

# RealView<sup>®</sup> Debugger

Version 4.1

## Target Configuration Guide



# RealView Debugger

## Target Configuration Guide

Copyright © 2002-2010 ARM Limited. All rights reserved.

### Release Information

The following changes have been made to this document.

#### Change History

Date	Issue	Confidentiality	Change
April 2002	A	Non-Confidential	Release v1.5
September 2002	B	Non-Confidential	Release v1.6
February 2003	C	Non-Confidential	Release v1.6.1
September 2003	D	Non-Confidential	Release v1.6.1 for RealView Developer Suite v2.0
January 2004	E	Non-Confidential	Release v1.7 for RealView Developer Suite v2.1
December 2004	F	Non-Confidential	Release v1.8 for RealView Developer Suite v2.2
May 2005	G	Non-Confidential	Release v1.8 SP1 for RealView Developer Suite v2.2 SP1
March 2006	H	Non-Confidential	Release v3.0 for RealView Development Suite v3.0
March 2007	I	Non-Confidential	Release v3.1 for RealView Development Suite v3.1
September 2008	J	Non-Confidential	Release v4.0 for RealView Development Suite v4.0
27 March 2009	K	Non-Confidential	Release v4.0.1 for RealView Development Suite v4.0
28 May 2010	L	Non-Confidential	Release 4.1 for RealView Development Suite v4.1

### Proprietary Notice

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

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

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

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

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

### 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 Debugger Target Configuration Guide

	<b>Preface</b>	
	About this book .....	vii
	Feedback .....	xi
<b>Chapter 1</b>	<b>Introduction</b>	
	1.1 About connection configuration .....	1-2
	1.2 Default configuration files .....	1-7
	1.3 How configuration files are linked together .....	1-13
	1.4 What the configuration files contain .....	1-14
	1.5 Locating the configuration files .....	1-16
	1.6 Summary of supplied BCD files .....	1-19
<b>Chapter 2</b>	<b>Customizing a Debug Interface configuration</b>	
	2.1 About customizing a Debug Interface configuration .....	2-2
	2.2 About customizing a DSTREAM or RealView ICE Debug Interface configuration ..	2-3
	2.3 Customizing a DSTREAM or RealView ICE Debug Interface configuration for non-CoreSight development platforms .....	2-7
	2.4 Customizing a DSTREAM or RealView ICE Debug Interface configuration for development platforms containing CoreSight components .....	2-9
	2.5 Customizing an RVISS Debug Interface configuration .....	2-11
	2.6 Customizing an ISSM Debug Interface configuration .....	2-15
	2.7 Customizing an RTSM Debug Interface configuration .....	2-17
	2.8 Customizing a Model Library Debug Interface configuration .....	2-19
	2.9 Customizing a Model Process Debug Interface configuration .....	2-21
	2.10 Customizing a SoC Designer Debug Interface configuration .....	2-23
<b>Chapter 3</b>	<b>Customizing a Debug Configuration</b>	
	3.1 About customizing a Debug Configuration .....	3-2
	3.2 Viewing the Connection Properties .....	3-5

3.3	Changing connection settings .....	3-10
3.4	Loading a different board file .....	3-15
3.5	Hiding a Debug Configuration .....	3-18
3.6	Specifying connect and disconnect mode .....	3-19
3.7	Creating a target-specific Advanced_Information group .....	3-23
3.8	Configuring vector catch .....	3-24
3.9	Configuring Semihosting .....	3-29
3.10	Configuring the CLI commands for hardware cross-triggering .....	3-33
3.11	Configuring a connection sequence for multiple targets .....	3-36
3.12	Running CLI commands automatically on connection .....	3-41
3.13	Configuring RealMonitor for connections through DSTREAM or RealView ICE ...	3-43
3.14	Flash programming .....	3-50
3.15	Using the Thumb-2EE helper macro .....	3-51
3.16	Restoring the default connections and configurations .....	3-55
3.17	Preparing Debug Configurations for distribution .....	3-56
3.18	Distributing Debug Configurations to other machines and users .....	3-58
3.19	Example of setting up an Integrator board and processor core module .....	3-60
3.20	Troubleshooting Debug Configurations .....	3-65
<b>Chapter 4</b>	<b>Configuring Custom Memory Maps, Registers and Peripherals</b>	
4.1	About configuring custom memory maps, registers, and peripherals .....	4-2
4.2	Assigning a board, chip, or component group to a Debug Configuration .....	4-7
4.3	Using the supplied BCD files .....	4-12
4.4	Creating a BCD file to use as a template .....	4-13
4.5	Basic procedure for creating BCD files .....	4-15
4.6	Creating a new BCD file .....	4-16
4.7	Creating and naming a board, chip, or component group .....	4-18
4.8	Assigning board/chip definitions .....	4-21
4.9	Setting top of memory .....	4-24
4.10	Creating a memory map block .....	4-29
4.11	Creating an enumeration for setting register values .....	4-35
4.12	Creating a custom memory mapped register .....	4-37
4.13	Creating a custom peripheral .....	4-40
4.14	Creating the register tab for displaying custom registers and peripherals .....	4-44
4.15	Creating memory map rules .....	4-47
4.16	Setting up controlled memory blocks .....	4-49
4.17	Creating a concatenated register .....	4-54
4.18	Troubleshooting BCD files .....	4-58
<b>Chapter 5</b>	<b>Debug Configuration Tutorial</b>	
5.1	About the Debug Configuration tutorial .....	5-2
5.2	Before starting the tutorial .....	5-5
5.3	Creating a new Debug Configuration .....	5-6
5.4	Configuring the new Debug Configuration .....	5-7
5.5	Creating the EtherRouter.bcd file .....	5-8
5.6	Creating the AMDLANCE.bcd file .....	5-10
5.7	Creating the EtherRouter BOARD group .....	5-11
5.8	Creating the AMDLANCE CHIP group .....	5-12
5.9	Assigning board/chip definitions .....	5-13
5.10	Creating the memory map .....	5-15
5.11	Creating the enumerations for the register values .....	5-19
5.12	Creating a custom register .....	5-21
5.13	Creating the register tab for displaying custom registers .....	5-24
5.14	Setting up controlled memory map blocks .....	5-27
5.15	Creating memory map rules .....	5-29
5.16	Displaying the controlled memory map blocks .....	5-31
5.17	Creating a concatenated register .....	5-33
<b>Chapter 6</b>	<b>Programming Flash with RealView Debugger</b>	
6.1	Introduction to Flash programming with RealView Debugger .....	6-2

6.2	RealView Debugger files used for Flash programming .....	6-7
6.3	pakflash utility command syntax .....	6-10
6.4	Programming Flash on the ARM development boards .....	6-11
6.5	Programming Flash for a custom development platform .....	6-16
6.6	Gathering information about your development platform .....	6-19
6.7	Creating algorithms for a Flash type supported by RealView Debugger .....	6-20
6.8	Creating algorithms for a Flash type not provided with RealView Debugger .....	6-23
6.9	Creating the Flash-level and board-level AME files .....	6-29
6.10	Generating the FME file .....	6-34
6.11	Checking the FME file with the dispflash utility .....	6-38
6.12	Creating a BCD file .....	6-39
6.13	Programming an image into Flash .....	6-44
6.14	Troubleshooting .....	6-46

## Appendix A

### Connection Properties Reference

A.1	About connection properties reference .....	A-2
A.2	Debug Configuration generic groups and settings .....	A-6
A.3	Debug Configuration Advanced_Information settings reference .....	A-10
A.4	Memory mapping Advanced_Information settings reference .....	A-20

## Appendix B

### ISSM Configuration Reference

B.1	Cortex-A8 model configuration .....	B-2
B.2	Cortex-M0 model configuration .....	B-10
B.3	Cortex-M1 model configuration .....	B-17
B.4	Cortex-M3 model configuration .....	B-25
B.5	Cortex-R4 model configuration .....	B-33

# Preface

This preface introduces the *RealView® Debugger Target Configuration Guide*. It contains the following sections:

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

## About this book

RealView Debugger provides a powerful debugging tool for ARM® architecture-based software projects. This book describes how to configure connections between RealView Debugger and the targets on your development platform.

## Intended audience

This book has been written for developers who are using RealView Debugger to debug ARM architecture-based development projects. It assumes that you are an experienced software developer. It assumes that you are familiar with Chapter 3 *Target Connection* in the *RealView Debugger User Guide*.

## Examples

The examples given in this book have all been tested and shown to work as described. Your hardware and software might not be the same as that used for testing these examples, so it is possible that certain addresses or values might vary slightly from those shown, and some of the examples might not apply to you. In these cases you might have to modify the instructions to suit your own circumstances.

## Using this book

This book is organized into the following chapters:

### **Chapter 1 *Introduction***

This chapter introduces the connection configuration mechanism that is used by RealView Debugger. It is recommended that you read this chapter.

