versatile EB with Cortex-R4F.

These replace the Integrator/CP based models supported in RVDS v3.1.

Note

A license is provided with RVDS Professional edition to support the Cortex-A9 simulated processor.

B.4.4 RealView Compilation Tools support in RVDS v4.0

In RVCT, `-O3` no longer implies `--fpmode=fast`.

Other changes to RVCT are described in the *RealView® Compilation Tools Essentials Guide*.

B.4.5 RealView Debugger support in RVDS v4.0

Changes to RealView Debugger are described in the *RealView® Debugger Essentials Guide*.

B.4.6 RealView ICE, RealView Trace, and RealView Trace 2 support in RVDS v4.0

The RealView ICE host software is provided with all editions of RVDS v4.0.

See also:

- *RealView® ICE and RealView Trace User Guide*.

B.4.7 ARM Profiler support in RVDS v4.0

The ARM Profiler plug-in and a license for its use are provided with RVDS Professional edition.

Changes to ARM Profiler include:

- Support for profiling an application running on the Symbian OS.
- Data capture support for the following processors:
  - ARM7TDMI®
  - ARM946E-S™
  - ARM966E-S™
  - ARM11 MPCore™
  - Cortex-A8
  - Cortex-M3
  - Cortex-R4(F).
- Data capture support for the following RTSMs:
  - Cortex-A9
  - Cortex-R4F.
- Cycle-accurate profiling for processors running at 10MHz.
- Support for filtering of data based on the Call Chain report.
- Live update, to view the capture of profiling data in real-time.
- Stack depth tracking, to provide a summary of the stack depth usage for a call chain during execution.

See also:

- *ARM® Profiler User Guide*.

B.4.8 IDE support in RVDS v4.0

ARM Workbench IDE is available for all supported platforms, and is installed as part of the RVDS installation. The following changes have been made to the ARM Workbench IDE:

- the Eclipse IDE and associated plug-ins are updated to the latest versions

- the following editors are provided:
  - a properties editor for use with ARM assembler and C/C++ source files
  - a scatter file editor to create and edit scatter-loading description files
  - an ELF content viewer.
- you can export IP-XACT design files to create memory map and peripheral definitions in the RealView Debugger *Board-Chip Definition* (BCD) file format
- updates to the ARM Workbench IDE and plug-ins on are available on the ARM website at `http://www.arm.com/eclipse`.

See also:

- *ARM® Workbench IDE User Guide*
- *RealView® Debugger Target Configuration Guide*.

B.4.9 Documentation in RVDS v4.0

Changes to the RVDS documentation suite include:

- The provision of a new standalone document browser. This enables you to view the HTML-formatted documentation without having to run the ARM Workbench IDE.
- The *RealView® Development Suite Glossary* is no longer a separate document. It is included in this document, the *RealView® Development Suite Getting Started Guide*.
- Changes to RVCT documentation are described in the *RealView® Compilation Tools Essentials Guide*.
- Changes to the RealView Debugger documentation are described in the *RealView® Debugger Essentials Guide*.
- All documents reflect the feature changes in the component tools.

See also *Glossary* on page Glossary-1 for a list of ARM terms used in the RVDS documentation.

B.4.10 Miscellaneous changes in RVDS v4.0

The following additional changes have been made in this release:

- An ARM Support Wizard is available. See *ARM Support Wizard* on page 1-11 for more information.

- New example projects are provided:
  - Cortex-A9 version of the cached Dhrystone example code
  - `linux_apps`
  - `mandelbrot`.

  See *RealView Development Suite examples* on page 1-18 for more information.
- Revised examples are:
  - Cached Dhrystone example code is revised to run on the latest software models and current hardware.
  - The MMUgen utility supports the generation of MMU page table entries for ARMv6 and ARMv7 cached processors.
  - The DCC examples have been updated for use with the latest processors.
  - *Application Note 133 Using VFP with RVDS* is revised for the latest processors.

B.4.11 Deprecated features in RVDS v4.0

The following features are deprecated, and are to be removed in a future release of RVDS:

- *RealView ARMulator® ISS* (RVISS) support
- *Instruction Set System Model* (ISSM) support
- SoC Designer support
- support for RealView ICE v1.2 and earlier
- ARM/Thumb synonyms in RVCT.

  for other deprecated features of RVCT, see the *RealView® Compilation Tools Essentials Guide*.
- for other deprecated features of RealView Debugger, see the *RealView® Debugger Essentials Guide* for more information.

B.4.12 Obsolete features in RVDS v4.0

The following features are obsolete:

- Windows 2000 support.
- CodeWarrior IDE support.
- ARM Ltd. Direct Connection to Versatile boards.

- The **Uninstall Wizard** option has been removed from:
  - **Start → All Programs → ARM** menu on Windows
  - **Start Menu → Programs → ARM** menu on Red Hat Linux.

  The product-specific **Modify or Uninstall RVDS *N.n*** option is still available.
- Some features of RVCT. See the *RealView® Compilation Tools Essentials Guide* for more information.
- Some features of RealView Debugger. See the *RealView® Debugger Essentials Guide* for more information.

B.5 Changes between RVDS v3.1 Professional edition and RVDS v3.1

The following major changes between RVDS v3.1 Professional edition and RVDS v3.1 are:

- RealView Profiler support including examples, documentation, and the following Real-Time System Models:
  - ARM926 *Emulation Board* (EB)
  - ARM1136 EB
  - ARM1176 EB
  - Cortex-A8 EB.
- RealView ICE v3.2 host software is provided.
- Licenses for the NEON™ Vectorizing Compiler and RealView Profiler are provided.
- New Eclipse plug-ins are provided:
  - ARM Flash Programmer
  - ARM Assembler Editor
  - CodeWarrior Importer.

B.6 Changes between RVDS v3.1 and RVDS v3.0 SP1

This section describes the major changes between RVDS v3.1 and RVDS v3.0 SP1. It contains the following:

- *Processor support in RVDS 3.1*
- *Simulator support in RVDS 3.1*
- *Project template support in RVDS 3.1* on page B-17
- *RealView Compilation Tools support in RVDS 3.1* on page B-17
- *RealView Debugger support in RVDS 3.1* on page B-18
- *IDE support in RVDS 3.1* on page B-18
- *Documentation changes in RVDS 3.1* on page B-18
- *Deprecated features in RVDS 3.1* on page B-18
- *Obsolete features in RVDS 3.1* on page B-18.

B.6.1 Processor support in RVDS 3.1

Processor support includes:

- ARM Cortex-M1
- ARM Cortex-M3 revision 1
- ARM Cortex-R4
- StarCore SC1200 DSP (debug support only).

B.6.2 Simulator support in RVDS 3.1

RVDS provides the following simulator support:

- *Instruction Set System Model* (ISSM) simulates the following additional processors:
  - Cortex-M1
  - Cortex-M3 revision 1, which supports cycle counting
  - Cortex-R4.
- SoC Designer, to allow connections to SoC Designer targets. You must purchase SoC Designer separately.
- *Real-Time System Model* (RTSM), to allow connections to RTSM targets.

Support for these is installed with RealView Debugger.

B.6.3 Project template support in RVDS 3.1

The Eclipse New Project Wizard enables you to create new projects for the RVDS component tools depending on the requirements of your application. These projects can be based on project templates supplied with RVDS.

Additional RealView Debugger and *RealView Compilation Tools* (RVCT) command line options are provided to support the use of RVDS project templates:

- `--no_project`
- `--project` *filename*
- `--reinitialize_workdir`
- `--workdir` *pathname*.

Also, preconfigured project templates are provided in the directory:

install_dir`\project_templates`

These project templates are grouped in the following subdirectories:

ARM RealView Development Boards

These project templates include RealView Debugger configurations that enable you to connect to targets through RealView ICE and associated connection interfaces.

Bare ARM Cores

These project templates include RealView Debugger configurations that enable you to connect to targets through ISSM and RealView ARMulator ISS interfaces as appropriate.

You can also set the project template and working directory values using the following environment variables:

- `RVDS_PROJECT`
- `RVDS_PROJECT_WORKDIR`.

See also

- *RealView® Compilation Tools Compiler Reference Guide*
- *RealView® Debugger User Guide*.

B.6.4 RealView Compilation Tools support in RVDS 3.1

Changes to RVCT are described in the *RealView® Compilation Tools Essentials Guide*.

B.6.5 RealView Debugger support in RVDS 3.1

Changes to RealView Debugger are described in the *RealView® Debugger Essentials Guide*.

B.6.6 IDE support in RVDS 3.1

Eclipse and the Eclipse Plug-in for RVDS are installed on all supported platforms as part of the RVDS installation. For more information on using the Eclipse Plug-in for RVDS, see the *RealView® Development Suite Eclipse Plug-in User Guide*.

B.6.7 Documentation changes in RVDS 3.1

The main changes to the RVDS documentation are as follows:

- All RVDS documentation is available in HTML format. The Eclipse viewer enables you to search across all the documentation. Although you can view the documentation in a separate web browser, you cannot search across all the documentation.
- Changes to the RealView Debugger documentation are described in the *RealView® Debugger Essentials Guide*.
- Changes to the RVCT documentation are described in the *RealView® Compilation Tools Essentials Guide*.

B.6.8 Deprecated features in RVDS 3.1

The following features are deprecated in RVDS v3.1:

- Windows 2000 support.
- CodeWarrior IDE support.
- Some features of RVCT. See the *RealView® Compilation Tools Essentials Guide* for more information.
- Some features of RealView Debugger. See the *RealView® Debugger Essentials Guide* for more information.

B.6.9 Obsolete features in RVDS 3.1

The following features are obsolete in RVDS v3.1:

- Support for *ARM eXtended Debugger* (AXD) and *ARM Symbolic Debugger* (armsd).

- The ARM Developer Suite™ v1.2.1 CD-ROM is no longer provided.
- Support for the Solaris platform.
- Support for the Red Hat Enterprise Linux v3 platform.
- Dynatext documentation is no longer provided.
- RVCT has obsolete features.
- RealView Debugger has obsolete features.

See also

- *RealView® Compilation Tools Essentials Guide*
- *RealView® Debugger Essentials Guide*.

B.7 Changes between RVDS v3.0 SP1 and RVDS v3.0

RVDS v3.0 Service Pack 1 also provides a consolidation of enhancements made in the RVCT and RealView Debugger since the original RVDS v3.0 release, including:

- preliminary support for Cortex-R4, including compiler support, debugger support, and a new *Instruction Set System Model* (ISSM) model
- improvements to compilation times and DWARF3 debug data sizes over RVDS v3.0
- SIMD NEON assembler extended to include Programmer's notation
- improved user interface for Debug of a multiprocessor MPCore™ target
- additional Cortex-M3 Examples
- support for Marvell Feroceon 88FRxxx processors.

The RealView Debugger Synchronization Control window has been re-engineered, which also includes the synchronization of various actions. A corresponding `SYNCHACTION` CLI command is provided.

For more information, see the *RealView® Development Suite v3.0 SP1 Release Notes*.

B.8 Changes between RVDS v3.0 and RealView Developer Suite v2.2 SP1

This section describes the changes between RVDS v3.0 and RealView Developer Suite v2.2 SP1. It contains the following:

- *New features in RVDS v3.0*
- *Debugger support in RVDS v3.0*
- *Compilation Tools support in RVDS v3.0* on page B-22
- *Simulator support in RVDS v3.0* on page B-23
- *CodeWarrior for RVDS changes in RVDS v3.0* on page B-23
- *Documentation changes in RVDS v3.0* on page B-23
- *Deprecated and removed features in RVDS v3.0* on page B-24.

B.8.1 New features in RVDS v3.0

The following new features are available in RVDS v3.0:

- Support for TrustZone® technology.
- Support for *Thumb®-2 Execution Environment* (Thumb-2EE).
- Support for the ARM Cortex processor family:
  - Cortex-A8
  - Cortex-M3.
- Simulator models for the Cortex-A8 and Cortex-M3 processors are available. These models are accessible through the new ISSM Target Access in RealView Debugger.

B.8.2 Debugger support in RVDS v3.0

The major changes to RealView Debugger v3.0 are as follows:

- RealView Debugger runs as a single process. The Target Vehicle Server (TVS) no longer exists as a separate entity.
- The Connection Control window has been re-engineered. See the *RealView® Debugger User Guide* for more information.
- The features on the **Synch** tab are available in a separate Synchronization Control window. See the *RealView® Debugger User Guide* for more information.
- The Register pane has been re-engineered. You can create a user-specific view by copying selected registers to a User tab. See the *RealView® Debugger User Guide* for more information.

- The RealView Debugger project manager and related functionality has been removed, so you can no longer create projects and build images within RealView Debugger. However, the source code editing and searching features are still available.

  Note

  To create and build your projects in RVDS v3.0, use CodeWarrior for RVDS (see *CodeWarrior for RVDS changes in RVDS v3.0* on page B-23).

- Simulator support has changed. See *Simulator support in RVDS v3.0* on page B-23 for more information.
- RealView Broker (RVBroker) has been re-engineered. Although RealView Debugger still runs RVBroker automatically for local host, RealView ARMulator® ISS connections, starting RealView Broker for remote simulator connections has changed. You must specify a username when starting RealView Broker on a remote workstation. See the *RealView® Debugger Target Configuration Guide* for more information.

  Note

  Support for Multi-ICE® direct connect has been removed in RVDS v3.0.

For more information about the changes to RealView Debugger, see the *RealView® Debugger Essentials Guide*.

B.8.3 Compilation Tools support in RVDS v3.0

The major changes to RVCT v3.0 are as follows:

- RVCT v3.0 supports Thumb-2EE.
- The ARM assembler can be used to assemble Intel Wireless MMX Technology instructions to develop code for the PXA270 processor.
- RVCT v3.0 provides full support for DWARF 3 (Draft Standard 9.6) debug tables, as described in the *ABI for the ARM Architecture (base standard)* [BSABI].
- The ARM compiler and linker support *Thread Local Storage* (TLS) to enable programs to use multiple threads.
- The ARM compiler supports improved loop optimization.

For more information about the changes to RVCT, see the *RealView® Compilation Tools Essentials Guide*.

B.8.4 Simulator support in RVDS v3.0

RVDS provides the following simulator support:

- ISSM, which simulates Cortex-A8 and Cortex-M3 processors.
- An MPCore simulated target is available in RealView ARMulator ISS. However, this does not model multiple processors, so connecting to this model connects only to a single processor.

The RDI ARMulator simulated target is no longer available. Use either:

- the `new_arm` connection on the `localhost` Target Access to connect to simulated ARM processors using RealView ARMulator ISS
- the `ISSM` Target Access to connect to one of the Cortex models.

These are both installed with RealView Debugger.

B.8.5 CodeWarrior for RVDS changes in RVDS v3.0

The major changes to CodeWarrior for RVDS are as follows:

- The External Build Wizard is supported. This is intended to replace the deprecated makefile importer and the Batch File Runner functionality.
- Support for the `.cc` file extension has been added.
- CodeWarrior warns you when an unrecognized source file extension (such as `.cmd`) is used.
- Panel settings have been added or removed in line with changes to the compilation tools. See the *RealView® Compilation Tools Essentials Guide* for more information.

For more information, see the *RealView® Development Suite CodeWarrior IDE Guide*.

B.8.6 Documentation changes in RVDS v3.0

Apart from documenting the new features of RVDS, the main changes to the RVDS documentation are with the RealView Debugger documentation. The RealView Debugger documentation has been reorganized as follows:

- The information in the *RealView® Debugger Extensions User Guide* is moved to the following documents:
  - the chapter that describes DSP support is included in the *RealView® Debugger User Guide*

  - — the chapter that describes Debugging multiple targets is included in the *RealView® Debugger User Guide*
  - — the chapter that describes Tracing in RealView Debugger is in the *RealView® Debugger Trace User Guide*
  - — the chapter that describes OS support is in the *RealView® Debugger RTOS Guide*.
- The chapter that describes connecting to targets in the *RealView® Debugger Target Configuration Guide* is in the *RealView® Debugger User Guide*.
- The *RealView® Debugger User Guide* has been restructured to be more task-based.
- The *RealView® Debugger Project Management Guide* is not provided, because the RealView Debugger project manager has been removed.