### **Chapter 2 *Customizing a Debug Interface configuration***

This chapter describes how to add a new Debug Configuration that enables you to access the targets on your development platform.

### **Chapter 3 *Customizing a Debug Configuration***

This chapter describes how you customize a Debug Configuration to meet your specific debugging requirements.

### **Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals***

This chapter is a reference to the tasks required to create a custom memory map, custom memory mapped registers, and peripherals for your development platform.

### **Chapter 5 *Debug Configuration Tutorial***

This chapter provides a tutorial on how to create a custom memory map, custom memory mapped registers, and peripherals for your development platform. It is suggested that you work through this chapter before you create your own custom memory map configurations.

### **Chapter 6 *Programming Flash with RealView Debugger***

This chapter describes how to use RealView Debugger to program Flash. It describes how to program the Flash types on the ARM boards. If you have a custom board and Flash type that is not supported by default, this chapter also describes how to create the files required to program your Flash type.

## Appendix

### Appendix A *Connection Properties Reference*

Read this appendix for details on the settings available for customizing a Debug Configuration.

### Appendix B *ISSM Configuration Reference*

Read this appendix for details on the configuration settings for *Instruction Set System Model* (ISSM) simulated targets provided with *RealView Development Suite* (RVDS).

## Typographical conventions

The typographical conventions are:

<i>italic</i>	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
<b>bold</b>	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
<u>monospace</u>	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
<i>monospace italic</i>	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
<b>monospace bold</b>	Denotes language keywords when used outside example code.
< and >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>

## Further reading

This section lists publications by ARM and by third parties.

See also:

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

### ARM publications

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

- *RealView Debugger Essentials Guide* (ARM DUI 0181)
- *RealView Debugger User Guide* (ARM DUI 0153)
- *RealView Debugger Trace User Guide* (ARM DUI 0322)
- *RealView Debugger RTOS Guide* (ARM DUI 0323)
- *RealView Debugger Command Line Reference Guide* (ARM DUI 0175).

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



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

- *RealView ARMulator ISS User Guide* (ARM DUI 0207).

For details on using and configuring *Real-Time System Models* (RTSMs), see:

- *RealView Development Suite Real-Time System Model User Guide* (ARM DUI 0424).

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

For details on ARM architectures, see:

- *ARMv6-M Architecture Reference Manual* (ARM DDI 0419)
- *ARMv7-M Architecture Reference Manual* (ARM DDI 0403)
- *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406).

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

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

- *DSTREAM Setting Up the Hardware* (ARM DUI 0481)
- *DSTREAM System and Interface Design Reference* (ARM DUI 0499)
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities* (ARM DUI 0498)
- *RealView ICE and RealView Trace Setting Up the Hardware* (ARM DUI 0515)
- *RealView ICE and RealView Trace System and Interface Design Reference* (ARM DUI 0517)
- *ARM Firmware Suite User Guide* (ARM DUI 0136)
- *ARM Firmware Suite Reference Guide* (ARM DUI 0102)
- *ARM RMHost User Guide* (ARM DUI 0137)
- *ARM RMTarget Integration Guide* (ARM DUI 0142).

## Other publications

For a comprehensive introduction to ARM architecture see:

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

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

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

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

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

For more information about the JTAG standard, see:

*IEEE Standard Test Access Port and Boundary Scan Architecture* (IEEE Std. 1149.1), available from the IEEE ([www.ieee.org](http://www.ieee.org)).

## Feedback

ARM welcomes feedback on this product, and its documentation.

### Feedback on this product

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

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

### Feedback on this book

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

- the title
- the number, ARM DUI 0182L
- 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 introduces the connection configuration mechanism used by RealView® Debugger. It contains the following sections:

- *About connection configuration* on page 1-2
- *Default configuration files* on page 1-7
- *How configuration files are linked together* on page 1-13
- *What the configuration files contain* on page 1-14.
- *Locating the configuration files* on page 1-16
- *Summary of supplied BCD files* on page 1-19.

## 1.1 About connection configuration

The following sections introduce the connection configuration features of RealView Debugger:

- *What is connection configuration?*
- *Connect to Target window on page 1-3*
- *Connection Properties window on page 1-6.*

### 1.1.1 What is connection configuration?

RealView Debugger enables you to debug applications running on hardware or software development platforms. Debugging applications requires that you set up connections to one or more targets on your development platform. These connections are associated with a specific *Debug Configuration* that defines a debugging environment for your development platform. You can create as many Debug Configurations as you require. You can also customize the debugging environment to use additional debugging features provided by RealView Debugger, such as OS-awareness.

A Debug Configuration is associated with a *Debug Interface*. The Debug Interface identifies the targets on your development platform, and provides the mechanism to enable RealView Debugger to communicate with those targets. Figure 1-1 shows the relationship between these entities in the Connect to Target window.

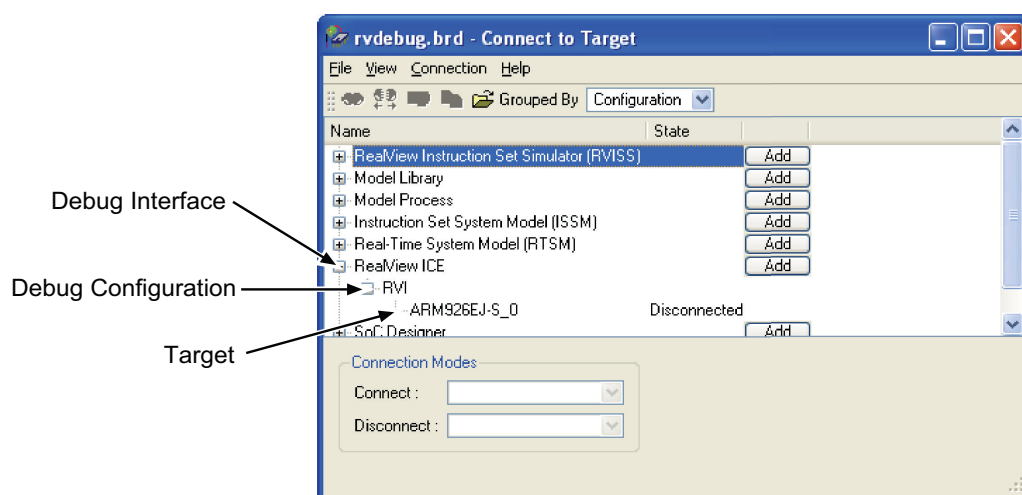


Figure 1-1 Relationships in the Connect to Target window

### Connection configuration summary

To summarize, connection configuration involves:

- Identifying the targets and configuring the Debug Interface parameters appropriate to your development platform.

The Debug Interfaces available to you depend on what you have installed.

#### ———— Note ————

Each Debug Interface provides a configuration dialog box that enables you to configure parameters for each target and for the associated Debug Interface unit.

- Creating and customizing a Debug Configuration that describes the debugging environment for your development platform.

- Optionally, defining the memory mapping information for your development platform. Memory mapping information is defined in separate configuration files, which you can associate with a Debug Configuration.

———— **Note** ————

Memory mapping configuration files are provided for supported ARM® architecture-based processors and development boards. If these are not suitable for your development platform, then you can create your own.

### See also

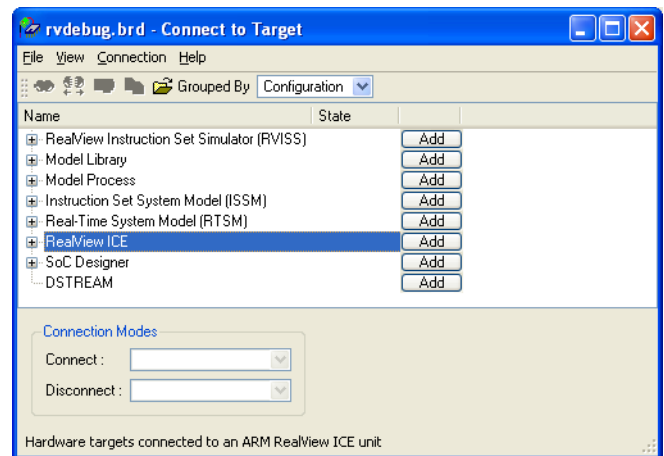
- *Connect to Target window*
- Chapter 2 *Customizing a Debug Interface configuration*
- Chapter 3 *Customizing a Debug Configuration*
- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals.*

## 1.1.2 Connect to Target window

The Connect to Target window displays:

- a list of the Debug Interfaces that are currently installed
- the Debug Configurations that you have created for each Debug Interface
- the targets within each Debug Configuration to which you can connect.

Figure 1-2 shows an example Connect to Target window with DSTREAM and RealView ICE host software installed.



**Figure 1-2 Example Connect to Target window**

There are two groupings available for viewing the connections:

**Target** The Target grouping shows all target connections as a list for each Debug Interface. The Debug Configuration name to which each target is associated is listed in the Configuration column, shown in Figure 1-2.

### Configuration

The Configuration grouping shows the target connections grouped by the related Debug Configuration. You are recommended to use this grouping when you add, copy, rename, customize, or delete a Debug Configuration. Figure 1-1 on page 1-2 shows an example of this grouping.

You can use the Connect to Target window to:

- Establish connections to the targets in your development platform, either individually or in a single operation.
- Add Debug Configurations and customize them to define specific debugging environments for your specific requirements. For example, you might have one Debug Configuration that you use for OS-awareness debugging, and another for basic debugging. The remainder of this document describes how to add and customize Debug Configurations.

## Debug Interface availability

All supported Debug Interfaces are listed in the Connect to Target window. Table 1-1 lists the supported Debug Interfaces. However, you can create target configurations for a specific Debug Interface only if the supporting software is installed for that interface.

**Table 1-1 Supported Debug Interfaces**

Debug Interface	Description
DSTREAM	<p>You must purchase a DSTREAM hardware unit and install the DSTREAM and RealView ICE host software.</p> <hr/> <p><b>Note</b></p> <p>This document assumes that you have installed the DSTREAM and RealView ICE host software and firmware.</p> <p>The DSTREAM and RealView ICE v4.0 host software is provided as an installation option with RVDS Professional edition.</p> <hr/>
Instruction Set System Model (ISSM)	<p>Supporting software is always installed. Use this Debug Interface to create Debug Configurations for <i>Instruction Set System Models</i> (ISSMs).</p> <p>A preconfigured model of the Cortex™-A8 processor is provided as an installation option. You can configure additional models in the Cortex family of processors.</p> <hr/>
Model Library	<p>Enables you to connect to your own <i>Cycle Accurate Debug Interface</i> (CADI) models defined in model libraries. This provides access to your model library files, and does not require the model to be running.</p> <hr/>
Model Process	<p>Enables you to connect to your own CADI model that is already running, either as a standalone executable or in a model simulator application.</p> <hr/>
Real-Time System Model (RTSM)	<p>Supporting software is always installed. Use this Debug Interface to create Debug Configurations for <i>Real-Time System Models</i> (RTSMs).</p> <p>Models of various ARM processors on an Emulation Baseboard are provided as an installation option with <i>RealView Development Suite</i> (RVDS) Professional edition. To create your own RTSMs, you must purchase the RealView System Generator software.</p> <hr/>

Table 1-1 Supported Debug Interfaces (continued)

Debug Interface	Description
RealView ICE	<p>You must purchase a RealView ICE hardware unit and install the RealView ICE host software.</p> <p>———— <b>Note</b> ————</p> <p>This document assumes that you have installed the DSTREAM and RealView ICE host software, and the firmware for RealView ICE v3.2, or later.</p> <p>The DSTREAM and RealView ICE v4.0 host software is provided as an installation option with RVDS Professional edition.</p>
RealView Instruction Set Simulator (RVISS)	Supporting software is always installed. Preconfigured <i>RealView Instruction Set Simulator</i> (RVISS) models of various ARM7, ARM9, and ARM11 processors are provided as an installation option. You can configure additional ARM7, ARM9, ARM10, ARM11, MPCore, and XScale models.
SoC Designer	You must purchase and install Carbon SoC Designer to create Debug Configurations for SoC Designer models.

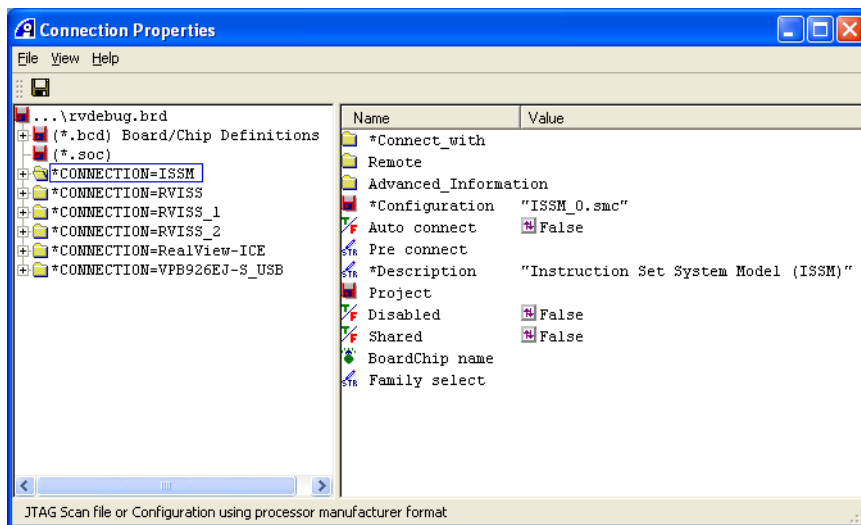
**See also**

- *Default configuration files* on page 1-7
- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*
- Carbon SoC Designer Plus, <http://carbondesignsystems.com/Products/SoCDesigner.aspx>.



### 1.1.3 Connection Properties window

RealView Debugger enables you to customize the configuration settings for a Debug Configuration. You can customize commonly used settings with the Connection Properties dialog box. However, for more advanced customizations, use the Connection Properties window. Figure 1-3 shows an example:



**Figure 1-3 Example Connection Properties window**

The main settings groups are:

- The `CONNECTION=DebugConfigurationName` groups, which contain the settings for each Debug Configuration, where *DebugConfigurationName* is the name that appears in the Connect to Target window.
- The `(*.bcd)` Board/Chip Definitions group, which lists all *Board/Chip Definitions* (BCD) files that RealView Debugger finds in its search path. They are listed in the order they are found.

#### Note

You can configure some items directly from the Connect to Target window, without having to use the Connection Properties window. For example, you can create, copy, rename, and delete Debug Configurations.

Your customizations are stored in various configuration files. The rest of this chapter describes the RealView Debugger connection-related configuration files in more detail.

#### See also

- *Debug Configuration settings* on page 1-8
- *The RealView Debugger search path* on page 1-17.

## 1.2 Default configuration files

Default configuration files are supplied as part of the RealView Debugger base product:

- Initially, the Debug Configurations defined in the default board file depend on what you installed during the RVDS installation, and can be:
  - RVISS configurations
  - ISSM configurations.
- BCD files defining the memory map related information for the standard ARM development boards:
  - Integrator™/AP, Integrator/CP, Emulation Baseboard, and Versatile board
  - the ARM7™, ARM9™, ARM10™, and ARM11™ Integrator family of Core Modules.
  - the Cortex-A8, Cortex-A9, and Cortex-M3 processors.

Debug Interface configuration files are also provided when you install additional connection products, such as DSTREAM or RealView ICE.

---

### **Note**

A version of the DSTREAM and RealView ICE host software is provided with RVDS Professional edition.

---

A SoC Designer configuration file is provided with RealView Debugger. However, you must purchase the Carbon SoC Designer Plus product separately.

The default name of the Debug Interface configuration file is based on the Debug Interface name. Because this name must be unique, RealView Debugger adds the *\_n* suffix. The name of the configuration file created is logged in the Diagnostic Log view. The configuration file is also stored in your home directory. For example, if you create a Debug Configuration for RealView ICE, then RealView Debugger might name it RVI\_1. Therefore, the Debug Interface configuration filename is called RVI\_1.rvc. However, the name of this file does not change if you later change the name of the Debug Configuration.

---

### **Note**

Some default configuration files are supplied as part of the RealView Debugger installation, see *What the configuration files contain* on page 1-14 for details.

---

See also:

- *Debug Configuration board file* on page 1-8
- *Debug Configuration settings* on page 1-8
- *DSTREAM and RealView ICE configuration files* on page 1-10
- *RVISS configuration files* on page 1-10
- *ISSM configuration files* on page 1-11
- *SoC Designer configuration files* on page 1-11
- *RTSM configuration files* on page 1-11
- *Board/Chip Definition files* on page 1-11.

### 1.2.1 Debug Configuration board file

RealView Debugger stores the Debug Configuration settings in a *board file*. A default board file is set up when you first install RealView Debugger. The board file is called `rvdebug.brd`, and is stored in your default settings directory:

```
C:\Documents and Settings\userID\Local Settings\Application
Data\ARM\rvdebug\version\shadowbase\etc
```

When you first use RealView Debugger after installation, a copy of the board file is created in your home directory. This means that if you damage your personal board file, you only have to delete it from your home directory and a new copy of the original default board file is placed there when you next run the debugger.

Although the board file is a text file, you must not edit the file manually. to make changes to a board file, or to create a new one, you must use:

- the Connect to Target window to add, copy, rename or delete Debug Configurations
- the Connection Properties dialog box to configure commonly used settings for a Debug Configuration.
- the Connection Properties window to configure advanced settings for a Debug Configuration.

#### See also

- *Connection Properties window* on page 1-6
- *The RealView Debugger home directory* on page 1-16.