For other detailed changes to the RVDS documentation suite, see:
- *RealView® Debugger Essentials Guide*
- *RealView® Compilation Tools Essentials Guide*.

B.8.7 Deprecated and removed features in RVDS v3.0

The following features are deprecated or removed in RVDS v3.0:

- Support for *ARM eXtended Debugger* (AXD) and *ARM Symbolic Debugger* (armsd) is deprecated.
- The makefile importer and Batch File Runner functionality in CodeWarrior is deprecated.
- Support for remote RealView Debugger connections through Multi-ICE direct connect has been removed. This means that connections to DSP processors is available only with RealView ICE, which you must purchase separately.
- The RealView Debugger project manager and related functionality has been removed.

For more information of other deprecated features, see:
- *RealView® Compilation Tools Essentials Guide*
- *RealView® Debugger Essentials Guide*.

B.9 Changes between RealView Developer Suite v2.2 SP1 and RealView Developer Suite v2.2

This section describes the changes between RealView Developer Suite v2.2 SP1 and RealView Developer Suite v2.2. It contains the following:

- *Documentation changes in RVDS v2.2 SP1*
- *Debugger support in RVDS v2.2 SP1*
- *Compilation Tools support in RVDS v2.2 SP1*.

B.9.1 Documentation changes in RVDS v2.2 SP1

The changes to the documentation include:

- The *RealView® Developer Suite CodeWarrior IDE Guide* is included, which describes how to use the ARM features of CodeWarrior.
- The chapter that described getting started with CodeWarrior has been removed from the *RealView® Developer Suite Getting Started Guide*, and incorporated into the *RealView® Developer Suite CodeWarrior IDE Guide*.
- Changes to the RealView Debugger documentation for the supported DSPs.

B.9.2 Debugger support in RVDS v2.2 SP1

The main difference between the debugging tools in RealView Developer Suite v2.2 SP1 and RealView Developer Suite v2.2 is with RealView Debugger, which has support for CEVA-Oak, CEVA-TeakLite, CEVA-Teak, ZSP400, and ZSP500 DSPs.

B.9.3 Compilation Tools support in RVDS v2.2 SP1

There are minor changes to the compilation tools between RealView Developer Suite v2.2 SP1 and RealView Developer Suite v2.2. See the *RealView® Compilation Tools Essentials Guide* for more information.

B.10 Changes between RealView Developer Suite v2.2 and RealView Developer Suite v2.1

This section describes the changes between RealView Developer Suite v2.2 and RealView Developer Suite v2.1. It contains the following:

- *IDE support in RVDS v2.2*
- *Debugger tool support in RVDS v2.2*
- *Compilation Tools support in RVDS v2.2* on page B-27
- *Agilent Probe support in RVDS v2.2* on page B-27.

B.10.1 IDE support in RVDS v2.2

The CodeWarrior IDE is provided to replace the RealView Debugger IDE. The CodeWarrior IDE in RealView Developer Suite v2.2 is based on Metrowerks CodeWarrior v5.6.

Note

In RealView Developer Suite v2.2, CodeWarrior for RealView Developer Suite is supported on Windows XP and Windows 2000 systems only, and is not supplied for Red Hat Linux.

B.10.2 Debugger tool support in RVDS v2.2

The main differences between the debugging tools in RealView Developer Suite v2.2 and RealView Developer Suite v2.1 are with RealView Debugger, which has:

- an improved menu structure
- an improved pane handling mechanism
- improved data navigation with the new Data Navigator pane
- internationalization support
- improved source code coloring
- trace, analysis, and profiling enhancements
- enhanced RTOS support
- support for gcc built images
- additional CLI commands, `PRINTDSM` and `TRACEEXTCOND`.

Also, support for standalone editors and the Vi editing mode has been removed from RealView Debugger.

For a detailed list of changes, see the *RealView® Debugger Essentials Guide*.

B.10.3 Compilation Tools support in RVDS v2.2

The main differences between the compilation tools in RealView Developer Suite v2.2 and RealView Developer Suite v2.1 are:

- RVCT v2.2 includes support for new ARMv6 cores, for example, the ARM1176JZF-S, incorporating ARM TrustZone technology, the ARM968EJ-S™, the ARM1156T2F-S™, and the ARM MPCore.
- Supported in RVCT v2.2, the new Thumb-2 instruction set introduces many new 32-bit instructions, and some new 16-bit instructions.

  The Thumb-2 instruction set includes older 16-bit Thumb instructions as a subset.
- RVCT v2.2 is fully compliant with the *Base Platform ABI for the ARM Architecture* [BPABI] (unpublished DRAFT).
- RVCT v2.2 provides initial support for DWARF3 (Draft Standard 9) debug tables, as described in the *ABI for the ARM Architecture (base standard)* [BSABI].
- The command-line option `-g` switches on the generation of debug tables for the current compilation. Optimization options are specified by `-O0`, `-O1`, `-O2`, or `-O3`. By default, using the `-g` option does not affect the optimization setting.

  This is a change in behavior for RVCT v2.2.
- RVCT v2.2 supports the command-line option `--apcs /fpic` to compile code that is compatible with System V shared libraries.
- The ARM linker supports building, and linking against, shared libraries. New command-line options are available to build SVr4 executable files and shared objects, and to specify how code is generated.
- The ARM linker supports the GNU-extended symbol versioning model.
- The ARM implementation of floating-point computations has been changed to provide improved support for C99 functions. Where this changes behavior significantly, a compatibility mode has been introduced to aid developers to migrate code to use the new features.
- RVCT v2.2 supports building of Linux applications and shared libraries.

For a detailed list of changes, see the *RealView® Compilation Tools Essentials Guide*.

B.10.4 Agilent Probe support in RVDS v2.2

Agilent Probe support is available as a custom installation option in RealView Developer Suite v2.2.

B.11 Changes between RealView Developer Suite v2.1 and RealView Developer Suite v2.0

This section describes the changes between RealView Developer Suite v2.1 and RealView Developer Suite v2.0. It contains the following:

- *Debugger tool support in RVDS v2.1*
- *Compilation Tools support in RVDS v2.1*.

B.11.1 Debugger tool support in RVDS v2.1

The main differences between the debugging tools in RealView Developer Suite v2.1 and RealView Developer Suite v2.0 are:

- *ARM eXtended Debugger* (AXD) is included
- *ARM Symbolic Debugger* (armsd) is included
- RealView Debugger has:
  - trace and profiling enhancements
  - enhanced RTOS support
  - new toolbar buttons and menu changes that mean you have quick access to commonly used features.

B.11.2 Compilation Tools support in RVDS v2.1

The main differences between the compilation tools in RealView Developer Suite v2.1 and RealView Developer Suite v2.0 are:

- Increased compliance with the *Application Binary Interface for the ARM Architecture (Base Standard)* (ABI for the ARM Architecture (Base Standard)). See the ABI for the ARM Architecture page at `http://www.arm.com/`.
- C++ exception handling is supported. Therefore, with the exception of export templates, the remainder of ISO C++ is supported as defined by the *ISO/IEC 14822 :1998 International Standard for C++*.
- More optimization features are included, such as multifile compilation and linker feedback.
- Compression of read/write data areas is provided, to further reduce the image size.
- Some GNU C and C++ extensions are supported.
- Many new command-line options have been added to the build tools.

- The single-dash keyword and some command-line options are deprecated.

Note

The tools check more strictly the requirement for eight-byte stack alignment. The compiler generates code with `PRESERVE8` and `REQUIRE8`. The linker checks that code that requires eight-byte alignment only calls code that preserves eight-byte alignment. Therefore, this has implications for your legacy assembler code, object files and libraries. You must check that your existing assembly files, object files, or libraries preserve eight-byte alignment and correct them if required. For more information, see the *RealView® Compilation Tools Assembler Guide* and the *RealView® Compilation Tools Linker and Utilities Guide* for more information.