### 1.2.2 Debug Configuration settings

The Debug Configuration settings describe the debugging environment for your development platform. Some configuration information is obtained directly from the development platform, such as the number and type of processors. Other configuration information, such as vector catch and memory mapping, is user-defined within RealView Debugger.

Therefore, a Debug Configuration:

- references the Debug Interface configuration file that identifies the targets on your development platform
- specifies debugger actions to take when a connection is made, for example, setting up the semihosting mechanism or running commands
- references one or more memory mapping configuration files appropriate to your development platform.

Each Debug Configuration you create and any modifications you make to the configuration are stored in a board file. Using internal configuration settings in this way means that you can change your Debug Configuration without leaving your RealView Debugger session.

Many Debug Configuration settings have default values where appropriate. Only settings that are changed from the default are stored in the board file, otherwise RealView Debugger uses the default value.

At the lowest level, RealView Debugger is able to access additional information about your development platform using a special group of settings, the `Advanced_Information` block, that is found in all the main Debug Configuration groups.

RealView Debugger can also access BCD files that contain information about a particular board or chip as supplied by the manufacturer. These files specify the memory map, custom registers, and peripherals of your development platform.

---

**Note**

---

You must configure memory map related details only in BCD files. All other settings must be configured in the Debug Configuration.

---

## The default Debug Configuration

RealView Debugger uses default configuration settings when you create a Debug Configuration. Table 1-2 shows the default values for the more common connection attributes.

**Table 1-2 Default values for common connection attributes**

Attribute	Default Value
Endianness	Determined from the target
Semihosting enabled	True
Software breakpoints	Enabled
Vector catch	True

If you are performing basic debugging operations, the default configuration might be sufficient. However, if you want to change the default debugging environment, you must customize the Debug Configuration. For example, you might want to disable semihosting for a connection.

## Customized Debug Configurations

By default, the same configuration settings are applied to all target connections established using that Debug Configuration. If you have multiple targets on your development platform, then you might want to use different settings for each target. For example, you might want to have semihosting enabled for one target connection, but not for connections to the other targets.

RealView Debugger enables you to define connection settings that are specific to:

- a single processor (for example, an ARM920T™ processor)
- a processor family (for example, the ARM9 processor family)
- a particular connection, if your development platform has identical processors (for example, ARM920T\_0 or ARM920T\_1).

### See also

- *What the configuration files contain* on page 1-14
- Chapter 3 *Customizing a Debug Configuration*
- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals*
- Appendix A *Connection Properties Reference*.

### 1.2.3 DSTREAM and RealView ICE configuration files

The configuration file (.rvc) is an XML formatted file that defines the target connections for hardware targets through a DSTREAM or RealView ICE Debug Interface unit.

You change the contents of this file when you modify the configuration of a DSTREAM or RealView ICE connection using the RVConfig utility.

When you install the DSTREAM and RealView ICE host software, the file rvi.rvc is installed in the directory identified in the ARM\_RVI\_T00LS environment variable.

RealView Debugger creates configuration files in the your home directory by default. However, you can specify a full path name in your Debug Configuration to use a different location, for example C:\test\_targets\rvi\_920T-tst.rvc.

#### ———— **Note** ————

Do not edit these files manually.

#### **See also**

- *The RealView Debugger search path on page 1-17*
- *Customizing a DSTREAM or RealView ICE Debug Interface configuration for non-CoreSight development platforms on page 2-7*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities.*

### 1.2.4 RVISS configuration files

An RVISS configuration file (.auc) is used to configure a processor model, such as an ARM940T™ simulated processor, on the RVISS Debug Interface. The following default configuration files are installed into your RealView Debugger home directory:

- default.auc, configured for an ARM7TDMI® target
- RVISS\_0.auc, configured for an ARM7TDMI target
- RVISS\_1.auc, configured for an ARM926EJ-S target
- RVISS\_2.auc, configured for an ARM1176JZ-S™ target.

You can also configure RVISS features using the various configuration files in the following directory:

`install_directory\RVARmulator\ARMulator\...\platform`

#### **See also**

- *Procedure for customizing an RVISS Debug Interface configuration on page 2-11*
- *RealView ARMulator ISS User Guide.*

### 1.2.5 ISSM configuration files

When you are working with an ISSM, such as the ARM Cortex™-A8 model, the Debug Interface configuration is defined using a System Model Configuration file `.smc`. A default configuration file `ISSM_0.smc` is installed into your RealView Debugger home directory, and is configured for a Cortex-A8 target.

#### See also

- *Procedure for customizing an ISSM Debug Interface configuration* on page 2-15.

### 1.2.6 SoC Designer configuration files

If you have created a system model with Carbon SoC Designer Plus, the Debug Interface configuration is defined using a System Model Configuration file `.smc`.

#### ———— Note ————

You must purchase Carbon SoC Designer Plus separately.

#### See also

- *Procedure for customizing a SoC Designer Debug Interface configuration (SoC Designer running)* on page 2-24
- Carbon SoC Designer Plus, <http://carbondesignsystems.com/Products/SoCDesigner.aspx>.

### 1.2.7 RTSM configuration files

When you are working with an RTSM, the Debug Interface configuration is defined using a System Model Configuration file `.smc`.

#### ———— Note ————

You must purchase RealView System Generator if you want to create your own models.

#### See also

- *Customizing an RTSM Debug Interface configuration* on page 2-17.

### 1.2.8 Board/Chip Definition files

*Board/Chip Definition* (BCD) files contain memory map related information, including memory mapped registers and peripheral registers.

Each board or chip is defined using a file named *filename.bcd*, where *filename* identifies the scope of the file contents, and can be:

- a processor, for example `CM940T.bcd`
- a development board, for example, `AP.bcd`
- a peripheral name or other meaningful name, for example, `CM920T_ETM.bcd`.

The BCD files supplied with RealView Debugger are stored in your default settings directory:

C:\Documents and Settings\userID\Local Settings\Application  
Data\ARM\rvdebug\version\shadowbase\etc

If you create your own BCD files, you are recommended to store them in your RealView Debugger home directory. RealView Debugger searches for BCD files using the sequence described in *The RealView Debugger search path* on page 1-17.

In general, you do not have to edit the supplied BCD files. However, where changes are required, use the Connection Properties window to make the required changes.

Descriptions of the individual settings are also available in the RealView Debugger online help.

It is suggested that you work through Chapter 5 *Debug Configuration Tutorial* if you want to create your own custom memory maps, registers, and peripherals.

**See also**

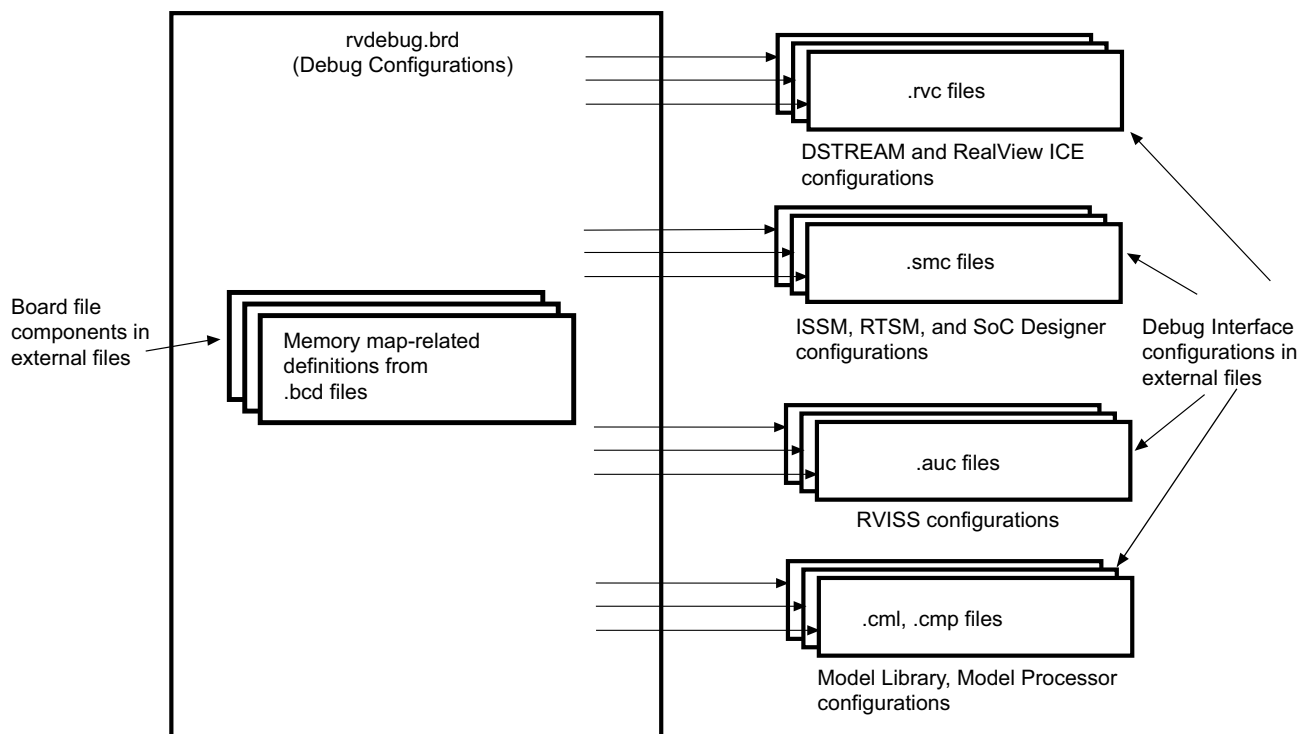
- *Memory mapping Advanced\_Information settings reference* on page A-20
- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals*
- Chapter 5 *Debug Configuration Tutorial*.