B.12 Changes between RealView Developer Suite v2.2 and ADS v1.2.1

This section describes the changes between RealView Developer Suite v2.2 and ARM Developer Suite® (ADS) v1.2.1. It contains the following:

- *CodeWarrior IDE changes between RVDS v2.2 and ASD v1.2.1*
- *Debugger changes between RVDS v2.2 and ASD v1.2.1* on page B-31
- *Compilation Tools changes between RVDS v2.2 and ASD v1.2.1* on page B-32
- *ARM simulator changes between RVDS v2.2 and ASD v1.2.1* on page B-33.

B.12.1 CodeWarrior IDE changes between RVDS v2.2 and ASD v1.2.1

The changes between CodeWarrior for RealView Developer Suite and CodeWarrior for ADS are:

- CodeWarrior for ADS was based on CodeWarrior v4.2. CodeWarrior for RealView Developer Suite is based on Metrowerks CodeWarrior v5.6.
- The CodeWarrior Perl plug-in, MWPerl, which provided support for processing Perl scripts in CodeWarrior v4.2 has been removed in CodeWarrior v5.6. It is no longer supported by Metrowerks.
- The ARM tool-specific configuration panels are tailored to RealView Developer Suite v2.2.
- The separate ARM compilers are combined into a single compiler in RealView Developer Suite v2.2, therefore there is only one compiler configuration panel in RealView Developer Suite v2.2.
- You can run and debug your image with RealView Debugger, in addition to AXD and armsd.
- You can concatenate libraries.
- You can import CodeWarrior for ADS projects into CodeWarrior for RealView Developer Suite.
- The default ARM stationery in CodeWarrior for RealView Developer Suite does not include a `DebugRel` build target. However, a `DebugRel` build target is created if you import a CodeWarrior for ADS project, to preserve any settings you might have configured for that build target.
- Unlike the ADS compiler, the RVCT compiler does not generate browser information. This functionality is provided by the builtin language parser of CodeWarrior.
- Code formatting.

- Code completion, including code completion for C++ template classes.
- Go to next/previous function.
- Word wrap when printing.
- Support for source-relative `#includes`.
- Find inside/outside of comments.
- Improved language parser speed and feedback.
- New editor bindings.
- Ability to show and hide the Code and Data columns in the project window.
- Support for workspaces.

Note

All target connection and debugging features in the CodeWarrior IDE are not available in CodeWarrior for RealView Developer Suite. You must run one of the ARM debuggers to perform these functions.

B.12.2 Debugger changes between RVDS v2.2 and ASD v1.2.1

The main differences between the debugging tools in RealView Developer Suite v2.2 and ADS v1.2.1 are:

- RealView Debugger is the latest ARM debugger, which enables you to perform advanced debugging functions such as:
  - multiprocessor debugging
  - OS-aware debugging
  - extended target visibility
  - trace, analysis, and profiling
  - access to the RealView ICE JTAG control unit over Ethernet and USB.
- AXD is enhanced to be able to debug C and C++ programs built with the new RVCT provided with RealView Developer Suite v2.2.

B.12.3 Compilation Tools changes between RVDS v2.2 and ASD v1.2.1

The main differences between the build tools in RealView Developer Suite v2.2 and ADS v1.2.1 are as follows:

- Compliance with the new ABI for the ARM Architecture (Base Standard). See the ABI for the ARM Architecture page at `http://www.arm.com/`. This is different to the old ADS ABI. Some compatibility is provided with the `--apcs /adsabi` command line option.
- There is full ISO C++ support as defined by the *ISO/IEC 14822 :1998 International Standard for C++*, by way of the EDG (Edison Design Group) front-end. This includes exceptions, namespaces, templates, and intelligent implementation of *Run-Time Type Information* (RTTI), but excludes the export of templates.
- Support for some GNU language extensions.
- ARM and Thumb compilation on a per-function basis.
- Re-engineered inline assembler, and a new embedded assembler that enables you to include out-of-line assembly code.
- Linker feedback to remove unused functions.
- Full support for ARM architecture v6 instructions has been added.
- Read/write data compression enables the optimization of ROM size.
- Removal of unused C++ virtual functions.
- Multifile compilation, which performs optimizations across multiple compilation units.
- You can specify a library search path, to indicate where to search for your user libraries.
- You can separate RO code and data into different execution regions.
- There are new scatter-loading attributes.
- Unicode and multibyte characters are supported.
- Compiler intrinsics are available to access the return address of a function, the current stack pointer value, and the current program counter value. An additional intrinsic enables you to insert the `BKPT` instruction in your C or C++ code.
- You can identify a function that does not return, so that the compiler generates more efficient code.

- The C++ name mangling scheme has changed.
- The *ARM Profiler* (`armprof`) is not provided with RVCT.

  Note

  This is not the same as the ARM Profiler provided with RVDS v3.1 Professional edition, and later versions.

- The ARM Applications Library is not provided with RVCT.
- Unlike the ADS compiler, the RVCT compiler does not generate browser information.
- There are changes to the assembler, compiler, and linker command-line options.

  Support for double dashes -- to indicate command-line keywords (for example, `--cpp`) and single dashes - for command-line single-letter options, with or without arguments (for example, `-S`).

  Note

  The single-dash command-line options used in previous versions of ADS and RVCT are still supported for backwards-compatibility.

- The fromelf option `-ihf` has been removed.

Note

The tools check more strictly the requirement for eight-byte stack alignment. The compiler generates code with `PRESERVE8` and `REQUIRE8`. The linker checks that code that requires eight-byte alignment only calls code that preserves eight-byte alignment. Therefore, this has implications for your legacy assembler code, object files and libraries. You must check that your existing assembly files, object files, or libraries preserve eight-byte alignment and correct them if required. For more information, see the *RealView® Compilation Tools Assembler Guide* and the *RealView® Compilation Tools Linker and Utilities Guide* for more information.

B.12.4 ARM simulator changes between RVDS v2.2 and ASD v1.2.1

RealView ARMulator ISS is the latest version of the ARM simulator. It supports connections through RealView Connection Broker and RDI. When connecting to the simulator through RealView Connection Broker under RealView Debugger, you can have multiple connections to the simulator. You can connect to the RDI interface of RealView ARMulator ISS using RealView Debugger, AXD v1.3, and armsd.

Note

Although you can install ADS in addition to RealView Developer Suite v2.2, you must exercise caution if you use both RealView ARMulator ISS and ADS ARMulator. See the *RealView® Developer Suite v2.2 Release Notes* for more information.

Glossary

The following terminology is used in the documentation provided with ARM® *RealView® Development Suite* (RVDS):

AAPCS *See* Procedure Call Standard for the ARM Architecture.

ABI for the ARM Architecture (base standard) (BSABI)
The ABI for the ARM Architecture is a collection of specifications, some open and some specific to ARM architecture, that regulate the inter-operation of binary code in a range of ARM architecture-based execution environments. The base standard specifies those aspects of code generation that must be standardized to support inter-operation and is aimed at authors and vendors of C and C++ compilers, linkers, and runtime libraries.

Adaptive clocking A technique used by RealView ICE where it sends out a clock signal and then waits for the returned clock before generating the next clock pulse. The technique enables the RealView ICE run control unit to adapt to differing signal drive capabilities and differing cable lengths.

Advanced eXtensible Interface (AXI)
A bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure.

The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.

Advanced Microcontroller Bus Architecture (AMBA®)
On-chip communications standard for high-performance 32-bit and 16-bit embedded microcontrollers.

Advanced High-performance Bus (AHB)
A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave IP developments and ARM recommends only a subset of the protocol is usually used. This subset is defined as the AMBA AHB-Lite protocol.

See also Advanced Microcontroller Bus Architecture *and* AHB-Lite.

AHB *See* Advanced High-performance Bus.

AHB Access Port (AHB-AP)
CoreSight™ supports access to a system bus infrastructure using the *AHB Access Port* (AHB-AP) in the *Debug Access Port* (DAP). The AHB-AP provides an AHB master port for direct access to system memory. If an alternate bus protocol is implemented, you can use an AHB bridge to map transactions. For example, you can use an AHB to AXI bridge to enable access to an AXI bus matrix.

CoreSight also supports AHB bus tracing using an *AHB Trace Macrocell* (HTM).

See also Advanced eXtensible Interface, AHB Trace Macrocell, CoreSight, *and* Debug Access Port.

AHB-AP *See* AHB Access Port.

AHB-Lite A subset of the full AMBA AHB protocol specification. It provides all of the basic functions required by the majority of AMBA AHB slave and master designs, particularly when used with a multi-layer AMBA interconnect. In most cases, the extra facilities provided by a full AMBA AHB interface are implemented more efficiently by using an AMBA AXI protocol interface.

AHB Trace Macrocell (HTM)
The AHB Trace Macrocell is a trace source that makes bus information visible that cannot be inferred from the processor trace using an ETM:

- An understanding of multi-layer bus utilization.

- Software debug. For example, visibility of access to memory areas and data accesses.
- Bus event detection for trace trigger or filters, and for bus profiling.

See also Advanced High-performance Bus.

AMBA *See* Advanced Microcontroller Bus Architecture.

AMBA Trace Bus (ATB)
The AMBA Trace Bus transfers trace data through CoreSight infrastructure in a SoC. Trace sources are ATB masters, and sinks are ATB slaves. Link components provide both master and slave interfaces.

See also CoreSight.

APB-AP *See* Debug Access Port.

armar The ARM librarian, that enables you to create libraries of files, such as object files.

See also ARM Compiler toolchain *and* RealView Compilation Tools (RVCT).

armasm The ARM assembler.

See also ARM Compiler toolchain *and* RealView Compilation Tools (RVCT).

armcc The ARM compiler for C and C++ code.

See also ARM Compiler toolchain *and* RealView Compilation Tools (RVCT).

armlink The ARM linker.

See also ARM Compiler toolchain *and* RealView Compilation Tools (RVCT).

ARM Advanced SIMD Extension
ARM Advanced SIMD Extension is an optional component of ARMv7 architecture. It is a 64/128 bit hybrid SIMD technology targeted at advanced media and signal processing applications and embedded processors. It is implemented as part of the ARM core, but has its own execution pipelines and a register bank that is distinct from the ARM core register bank.

ARM Advanced SIMD Extension supports integer, fixed-point, and single-precision floating-point SIMD operations. These instructions are available in both ARM and Thumb®-2.

ARM Advanced SIMD Extension is also known as *ARM NEON Technology* (NEON™).

ARM Compiler toolchain

The ARM Compiler toolchain is a suite of tools, together with supporting documentation and examples, that enables you to write and build applications for the ARM family of processors. The ARM Compiler toolchain supersedes RealView Compilation Tools

See also armar, armasm, armcc, armlink, fromelf, *and* RealView Compilation Tools(RVCT).

ARM instruction A word that encodes an operation for an ARM processor operating in ARM state. ARM instructions must be word-aligned.

See also Thumb instruction, Thumb-2 instruction, *and* Thumb-2EE instruction.

ARM Profiler A plug-in to the ARM Workbench IDE that enables non-intrusive analysis of embedded software over long periods of time, on targets running at operational frequencies of up to 400 MHz. Targets can be *Real-Time System Models* (RTSMs) and hardware targets. ARM Profiler is provided with RVDS Professional edition.

This is not the same as the older ARM Profiler tool, `armprof`.

See also Real-Time System Model (RTSM), RealView ICE, *and* RealView Trace and RealView Trace 2.

ARM state A processor that is executing ARM instructions is operating in ARM state. The processor switches to Thumb state (and to recognizing Thumb instructions) when directed to do so by a state-changing instruction such as `BX` or `BLX`.

See also Jazelle® state, Thumb state, *and* ThumbEE state.

ARM TrustZone® technology

The hardware and software that enables security features to be integrated throughout a SoC device.

ARM Workbench IDE

ARM Workbench IDE is based around the Eclipse IDE, and provides additional features to support the ARM development tools provided in RVDS.

See also RealView Development Suite (RVDS).

ATB *See* AMBA Trace Bus.

AXI *See* Advanced eXtensible Interface.

BCD file *See* Board/Chip Definition (BCD) file.

Big-endian In the context of the ARM architecture, big-endian is defined as the memory organization in which the least significant byte of a word is at a higher address than the most significant byte.

See also Little-endian.

Board file RealView Debugger uses this term to refer to the top-level configuration file, normally called `rvdebug.brd`, that references one or more other configuration files. A board file contains:

- the Debug Configuration (connection-level) settings
- references to the Debug Interface configuration file that identifies the targets on the development platform
- references to any *Board/Chip Definition* (BCD) files assigned to a Debug Configuration.

See also Board/Chip Definition (BCD) file, Debug Configuration, Debug Interface, Development platform, *and* Target.

Board/Chip Definition (BCD) file
In the context of RealView Debugger, a BCD file enables you to define the memory map and memory mapped registers for a target development board or processor. Various BCD files are provided with RVDS for ARM development boards (for example `CP.bcd` for the Integrator®/CP development board) and processor core modules (for example `CM940T.bcd` for the ARM940T™ processor).

See also Board file *and* Debug Configuration.

Breakpoint unit In the context of RealView Debugger, a unit within a Chained breakpoint that combines with other breakpoint units to create a complex hardware breakpoint.

See also Chained breakpoint *and* Hardware breakpoint.

BSABI *See* ABI for the ARM Architecture (base standard).

Canonical Frame Address (CFA)
In DWARF, this is an address on the stack specifying where the call frame of an interrupted function is located.

Captive thread Captive threads are all threads that can be brought under the control of RVDS. Special threads, called non-captive threads, are essential to the operation of *Running System Debug* (RSD) and so are not under debugger control. Non-captive threads are grayed out in the GUI.

See also Running System Debug.

CFA *See* Canonical Frame Address.

Chained breakpoint In the context of RealView Debugger, a complex breakpoint that comprises multiple hardware breakpoint units.

See also Breakpoint unit, Conditional breakpoint, Data breakpoint, *and* Hardware breakpoint.

Chained tracepoint In the context of RealView Debugger, a complex tracepoint that comprises multiple tracepoint units.

See also Tracepoint *and* Tracepoint unit.

Conditional breakpoint
A breakpoint that has one or more condition qualifiers assigned. The breakpoint is activated when all assigned conditions are met, and either stops or continues execution depending on the action qualifiers that are assigned. The condition normally references the values of program variables that are in scope at the breakpoint location.

See also Chained breakpoint, Data breakpoint, Hardware breakpoint, Instruction breakpoint, Software breakpoint, *and* Unconditional breakpoint.

Core module In the context of an ARM Integrator development board, an add-on development board that contains an ARM architecture-based processor and local memory. Core modules can run standalone, or can be stacked onto Integrator development boards.

See also Integrator.

CoreSight CoreSight is an infrastructure that enables the debugging, monitoring, and optimization of performance of a complete *System on Chip* (SoC) design.

See also CoreSight ECT, CoreSight ETB, CoreSight ETM, Trace Funnel, *and* Trace Port Interface Unit.

CoreSight ECT CoreSight ECT is a control and access component that supports the interaction and synchronization of multiple triggering events within a SoC:

See also CoreSight, Cross Trigger Interface, Cross Trigger Matrix, *and* Embedded Cross Trigger.

CoreSight ETB CoreSight ETB is a trace sink that provides on-chip storage of trace data using a configurable sized RAM.

See also CoreSight, CoreSight ETB, Embedded Trace Buffer, *and* Embedded Trace Macrocell.

CoreSight ETM CoreSight ETM is a trace source that provides processor driven trace through an ATB compliant trace port.

See also AMBA Trace Bus, CoreSight, CoreSight ETB, *and* Embedded Trace Macrocell.

CPSR *See* Current Program Status Register.

Cross Trigger Interface (CTI)
The Cross Trigger Interface provides the interface between a component or subsystem and the Cross Trigger Matrix. The system requires a CTI for each subsystem that supports cross triggering.

See also CoreSight, CoreSight ECT, Cross Trigger Matrix, *and* Embedded Cross Trigger.

Cross Trigger Matrix (CTM)
The Cross Trigger Matrix combines the trigger requests generated from CTIs and broadcasts them to all CTIs as channel triggers. This enables subsystems to interact, cross trigger, with one another. CTMs can be connected together to increase the number of CTIs

See also CoreSight, CoreSight ECT, Cross Trigger Interface, *and* Embedded Cross Trigger.