### 1.3 How configuration files are linked together

The board file might reference several other configuration files to form the complete configuration, for example:

- the DSTREAM or RealView ICE debug hardware configurations, \*.rvc
- the Board/Chip Definition files, \*.bcd.

This relationship is shown in Figure 1-4.



**Figure 1-4 Relationship between configuration files**

The configuration files, such as the RealView ICE configuration files (\*.rvc), contain the remaining information required to configure a specific Debug Interface. These files are not structured in the same way as the board files. They use the format required by the Debug Interface.

See also:

- *Default configuration files* on page 1-7
- *What the configuration files contain* on page 1-14
- *Locating the configuration files* on page 1-16
- *Summary of supplied BCD files* on page 1-19.



## 1.4 What the configuration files contain

The target-related configuration files that RealView Debugger stores in your home directory include the following files:

- \*.auc Configuration files for the RVISS Debug Interface.
- \*.brd Top-level board files. By default, there is one board file `rvdebug.brd` for each user. This file contains an entry for each Debug Configuration that you create, which references the filenames of the other configuration files, for example `.rvc` and `.bcd` files.

---

### Note

The first time you start RealView Debugger after installation, or after installing additional Debug Interface products, your user-specific `rvdebug.brd` is created or updated with the contents of the files in your default settings directory:

`C:\Documents and Settings\userID\Local Settings\Application Data\ARM\rvdebug\version\shadowbase\etc\*.brd`

---

Your user-specific `rvdebug.brd` file is modified when you change and save Debug Configuration settings in the Connection Properties window.

---

### Note

Only Debug Configuration settings must be configured in the `rvdebug.brd` file.

---

- \*.bcd BCD files contain chip-specific and board-specific settings groups as named configurations. You must assign these groups to a Debug Configuration to use them. The name of the `Advanced_Information` block in each group determines whether the settings are used when you connect to a target in the related Debug Configuration.

Many BCD files are supplied by hardware manufacturers to define the memory map related settings for specific targets and development boards. By default, these files are located in your default settings directory:

`C:\Documents and Settings\userID\Local Settings\Application Data\ARM\rvdebug\version\shadowbase\etc`

RealView Debugger lists all BCD files that it finds in its search path in the `(*.bcd)` Board/Chip Definitions group when you display the Connection Properties window.

You are recommended to make changes to BCD files using the Connection Properties window. If you change a supplied file or you create your own, you are recommended to store them in your RealView Debugger home directory.

---

### Note

Only memory map related settings must be configured in a BCD file.

---

- \*.cm1 Configuration files for the Model Library Debug Interface.
- \*.cmp Configuration files for the Model Process Debug Interface.
- \*.rvc Configuration files for connections through a DSTREAM or RealView ICE unit. You change the contents of one of these files when you modify the configuration using the RVConfig utility.
- \*.smc Configuration files for the following Debug Interfaces:
  - ISSM

- RTSM
- SoC Designer.

You change the contents of one of these files when you modify the configuration using the related configuration utility.

See also:

- *Connection Properties window* on page 1-6
- *Debug Configuration board file* on page 1-8
- *Board/Chip Definition files* on page 1-11
- *The RealView Debugger home directory* on page 1-16
- *Customizing a DSTREAM or RealView ICE Debug Interface configuration for non-CoreSight development platforms* on page 2-7
- *Customizing an ISSM Debug Interface configuration* on page 2-15
- *Customizing an RTSM Debug Interface configuration* on page 2-17
- *Customizing a Model Library Debug Interface configuration* on page 2-19
- *Customizing a Model Process Debug Interface configuration* on page 2-21
- *Procedure for customizing a SoC Designer Debug Interface configuration (SoC Designer running)* on page 2-24
- *Creating and naming a board, chip, or component group* on page 4-18
- Chapter 3 *Customizing a Debug Configuration*
- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*
- *RealView ARMulator ISS User Guide*
- Carbon SoC Designer Plus, <http://carbondesignsystems.com/Products/SoCDesigner.aspx>.

## 1.5 Locating the configuration files

The following sections describe how RealView Debugger directories are created when you install the base product, and how they are used to find the configuration files:

- *The install directory*
- *The RealView Debugger home directory*
- *The RealView Debugger search path on page 1-17*
- *Saving and restoring connection properties on page 1-17.*

### 1.5.1 The install directory

RealView Debugger must be able to locate the product installation directory so that it can locate plug-ins, data, and configuration files stored there.

#### See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 2 *The RealView Debugger Environment*
  - Appendix B *Configuration Files Reference*
  - Appendix E *RealView Debugger on Red Hat Linux*.

### 1.5.2 The RealView Debugger home directory

The first time you run RealView Debugger after installation, it creates the RealView Debugger home directory:

- On Windows, this is created in the Documents and Settings folder:  
C:\Documents and Settings\username\Application Data\ARM\rvdebug\version
- On Red Hat Linux, this is created in the directory ~/rvd.

RealView Debugger creates or copies files into this directory ready for your first debugging session.

#### See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 2 *The RealView Debugger Environment*.

### 1.5.3 The RealView Debugger search path

RealView Debugger searches several directories for the various configuration files, including the default file, `rvdebug.brd`. The search path the debugger uses is:

1. The current working directory, sometimes called the Start In directory.
2. Your RealView Debugger home directory.
3. Your default settings directory:

`C:\Documents and Settings\userID\Local Settings\Application  
Data\ARM\rvdebug\version\shadowbase\etc`

RealView Debugger searches all of these directories for workspace files and other configuration files. In particular, this is how \*.bcd files are found. When two or more files with the same filename are found in more than one of the searched directories, then the first file that is found with that name is loaded, and any subsequent files with that name are ignored.

#### See also

- *Board/Chip Definition files* on page 1-11
- *The RealView Debugger home directory* on page 1-16
- the following in the *RealView Debugger User Guide*:  
— Chapter 2 *The RealView Debugger Environment*.

### 1.5.4 Saving and restoring connection properties

When you are configuring RealView Debugger, you are recommended to keep backups of known-good configuration information before changing settings. If you want to restore the default configuration files, then you can do one of the following:

- Start RealView Debugger with the `--cleanstart` option.
- Delete your RealView Debugger home directory. The next time you start RealView Debugger it recreates the home directory and populates it with the default set of configuration files.

#### ————— Note —————

It is recommended that you make backups before using the worked examples in the rest of this book.

#### Using the automatic backup files

If you edit a board file, or a Board/Chip Definition file, RealView Debugger automatically renames the original file by adding a .bak file extension. Any previous backup copy of the file is deleted.

If you want to restore a backup file:

1. Exit the Connection Properties window without saving changes.
2. Delete the current file or files.
3. Rename the backup file to the original filename by deleting .bak from the name.

## Performing manual file or directory backups

For safer backups, you are recommended to make copies of the files in another location. The simplest policy is to save the whole directory when you make a backup, and then restore individual files when you want to revert changes.

---

### Note

---

If you restore the whole directory, then in addition to restoring the Connection Properties configuration information, you restore preferences that you might not want to change, for example workspace properties and window layout.

---

Creating a directory backup requires you to locate and copy the home directory to a safe place. You do not have to exit the debugger to do this.

Restoring previously backed up files requires you to:

1. Locate the backup that you want to restore from and the debugger home directory that RealView Debugger is using for your session.
2. Determine the files to restore.  
Deciding which files to restore depends on the type of configuration change you have performed. These hints might help:
  - If you have changed everything, or you are not sure what to restore, select all the files listed in *What the configuration files contain* on page 1-14 to restore the Connection Properties window to its original state.
  - If you have configured or reconfigured a chip or board using BOARD, CHIP or COMPONENT groups, select appropriate files from the \*.bcd set.  
If you have created new \*.bcd files in your debugger home directory, you might also want to delete them from that directory. However, a \*.bcd file is not used unless it is explicitly referenced from a Debug Configuration.
  - If you have changed target-specific settings by changing items in the related Debug Interface configuration dialog box, select the configuration files for the related target connections you have reconfigured. For example, if you have changed settings for connections through DSTREAM or RealView ICE using the RVConfig utility, then select the \*.rvc files.
3. Copy the backup files to the debugger home directory.

### See also

- *Default configuration files* on page 1-7
- *What the configuration files contain* on page 1-14
- *The RealView Debugger home directory* on page 1-16
- the following in the *RealView Debugger User Guide*:
  - *Syntax of the rvdebug command* on page 2-2
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*.

## 1.6 Summary of supplied BCD files

The supplied BCD files include:

AP.bcd	A description of the registers and memory map for the ARM Integrator/AP, including Flash programming support.
AT91SAM9*.bcd	A description of the registers and memory map for various Atmel AT91SAM9 Evaluation Kits.
BeagleBoard_OMAP3530_Complete.bcd	A description of the complete set of registers and memory map for the Texas Instruments OMAP 3530 applications processor.
BeagleBoard_OMAP3530_Essentials.bcd	A description of the essential registers and memory map for the Texas Instruments OMAP 3530 applications processor.
CM7TDMI.bcd	A description of the registers and memory map for the ARM7TDMI processor core module.
CM720T.bcd	A description of the registers and memory map for the ARM720T™ processor core module.
CM740T.bcd	A description of the registers and memory map for the ARM740T processor core module.
CM920T.bcd	A description of the registers and memory map for the ARM920T processor core module.
CM920T_ETM.bcd	A description of the registers and memory map for the ARM920T-ETM processor core module.
CM922TExcal.bcd	A description of the registers and memory map for the ARM922T™ XA10 processor core module with four cross-trigger channels.
CM926EJS.bcd	A description of the registers and memory map for the ARM926EJ-S processor core module.
CM940T.bcd	A description of the registers and memory map for the ARM940T processor core module.
CM946ES.bcd	A description of the registers and memory map for the ARM946E-S™ processor core module.
CM966ES.bcd	A description of the registers and memory map for the ARM966E-S™ processor core module.
CM10200.bcd	A description of the registers and memory map for the ARM10200™ processor core module.
CM10200E.bcd	A description of the registers and memory map for the ARM10200E™ processor core module.
Cortex-M3.bcd	A description of the registers and memory map for the Cortex-M3 processor core module.
CP.bcd	A description of the registers and memory map for the ARM Integrator/CP, including Flash programming support. This description is also suitable for use with Integrator core modules.

CT11MP.bcd	A description of the registers and cross-trigger enabling and disabling commands for the ARM11™ MPCore™ processor core module.
EB-CortexM3.bcd	A description of the registers and memory map for the ARM EB with Cortex-M3 platform.
EB_1136.bcd	A description of the registers and memory map for the ARM EB with CT1136EJ-S platform, including Flash programming support.
EB_CT11MPCore.bcd	A description of the registers and memory map for the ARM EB with CT11MPCore platform, including Flash programming support.
ect.bcd	A description of the registers provided in the ARM <i>Embedded Cross Trigger</i> (ECT). Use for targets that contain a memory-mapped ECT.
<p style="text-align: center;"><b>Note</b></p> <p style="text-align: center;">Do not use this file for the CoreSight™ ECT.</p>	
Eva17T.bcd	A description of the registers and memory map for the ARM Evaluator-7T, including Flash support and a description of the internal registers for both KS32C50100 and S3C4510B processors.
ICYMX35.bcd	A description of the registers and memory map for the ICYMX35 Starter Board, including Flash programming support.
iMX25_PDK.bcd, iMX31_PDK.bcd, iMX51_PDK.bcd	A description of the registers and memory map for the Freescale iMX25, iMX31, and iMX51 Development Kits.
iMX27_LiteKit.bcd, iMX31_LiteKit.bcd	A description of the registers and memory map for the Freescale iMX27 and iMX31 Lite Development Kits, including Flash support.
iMX31.bcd	A description of the registers and memory map for the Freescale iMX31 development board.
MCBSTM32.bcd	A description of the registers and memory map for the STMicroelectronics 32-bit ARM Cortex-M3 Based Microcontroller.
MCBSTM32E.bcd	A description of the registers and memory map for the STMicroelectronics 32-bit ARM Cortex-M3 Based Microcontroller.
OMAP34x-II_MDP.bcd	A description of the registers and memory map for the Texas Instruments Zoom OMAP34x-II Mobile Development Platform.
OMAP3530_Complete.bcd	A description of the complete set of registers and memory map for the Texas Instruments OMAP 3530 board.
OMAP3530_Essentials.bcd	A description of the essential registers and memory map for the Texas Instruments OMAP 3530 board.
OMAP5912.bcd	A description of the registers and memory map for the Texas Instruments OMAP 5912board, including Flash programming support.
PB-A8.bcd	A description of the registers and memory map for the Versatile Platform for Cortex-A8, including Flash programming support.

PB1176JZF-S.bcd	A description of the registers and memory map for the Versatile Platform for PB1176JZF-S, including Flash programming support.
PBARM11MPCore.bcd	A description of the registers and memory map for the Versatile Platform for ARM11 MPCore, including Flash programming support.
PBX-A9.bcd	A description of the registers and memory map for the Platform Baseboard Explore for Cortex-A9, including Flash programming support.
philips_lpc*.bcd	Descriptions of the registers and memory map for the various Philips LPCxxx platforms, including Flash programming support.
phyCORE_iMX27.bcd	A description of the registers and memory map for the PHYTEC phyCORE-iMX27 Multimedia Processor with ARM926EJ-S, including Flash programming support.
phyCORE_iMX31.bcd	A description of the registers and memory map for the Freescale iMX31 development Kit, including Flash programming support.
phyCORE_iMX35.bcd	A description of the registers and memory map for the PHYTEC phyCORE-iMX35 Board, including Flash programming support.
realmonitor.bcd	A description of the registers to enable RealMonitor support.
S5PC100_Complete.bcd	A description of the complete set of registers and memory map for the Samsung S5PC100 board.
S5PC100_Essentials.bcd	A description of the essential registers and memory map for the Samsung S5PC100 board, including Flash programming support.
SMDK2450_NAND_BOOT.bcd	A description of the registers and memory map for the Samsung S3C2450 Development Kit SMDK2450 with NAND Flash boot.
SMDK2450_NOR_BOOT.bcd	A description of the registers and memory map for the Samsung S3C2450 Development Kit SMDK2450 with NOR Flash boot, including Flash programming support.
thumb2ee.bcd	<p>Provided for ARM architecture-based processors that support the Thumb<sup>®</sup>-2 Execution Environment (Thumb-2EE). This file references the include file thumb2ee.inc. When you load an image, RealView Debugger loads the include file. This defines the handleraddr(handlerIndex) macro that returns the address of the Thumb-2EE handler, where handlerIndex is the number of the required handler. You can use this macro when setting breakpoints and tracepoints, and for viewing disassembly and memory locations.</p> <p style="text-align: center;"><b>————— Note —————</b></p> <p>To see the definition of the handleraddr() macro after it is loaded, enter the CLI command:</p> <p>SHOW handleraddr</p>
TMS320DM355.bcd	A description of the registers and memory map for the TMS320DM355 Digital Media System on Chip.



VAB926EJ-s.bcd      A description of the registers and memory map for the Versatile Application Baseboard for ARM926EJ-S.

vpb926ej-s\_256KB.bcd

A description of the registers and memory map for the Integrator Versatile Platform for ARM926EJ-S. This includes the 256KB variant of the Flash part included with the board.

vpb926ej-s\_64KB.bcd A description of the registers and memory map for the Integrator Versatile Platform for ARM926EJ-S. This includes the 64KB variant of the Flash part included with the board.

zoran4100.bcd      A description of the registers and memory map for the Zoran ZJP4100.

Other BCD files are provided, or might be included with RealView Debugger plug-ins such as OS-awareness downloads.

See also:

- *Board/Chip Definition files* on page 1-11
- *Summary of files used to program Flash on supported development platforms* on page 6-2
- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals*
- the following in the *RealView Debugger User Guide*:
  - *Configuring embedded cross-triggering* on page 7-25
  - *Configuring CoreSight embedded cross-triggering* on page 7-27.

# Chapter 2

## Customizing a Debug Interface configuration

This chapter describes how you can customize a Debug Interface configuration to identify the targets on your development platform, and configure the target-specific parameters. It contains the following sections:

- *About customizing a Debug Interface configuration on page 2-2*
- *About customizing a DSTREAM or RealView ICE Debug Interface configuration on page 2-3*
- *Customizing a DSTREAM or RealView ICE Debug Interface configuration for non-CoreSight development platforms on page 2-7*
- *Customizing a DSTREAM or RealView ICE Debug Interface configuration for development platforms containing CoreSight components on page 2-9*
- *Customizing an RVISS Debug Interface configuration on page 2-11*
- *Customizing an ISSM Debug Interface configuration on page 2-15*
- *Customizing an RTSM Debug Interface configuration on page 2-17*
- *Customizing a Model Library Debug Interface configuration on page 2-19*
- *Customizing a Model Process Debug Interface configuration on page 2-21*
- *Customizing a SoC Designer Debug Interface configuration on page 2-23.*

## 2.1 About customizing a Debug Interface configuration

You configure the way that RealView Debugger connects to, and interacts with, your targets using connection properties contained in board file entries. A target might be:

- an ARM® architecture-based processor (simulated or hardware)
- a CoreSight™ component.