CTI *See* Cross Trigger Interface.

CTM *See* Cross Trigger Matrix.

Current Program Status Register (CPSR)
A register containing the current state of control bits and flags.

See also Program Status Register *and* Saved Program Status Register.

DAP *See* Debug Access Port.

Data breakpoint A hardware breakpoint that activates when a given location is accessed in a specific way. The breakpoint can also check for a specific data value being access at the given location, if required.

See also Chained breakpoint, Conditional breakpoint, Hardware breakpoint, Instruction breakpoint, Software breakpoint, *and* Unconditional breakpoint.

DCC *See* Debug Communications Channel.

Debug Agent (DA) The Debug Agent resides on the target to provide target-side support for *Running System Debug* (RSD) in RealView Debugger. The Debug Agent and RealView Debugger communicate with each other using the *Debug Communications Channel* (DCC). This enables data to be passed between the debugger and the target using the ICE interface, without stopping the program or entering debug state.

See also Running System Debug *and* Debug Communications Channel.

Debug Access Port (DAP)

The Debug Access Port is a control and access component that enables debug access to the complete SoC through system master ports.

External read/write access to the internal interface is provided by the *JTAG Debug Port* (JTAG-DP). The JTAG-DP is a standard JTAG interface for debug access and provides standard JTAG access to an SoC through the DAP. It interfaces to the DAP internal bus.

Internal access to on-chip busses and other interfaces is provided by the *Access Ports* (APs). The three APs are:

- the *AHB Access Port* (AHB-AP) that provides an AHB-Lite master for access to a system AHB bus
- the *APB Access Port* (APB-AP) that provides an AMBA 3 APB master for access to the Debug APB that configures all CoreSight components
- the *JTAG Access Port* (JTAG-AP) that provides JTAG access to on-chip components and operates as a JTAG master port to drive JTAG chains throughout the SoC.

See also CoreSight.

Debug Communications Channel (DCC)

A debug communications channel enables data to be passed between RealView Debugger and the EmbeddedICE logic on the target using the JTAG interface, without stopping the program flow or entering debug state.

Debug Configuration

In the context of RealView Debugger, a Debug Configuration defines a debugging environment for the development platform that is accessed through a particular Debug Interface. Multiple Debug Configurations can be created for a Debug Interface, each providing a separate debugging environment to different development platforms, or different debugging environments to the same development platform.

All Debug Configurations are stored in the main RealView Debugger board file. Each configuration might reference one or more BCD files.

See also Board file, Board/Chip Definition (BCD) file, Debug Interface *and* Target.

Debug illusion

The experience that a debugger creates in the mind of the software developer. The key features of debug illusion include:

- mixed source code and disassembly
- a function call stack showing symbolic function prototypes with names and argument types

- display of variables using their source code name
- source level stepping and breakpoints.

This illusion is created by the debugger using data from the system being debugged and symbolic debug information from the code generation tool chain.

Debug Interface
In the context of RealView Debugger, the Debug Interface identifies the targets on your development platform, and provides the mechanism that enables RealView Debugger to communicate with those targets. The Debug Interface corresponds directly to a piece of hardware or a software simulator.

See also Debug Configuration *and* Target.

Development platform
Contains the components, either hardware or simulated, that you are using to develop your application. It can include:

- a development board, such as an Integrator/CP
- peripherals
- one or more ARM architecture-based processors
- CoreSight components
- one or more DSPs.

See also CoreSight *and* Target.

DSTREAM
A combined debug and trace unit that enables you to connect a software debugger to an ARM processor-based target using a hardware interface such as JTAG or *Serial Wire Debug* (SWD).

Profiling and tracing from the external trace port of a SoC with DSTREAM is not supported.

See also JTAG Interface Unit, *and* Serial Wire Debug (SWD).

Doubleword
In the context of the ARM architecture, a 64-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

ECT
See Embedded Cross Trigger.

Embedded assembler
Embedded assembler is assembler code that is included in a C or C++ file, and is separate from other C or C++ functions.

Embedded Cross Trigger (ECT)
The Embedded Cross Trigger provides a standard interconnect mechanism to pass debug or profiling events around the SoC. It comprises:

- *Cross Trigger Interface* (CTI)

- *Cross Trigger Matrix* (CTM).

See also CoreSight *and* CoreSight ECT.

Embedded Trace Buffer™ (ETB™)
The Embedded Trace Buffer provides logic inside the core that extends the information capture functionality of the Embedded Trace Macrocell.

See also CoreSight ETB *and* Embedded Trace Macrocell.

Embedded Trace Macrocell™ (ETM)
A block of logic, embedded in the hardware, that is connected to the address, data, and status signals of the processor. It broadcasts branch addresses, and data and status information in a compressed protocol through the trace port. It contains the resources used to trigger and filter the trace output.

See also CoreSight ETM *and* Embedded Trace Buffer.

EmbeddedICE® logic
The EmbeddedICE logic is an on-chip logic block that provides TAP-based debug support for ARM architecture-based processors. It is accessed through the TAP controller on the ARM architecture-based processor using the JTAG interface.

See also IEEE1149.1.

Emulator
In the context of target connection hardware, an emulator provides an interface to the pins of a real core (emulating the pins to the external world) and enables you to control or manipulate signals on those pins.

ETB
See Embedded Trace Buffer.

ETM
See Embedded Trace Macrocell.

ETV
See Extended Target Visibility.

Execution vehicle
Part of the debug target interface, execution vehicles process requests from the client tools to the target.

See also Debug Interface.

Execution view
The address of regions and sections after the image has been loaded into memory and started execution.

Extended Target Visibility (ETV)
Extended Target Visibility enables RealView Debugger to access features of the underlying target, such as chip-level information provided by the hardware manufacturer or SoC designer.

Filtering In the context of RealView Debugger Trace, a facility that enables you to refine the results of a trace capture that has already been performed in RealView Debugger. This is useful if you want to refine your area of interest within the display.

FIQ Fast Interrupt.

fromelf The ARM image conversion utility. This accepts ELF format input files and converts them to a variety of output formats. `fromelf` can also generate text information about the input image, such as code and data size.

See also ARM Compiler toolchain *and* RealView Compilation Tools (RVCT).

Halfword In the context of the ARM architecture, defined as a 16-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

Halted System Debug (HSD)
Usually used for OS aware debugging, *Halted System Debug* (HSD) means that a target can only be debugged when it is not running. Any target must be stopped before carrying out any analysis of the system. With the target stopped, RealView Debugger presents OS awareness information by reading and interpreting target memory.

See also Running System Debug (RSD).

Hardware breakpoint
A breakpoint that is implemented using non-intrusive additional hardware. Hardware breakpoints are the only method of halting execution when the location is in *Read Only Memory* (ROM) or Flash. Using a hardware breakpoint often results in the processor halting completely. This is usually undesirable for a real-time system.

See also Chained breakpoint, Conditional breakpoint, Data breakpoint, Instruction breakpoint, Software breakpoint, *and* Unconditional breakpoint.

Hint instruction A hint instruction provides information to the hardware that the hardware can take advantage of. An implementation can choose whether to implement hint instructions or not. If they are not implemented, they execute as `NOP`.

HSD *See* Halted System Debug.

HTM *See* AHB Trace Macrocell.

ICE Extension Unit A hardware extension to the EmbeddedICE logic that provides more breakpoint units.

IEEE 1149.1 The IEEE Standard that defines TAP. Commonly, but incorrectly, referred to as JTAG.

Immediate values Values that are encoded directly in the instruction and used as numeric data when the instruction is executed. Many ARM and Thumb instructions enable small numeric values to be encoded as immediate values within the instruction that operates on them.

Implementation defined
: In the context of the ARM architecture, this means that the behavior is not architecturally defined, but must be defined and documented by individual implementations.

In-Circuit Emulator
: A device enabling access to and modification of the signals of a circuit while that circuit is operating.

Input section
: Contains code or initialized data or describes a fragment of memory that must be set to zero before the application starts.

Instruction breakpoint
: A location in the image containing an instruction that, if executed, activates a breakpoint. The breakpoint activation can be delayed by assigning condition qualifiers, and subsequent execution of the image is determined by any actions assigned to the breakpoint.

  See also Conditional breakpoint, Data breakpoint, Hardware breakpoint, Software breakpoint, *and* Unconditional breakpoint.

Instruction Register (IR)
: When referring to a TAP controller, a register that controls the operation of the TAP.

Instruction Set System Model (ISSM)
: In the context of RVDS, a set of models that simulate the ARM Cortex™ family of processors. These models are provided with RVDS.

  See also Real-Time System Model (RTSM), RealView ARMulator ISS, Simulator, *and* SoC Designer Simulator.

Integrator
: A range of ARM hardware development platforms. Core modules are available that contain the processor and local memory.

  See also Core module.

Interworking
: A method of working that enables branches between ARM and Thumb code.

IRQ
: Interrupt Request.

ISSM
: *See* Instruction Set System Model (ISSM).

IT block
: A block of up to four instructions following the 16-bit Thumb-2 *If-Then* (`IT`) instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others.

Jazelle
: The Jazelle architecture extends the existing ARM architecture to enable direct execution of selected *Java Virtual Machine* (JVM) opcodes.

Jazelle state A processor that is executing Jazelle bytecode instructions is operating in Jazelle state.

See also ARM state, Thumb state, *and* ThumbEE state.

JTAG-AP *See* Debug Access Port.

JTAG-DP *See* Debug Access Port.

JTAG interface unit In the context of ARM RealView tools, a protocol converter that converts low-level commands from RVDS debuggers into JTAG signals to the processor, for example to the EmbeddedICE logic and the ETM.

See also RealView Debugger *and* Workbench Debugger.

Little-endian In the context of the ARM architecture, little-endian is defined as the memory organization in which the most significant byte of a word is at a higher address than the least significant byte.

See also Big-endian.

Load view The address of regions and sections when the image has been loaded into memory but has not yet started execution.

Memory hint In the context of the ARM architecture, a memory hint instruction enables a programmer to provide advance information to memory systems about future memory accesses, without actually loading or storing any data.

MPCore An integrated *Symmetric Multiprocessor System* (SMP) delivered as a traditional uniprocessor core. The chip contains up to four ARM1136J-S™ based CPUs with cache coherency.

MPU Multi-Processor Unit.

NEON *See* ARM Advanced SIMD Extension.

Normal and Secure Worlds
Effectively two virtual processors on a single physical processor. The Normal World processes operations that are not security-critical, and it delegates security-critical operations to the Secure World. Client applications reside and execute in the Normal World. Native services reside and execute in the Secure World. The secure parts of TrustZone Software run in the Secure World.

See also Secure monitor.

Normal World *See* Normal and Secure Worlds.

Output section A contiguous sequence of input sections that have the same RO, RW, or ZI attributes. The sections are grouped together in larger fragments called regions. The regions are grouped together into the final executable image.

See also Region.

OS-awareness OS-awareness is a feature provided by RealView Debugger that enables you to:

- debug applications running on an embedded OS development platform, such as a *Real-Time Operating System* (RTOS)
- present thread information and scope some debugging operations to specific threads.

PCH *See* PreCompiled Header.

PreCompiled Header (PCH)
A header file that is precompiled. This avoids the compiler having to compile the file each time it is included by source files.

Procedure Call Standard for the ARM Architecture (AAPCS)
Procedure Call Standard for the ARM Architecture defines how registers and the stack will be used for subroutine calls.

Profiling In the context of RealView Debugger Trace, the accumulation of statistics during execution of a program being debugged, to measure performance or to determine critical areas of code.

Program Counter (PC)
In the context of the ARM architecture, integer register R15.

Program Status Register (PSR)
Contains some information about the current program and some information about the current processor. Also referred to as Current PSR (CPSR), to emphasize the distinction between it and the Saved PSR (SPSR). The SPSR holds the value the PSR had when the current function was called, and which will be restored when control is returned.

An Enhanced Program Status Register (EPSR) contains an additional bit (the Q bit, signifying saturation) used by some ARM architecture-based processors, including the ARM9E.

See also Current Program Status Register *and* Saved Program Status Register.

Project template A collection of RealView Debugger and RVCT configuration files for specific target development platforms. These templates enable you to create a target-specific development project in the ARM Workbench IDE.

See also RealView Compilation Tools (RVCT), RealView Debugger, ARM Workbench IDE, *and* Workbench Debugger.

PSR *See* Program Status Register

PU Protection Unit.

Read-Only Position Independent (ROPI)
In the context of the ARM architecture, code or read-only data that can be placed at any address.

Read Write Position Independent (RWPI)
In the context of the ARM architecture, read/write data addresses that can be changed at runtime.

RealMonitor A small program that, when integrated into your target application or *Real-Time Operating System* (RTOS), enables you to observe and debug your target while parts of your application continue to run.

Real-Time System Model (RTSM)
An RTSM contains a hard-coded system containing one or more specific simulated processors and an emulation baseboard. Some RTSMs are provided with RVDS Professional edition.

See also Instruction Set System Model (ISSM), RealView ARMulator ISS (RVISS), ARM Profiler, Simulator, *and* SoC Designer Simulator.

RealView ARMulator® ISS (RVISS)
One of the ARM simulators supplied with RVDS.

RVISS is a collection of programs that simulate the instruction sets and architecture of various ARM processors. This provides instruction-accurate simulation and enables ARM and Thumb executable programs to be run on non-native hardware.

RVISS provides modules that model:

- the ARM processor core
- the memory used by the processor.

There are alternative predefined models for each of these parts. However, you can create your own models if a supplied model does not meet your requirements.

See also Instruction Set System Model (ISSM), Real-Time System Model (RTSM), Simulator, *and* SoC Designer Simulator.

RealView Compilation Tools (RVCT)
RVCT is a suite of tools, together with supporting documentation and examples, that enables you to write and build applications for the ARM family of processors.

See also armar, armasm, armcc, armlink, fromelf, *and* ARM Compiler toolchain.

RealView Debugger The latest debugger software from ARM that enables you to make use of a debug agent in order to examine and control the execution of software running on a debug target. RealView Debugger is supplied in both Windows and Red Hat Linux versions.

RealView Debugger Trace
Part of the RVDS product that extends the debugging capability with the addition of real-time program and data tracing. It is available from the RealView Debugger Code window.

See also RealView ICE *and* RealView Trace and RealView Trace 2.

RealView Development Suite (RVDS)
The suite of software development applications, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of processors. RVDS v3.1 and later can be obtained in both Professional and Standard editions. RVDS supersedes ARM Developer Suite™.

See also Real-Time System Model (RTSM).

RealView ICE A JTAG-based debug solution to debug software running on ARM architecture-based processors. RealView ICE host software is provided with RVDS. The RealView ICE run control unit must be purchased as a separate product.

See also RealView Debugger Trace, *and* RealView Trace and RealView Trace 2.

RealView Trace and RealView Trace 2
Works in conjunction with RealView ICE to provide real-time trace functionality for software running in leading edge System-on-Chip devices with deeply embedded processor cores. RealView Trace 2 also enables data streaming directly to ARM Profiler to perform real-time hardware platform profiling. The RealView Trace and RealView Trace 2 hardware units must be purchased as separate products.

See also RealView Debugger Trace, ARM Profiler, *and* RealView ICE.

Region In an image, a region is a contiguous sequence of one to three output sections (RO, RW, and ZI). A region typically maps onto a physical memory device, such as ROM, RAM, or peripheral.

See also Root region.

ROPI *See* Read-Only Position Independent.

Root region In an image, regions having the same load and execution address. A non-root region is a region that must be copied from its load address to its execution address.

RSD *See* Running System Debug.

RTSM *See* Real-Time System Model (RTSM).

Running System Debug (RSD)
: Used for OS-aware debugging, *Running System Debug* (RSD) means that a target can be debugged when it is running. This means that the debug target does not have to be stopped before carrying out any analysis of the system. RSD gives access to the application using a *Debug Agent* (DA) that resides on the target. The Debug Agent is scheduled along with other tasks in the system.

  See also Debug Agent *and* Halted System Debug (HSD).

RVCT
: *See* RealView Compilation Tools (RVCT).

RVDS
: *See* RealView Development Suite (RVDS).