Using Debug Configuration entries enables you to configure:

- debugger to target connection details, such as the Debug Interface type and instance, TAP controller positions, and connection interface address
- debugger actions taken when a connection is made, for example running commands.

Debug Configurations are stored in your board file (.brd), which is located in your RealView Debugger home directory. Debug Interface configuration files might also be located in this directory, for example .rvc files.

It is recommended that you back up this directory before starting the examples described in this chapter, so that you can restore your original configuration later.

---

### **Note**

---

You cannot customize a Debug Configuration when the debugger is connected to a target on that Debug Configuration.

---

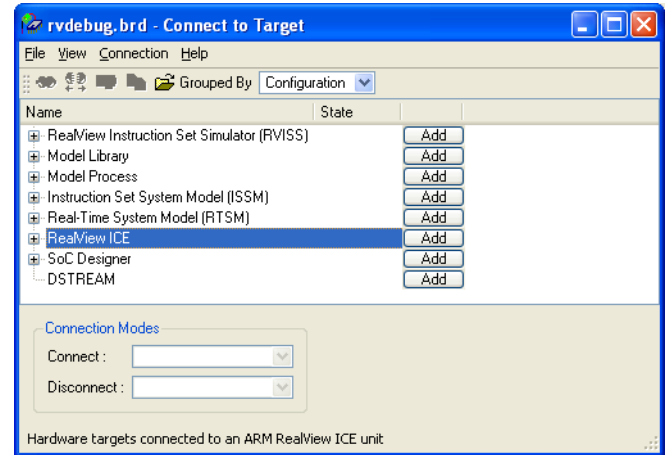
There are descriptions of the general layout and controls of the RealView Debugger settings windows, including the Connection Properties window, in the RealView Debugger online help topic *Changing Settings*. This chapter assumes that you are familiar with the procedures described in this help topic.

See also:

- *About connection configuration* on page 1-2
- *Saving and restoring connection properties* on page 1-17
- *Relationship between connection properties and Debug Configurations* on page 3-2
- *Restoring the default connections and configurations* on page 3-55
- *Troubleshooting Debug Configurations* on page 3-65.
- the following in the *RealView Debugger Essentials Guide*:
  - *Supported Debug Interfaces* on page 1-4

## 2.2 About customizing a DSTREAM or RealView ICE Debug Interface configuration

When you install the DSTREAM and RealView ICE host software, it adds configuration files to your RealView Debugger installation. The board file in your RealView Debugger home directory is updated so that the DSTREAM and RealView ICE Debug Interface entries appears in the Connect to Target window. Figure 2-1 shows an example:



**Figure 2-1 DSTREAM and RealView ICE Debug Interfaces in the Connect to Target window**

### Note

The DSTREAM or RealView ICE unit must be purchased separately.

A DSTREAM or RealView ICE Debug Interface configuration is stored in a configuration file with the .rvc extension. When you create a DSTREAM or RealView ICE Debug Configuration, RealView Debugger creates a corresponding .rvc file for that Debug Configuration. You can use this Debug Configuration as the basis for any new DSTREAM or RealView ICE Debug Configurations that you create.

See also:

- *DSTREAM and RealView ICE device names for supported CoreSight components on page 2-4*
- *Considerations when customizing DSTREAM or RealView ICE Debug Interface configurations on page 2-4*
- *Recommended settings for an ARM Integrator development board on page 2-5*
- *Customizing a DSTREAM or RealView ICE Debug Interface configuration for non-CoreSight development platforms on page 2-7*
- *Customizing a DSTREAM or RealView ICE Debug Interface configuration for development platforms containing CoreSight components on page 2-9*
- the following in the *RealView Debugger Essentials Guide*:
  - *Supported Debug Interfaces on page 1-4*
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration on page 3-8 if you want to create a different RealView ICE Debug Configuration.*

### 2.2.1 DSTREAM and RealView ICE device names for supported CoreSight components

Table 2-1 shows the DSTREAM and RealView ICE device name for each CoreSight component that is supported.

**Table 2-1 DSTREAM and RealView ICE device names for CoreSight Components**

CoreSight component	Device name
CoreSight <i>Debug Access Port</i> (DAP)	ARMCS-DP
CoreSight <i>Cross Trigger Interface</i> (CTI)	CSCTI
CoreSight <i>Embedded Trace Buffer</i> <sup>®</sup> (ETB <sup>®</sup> )	CSETB
CoreSight <i>Embedded Trace Macrocell</i> <sup>™</sup> (ETM <sup>™</sup> )	CSETM
CoreSight <i>AHB Trace Macrocell</i> (HTM) <sup>a</sup>	CSHTM
CoreSight <i>Instrumentation Trace Macrocell</i> (ITM) <sup>a</sup>	CSITM
Provide generic support for your own CoreSight-compatible device that can be used by RealView Debugger.	CSREG
CoreSight <i>Program Flow Trace Macrocell</i> <sup>™</sup> (PTM) <sup>a</sup>	CSPTM
CoreSight <i>Serial Wire Output</i> (SWO) <sup>a</sup>	CSSWO
CoreSight Trace Funnel	CSTFunnel
CoreSight <i>Trace Port Interface Unit</i> (TPIU)	CSTPIU
JTAG Debug Port	ARMJTAG-DP
JTAG Access Port for the ARM1136JF-S	ARM1136JFS-JTAG-AP
JTAG Access Port for the ARM1156T2F-S <sup>™</sup>	ARM1156T2FS-JTAG-AP
JTAG Access Port for the ARM1176JZ-F <sup>™</sup>	ARM1176ZF-JTAG-AP
Serial Wire Debug Port	ARMSW-DP

a. Although this component can be added as a target, you cannot capture trace from it in this release.

#### Note

The ARMCS-DP, ARMJTAG-DP, and ARMSW-DP devices do not appear in the Connect to Target window.

### 2.2.2 Considerations when customizing DSTREAM or RealView ICE Debug Interface configurations

Be aware of the following when configuring devices:

- Autoconfiguration does have side effects and might be intrusive. Where this is not acceptable, you must manually add the devices to the scan chain.
- Autoconfiguring a CoreSight system is a two-stage process:
  1. Autoconfigure the scan chain to detect the CoreSight DAP.
  2. If the CoreSight DAP contains a ROM table, then read the ROM table of the to determine the devices that are connected to it.

If the CoreSight DAP does not contain a ROM table, then manually add the devices that are connected to it. As a minimum, you have only to add the target processor if you do not want to capture trace information.

- If you want to view the CoreSight topology in RealView Debugger, then you must assign CoreSight Associations to your Debug Interface configuration.  
Search for "association file", including the quotes, in the ARM Information Center for more details about CoreSight Association files.
- Autoconfiguration does not work for development platforms containing non-ARM targets. In this case, you must manually add all devices to the scan chain.
- Where a target runs at much lower CPU clock speed than an ARM architecture-based processor, the JTAG clock speed for a multiprocessor platform might be lower than a platform containing only ARM architecture-based processors.  
In general, the JTAG clock speed over the 20 pin JTAG connection can be set to no more than one eighth of the slowest CPU on the JTAG scan chain in the multiprocessor platform. If your platform supports **RTCK** (return clock), then you might be able to take advantage of Adaptive Clocking.
- When setting up a custom DSTREAM or RealView ICE connection you might also want to set some of the RealView Debugger settings. For example, if you are using *Running System Debug* (RSD) you also have to disable vector catch and semihosting.

#### See also

- the following in the *RealView Debugger RTOS Guide*:  
— *Customizing an OS-aware Debug Configuration* on page 2-6.
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*.

### 2.2.3 Recommended settings for an ARM Integrator development board

Table 2-2 lists the recommended settings to use for an ARM Integrator™ development board.

**Table 2-2 Recommended settings for an ARM Integrator development board**

Setting	Value
Reset Type	<b>nSRST+nTRST</b>
Perform TAP Reset on first connect	Selected
Reset On Disconnect (Default)	Deselected
nSRST Hold Time (ms)	<b>100</b>
nSRST Post Reset Delay (ms)	<b>1000</b>
nTRST Hold Time (ms)	<b>100</b>
nTRST Post Reset Delay (ms)	<b>100</b>
TAP reset via State Transitions	Selected
Target nSRST + nTRST Linked	Deselected
Post Reset State (device-specific setting)	<b>STOPPED</b>

See also:

- *About customizing a DSTREAM or RealView ICE Debug Interface configuration on page 2-3.*

## 2.3 Customizing a DSTREAM or RealView ICE Debug Interface configuration for non-CoreSight development platforms

The difference between customizing a Debug Interface configuration for use with DSTREAM or RealView ICE is the Debug Interface that you use. The following procedure uses the RealView ICE Debug Interface as an example.