RWPI
: *See* Read Write Position Independent.

Saved Program Status Register (SPSR)
: A register that holds a copy of what was in the Current Program Status Register before the most recent exception. Each exception mode has its own SPSR.

Scatter-loading
: Assigning the address and grouping of code and data sections individually rather than using single large blocks.

Section
: In the context of applications targeted at ARM architecture-based processors, a block of software code or data for an Image.

  See also Input section *and* Output section.

Secure monitor
: Reliably switches the ARM processor between Normal World and Secure World execution environments. The Secure monitor is transparent to TrustZone Software developers.

Secure World
: *See* Normal and Secure Worlds.

Semihosting
: A mechanism whereby the target communicates I/O requests made in the application code to the host system, rather than attempting to support the I/O itself.

Serial Wire Debug (SWD)
: Serial Wire Debug is a two-pin, bi-directional, data signal plus clock that replaces the 5-pin or 6-pin JTAG interface. The Serial Wire/JTAG debug port provides access to system memory peripherals and debug configuration registers.

Simple tracepoint
: A type of tracepoint that enables you to set trigger points, trace start and end points, or trace ranges for memory and data accesses.

  See also Tracepoints.

Simulator In the context of the ARM tools, a simulator executes non-native instructions in software (simulating a core).

See also Instruction Set System Model (ISSM), Real-Time System Model (RTSM), RealView ARMulator ISS, *and* SoC Designer Simulator.

SoC Designer Simulator

SoC Designer Simulator is part of the Carbon SoC Designer Plus toolset that can be used for fast modeling, simulation and debugging of complex *System-on-Chip* (SoC) designs. System and processor models created with the Carbon SoC Designer Plus tools can be debugged with Carbon SoC Designer Simulator in conjunction with RealView Debugger.

See also Instruction Set System Model (ISSM), Real-Time System Model (RTSM), RealView ARMulator ISS, *and* Simulator.

Software breakpoint A breakpoint that is implemented by replacing an instruction in memory with one that causes the processor to take exceptional action. Because instruction memory must be altered software breakpoints cannot be used where instructions are stored in read-only memory. Using software breakpoints can enable interrupt processing to continue during the breakpoint, making them more suitable for use in real-time systems.

See also Chained breakpoint, Conditional breakpoint, Data breakpoint, Hardware breakpoint, Instruction breakpoint, Software breakpoint, *and* Unconditional breakpoint.

SPSR Saved Program Status Register.

See also Program Status Register.

Stack Pointer (SP) Integer register R13.

Supervisor Call (SVC)

An instruction that causes the processor to call a programmer-specified subroutine. Used by the ARM standard C library to handle semihosting. This replaces *software interrupt* (SWI).

SVC *See* Supervisor Call.

SWI *See* Supervisor Call.

TAP Controller Logic on a device which enables access to some or all of that device for test purposes. The circuit functionality is defined in IEEE1149.1.

See also Test Access Port *and* IEEE1149.1.

Target

In the context of RealView Debugger, a Target is the part of your development platform to which RealView Debugger can connect, and on which debugging operations can be performed. A target can be:

- A runnable target, such as an ARM architecture-based processor. When connected to a runnable target, you can perform execution-related debugging operations on that target, such as stepping and tracing.
- A non-runnable CoreSight component. CoreSight components provide a system wide solution to real-time debug and trace.

See also CoreSight, Debug Configuration, *and* Debug Interface.

Target Vehicle

Target vehicles provide RVDS with a standard interface to disparate targets so that the debugger can connect easily to new target types without having to make changes to the debugger core software. The interface can be a hardware or software interface.

See also Instruction Set System Model (ISSM), Real-Time System Model (RTSM), RealView ARMulator ISS, RealView ICE, *and* SoC Designer Simulator.

TCM

Tightly Coupled Memory.

TDI

Test Data Input.

TDO

Test Data Output.

Thumb instruction

One halfword or two halfwords that encode an operation for an ARM architecture-based processor operating in Thumb state. Thumb instructions must be halfword-aligned.

See also ARM instruction, Thumb-2 instruction, *and* Thumb-2EE instruction.

Thumb state

A processor that is executing Thumb instructions is operating in Thumb state. The processor switches to ARM state (and to recognizing ARM instructions) when directed to do so by a state-changing instruction such as `BX`, `BLX`.

See also ARM state, Jazelle state, *and* ThumbEE state.

Thumb-2 instruction

Thumb-2 is a major enhancement of the Thumb instruction set, and is defined by ARMv6T2 and ARMv7M architectures. Thumb-2 provides almost exactly the same functionality as the ARM instruction set. It has both 16-bit and 32-bit instructions, and achieves performance similar to ARM code, but with code density similar to Thumb code.

See also ARM instruction, Thumb instruction, *and* Thumb-2EE instruction.

Thumb-2EE instruction

Thumb-2 Execution Environment (Thumb-2EE) is defined by ARMv7 architecture. The Thumb-2EE instruction set is based on Thumb-2, with some changes and additions to make it a better target for dynamically generated code, that is, code compiled on the device either shortly before or during execution.

See also ARM instruction, Thumb instruction, *and* Thumb-2 instruction.

ThumbEE state

A processor that is executing Thumb-2EE instructions is operating in ThumbEE state. In this state, the instruction set is almost identical to the Thumb instruction set. However, some instructions have modified behavior, some instructions are not available, and some new instructions become available.

See also ARM state, Jazelle state, *and* Thumb state.

TPA

Trace Port Analyzer.

TPIU

See Trace Port Interface Unit.

Trace Funnel

The Trace Funnel combines up to eight trace sources (ETM or HTM) on a single funnel. However, in this release, trace data can be captured only from a single ETM at a time.

See also AHB Trace Macrocell, CoreSight, CoreSight ETM, *and* Embedded Trace Macrocell.

Trace Port Interface Unit (TPIU)

The Trace Port Interface Unit is a trace sink that drains trace data off-chip to a TPA, such as RealView Trace.

See also CoreSight, CoreSight ETB, CoreSight ETM, *and* RealView Trace.

Tracepoint

A tracepoint can be set on a line of source code, a line of assembly code, or a memory address. In RealView Debugger, you can set a variety of tracepoints to determine exactly what program information is traced.

See also Chained tracepoint *and* Tracepoint unit.

Tracepoint unit

In the context of RealView Debugger, a unit within a Chained tracepoint that combines with other tracepoint units to create a complex tracepoint.

See also Chained tracepoint *and* Tracepoint.

Trigger

In the context of breakpoints, a trigger is the action of noticing that the breakpoint has been reached by the target and that any associated conditions are met.

In the context of tracing, a trigger is an event that instructs the debugger to stop collecting trace and display the trace information around the trigger position, without halting the processor. The exact information that is displayed depends on the position of the trigger within the buffer.

TrustZone Software

A secure software framework that enables best use of security extensions built into the ARM architecture. Used in single-processor ARM cores that can operate as two virtual CPUs.

Unconditional breakpoint

A breakpoint that does not have a conditional qualifier assigned. The breakpoint activates immediately it is hit, but subsequent image execution is determined by any actions assigned to the breakpoint.

See also Conditional breakpoint, Data breakpoint, Hardware breakpoint, Instruction breakpoint, Software breakpoint, *and* Unconditional breakpoint.

Undefined In the context of the ARM architecture, an attempt to execute an undefined instruction causes an Undefined Instruction exception.

Unpredictable In the context of the ARM architecture, the result of an unpredictable instruction cannot be relied upon. Unpredictable instructions or results must not represent security holes. Unpredictable instructions must not halt or hang the processor, or any parts of the system.

Veneer In the context of the ARM architecture, a small block of code used with subroutine calls when there is a requirement to change processor state or branch to an address that cannot be reached in the current processor state.

VFP A standard for floating-point coprocessors where several data values can be processed by a single instruction.

Watch In RealView Debugger, a watch is a variable or expression that you require the debugger to display at every step or breakpoint so that you can see how its value changes. The Watch pane is part of the RealView Debugger Code window. It displays the watchpoints you have defined.

Watchpoint In RVDS, this is a hardware breakpoint.

Word In the context of the ARM architecture, a word holds a value held in four contiguous bytes. A 32-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.