To customize a RealView ICE Debug Interface configuration containing only ARM architecture-based processors and that does not support CoreSight:

1. Open the RVConfig utility:
  - a. Select the **Configuration** grouping from the Grouped By list.
  - b. Expand the RealView ICE Debug Interface to see the existing Debug Configurations.
  - c. Right-click on the name of the Debug Configuration to be customized to display the context menu.
  - d. Select **Configure...** from the context menu to display the RVConfig utility.

Alternatively, create a new RealView ICE Debug Configuration, which automatically displays the RVConfig utility.
2. Locate the RealView ICE unit connected to your development platform, or enter the IP address or host name of the unit.
3. Click **Connect** to connect the RVConfig utility to the RealView ICE unit.
4. If your development platform contains only ARM architecture-based processors, then click **Auto Configure Scan Chain** to detect all the targets. The devices are added to the list in the order they are positioned on the scan chain.

———— **Note** ————

You cannot autoconfigure if your development platform contains non-ARM targets.

5. Select the device in the left pane to configure it using the supplied template.  
For example, for an ARM940T™ you might want to set the Code Sequence Code Address or Code Sequence Code Size, to specify target memory available to RealView ICE.
6. Select Advanced in the left pane to configure advanced settings.  
For example, you might want to select the **Reset On Disconnect (Default)** check box so that a target is reset when you disconnect.
7. Select **Save** from the **File** menu to save your changes.
8. Select **Exit** from the **File** menu to close the RVConfig utility. You can connect to the targets in the Debug Configuration in the usual way.

See also:

- *Customizing a DSTREAM or RealView ICE Debug Interface configuration for development platforms containing CoreSight components* on page 2-9
- *Considerations when customizing DSTREAM or RealView ICE Debug Interface configurations* on page 2-4
- the following in the *RealView Debugger Essentials Guide*:  
— *Supported Debug Interfaces* on page 1-4



- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
  - *Connecting to a target* on page 3-27.
- *DSTRAM and RealView ICE Using the Debug Hardware Configuration Utilities*.

## 2.4 Customizing a DSTREAM or RealView ICE Debug Interface configuration for development platforms containing CoreSight components

You can use DSTREAM or RealView ICE to connect to a development platform that incorporates CoreSight components. How you configure a development platform that supports CoreSight depends on your requirements:

- If you do not want to capture trace information, then you have only to add the CoreSight DAP and the target processor.
- If you want to capture trace information, then you must add the CoreSight *Debug Access Port* (DAP), the target processor, and the appropriate CoreSight components you want to use for tracing.

In both cases, you autoconfigure the scan chain to detect the CoreSight DAP. If your CoreSight DAP contains a ROM table, you can then read the ROM table to determine the devices that are connected to the DAP. If no ROM table is provided, or is corrupted, you must manually add the remaining devices as required.

The difference between customizing a Debug Interface configuration for use with DSTREAM or RealView ICE is the Debug Interface that you use. The following procedure uses the RealView ICE Debug Interface as an example.

To customize a RealView ICE Debug Interface configuration for a development platform containing CoreSight components:

1. Create a new RealView ICE Debug Configuration:
  - a. Select the **Configuration** grouping from the Grouped By list.
  - b. Expand the RealView ICE Debug Interface to see the existing Debug Configurations.
  - c. Click **Add** to create a new RealView ICE Debug Configuration. The RVConfig utility is automatically displayed.
2. Select your RealView ICE Debug Interface unit, if it appears in the RVConfig browser. Alternatively, enter the IP address or host name of the unit.
3. Click **Connect** to connect the RVConfig utility to the RealView ICE Debug Interface unit.
4. Click **Auto Configure Scan Chain**. The ARMCS-DP target is detected and added to the scan chain schematic diagram.
5. If the ARMCS-DP device has a ROM table:
  - a. Right-click on the ARMCS-DP device in the scan chain schematic diagram to display the context menu.
  - b. Select **Read coresight ROM table** from the context menu. The devices connected to the CoreSight DAP are added to the scan chain schematic diagram.

If the ARMCS-DP device does not have a ROM table:

  - a. Click **Add Device...** to display the Add Device dialog box.
  - b. Expand the group in the Registered Devices list containing the device to be added.
  - c. Select the required device from the Registered Devices list.
  - d. Click **OK**. The device is added to the scan chain schematic diagram.
  - e. Repeat these steps for each additional device on your development platform.
6. Select the device in the left pane, and configure the parameters in the panel.
7. Select Advanced in the left pane, and configure the advanced settings.

8. Select **Save** from the **File** menu to save your changes.
9. Select **Exit** from the **File** menu to close the RVConfig utility. You can connect to the targets in your new Debug Configuration in the usual way.

See also:

- *Customizing a DSTREAM or RealView ICE Debug Interface configuration for non-CoreSight development platforms* on page 2-7
- *DSTREAM and RealView ICE device names for supported CoreSight components* on page 2-4
- *Considerations when customizing DSTREAM or RealView ICE Debug Interface configurations* on page 2-4
- the following in the *RealView Debugger Essentials Guide*:
  - *Supported Debug Interfaces* on page 1-4
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
  - *Changing the name of a Debug Configuration* on page 3-17
  - *Connecting to a target* on page 3-27
  - *Viewing information about the target topology* on page 3-36.
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities.*

## 2.5 Customizing an RVISS Debug Interface configuration

RealView Debugger includes support for the *RealView Instruction Set Simulator* (RVISS) simulator to emulate the instruction sets of different ARM processors and their supporting architecture.

Debug Configurations are provided for the following processors:

- ARM7TDMI®
- ARM926EJ-S™
- ARM1176JZF-S™.

You can:

- use the Debug Configurations without any additional modification
- modify any of the Debug Configurations to use a different target
- create a new Debug Configuration.

See also:

- *Procedure for customizing an RVISS Debug Interface configuration*
- *Considerations when customizing RVISS Debug Interface configurations* on page 2-13.

### 2.5.1 Procedure for customizing an RVISS Debug Interface configuration

To customize an RVISS Debug Interface configuration:

1. Open the RVISS Debug Interface configuration dialog box:
  - a. Select the **Configuration** grouping from the Grouped By list.
  - b. Expand the RealView Instruction Set Simulator (RVISS) Debug Interface to see the existing Debug Configurations.
  - c. Right-click on the Debug Configuration to be customized to display the context menu.
  - d. Select **Configure...** from the context menu to display the ARMulator Configuration dialog box. Figure 2-2 on page 2-12 shows an example:

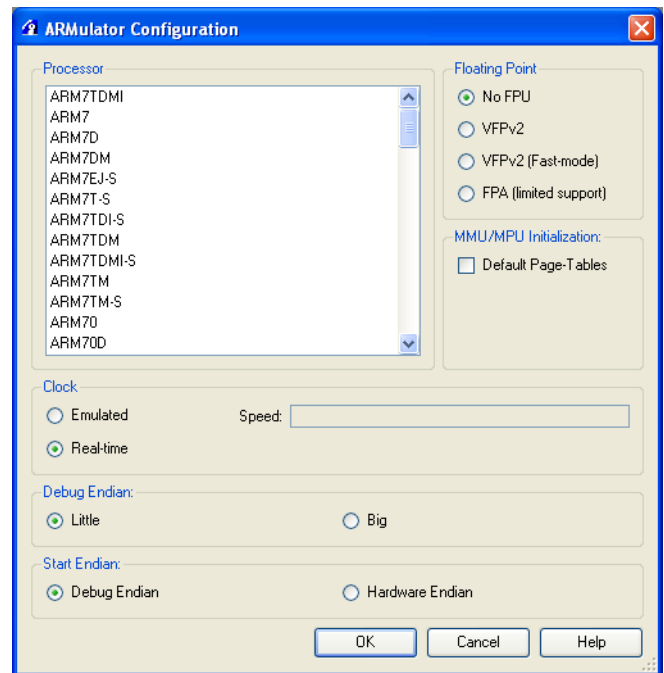


Figure 2-2 ARMulator configuration dialog box

2. Select the Processor to be simulated in this Debug Configuration, and specify the other settings as required:

**Processor**

Use the list to specify which ARM processor you want RVISS to simulate. The list of processors includes all available variants currently supported for RVISS Debug Configurations.

**Clock**

Choose between simulating a processor clock running at a speed that you can specify, or executing instructions in real-time. If you select **Emulated**, specify the speed in Hz. You cannot include the units. For example, enter **50000** to specify 50kHz.

Changing this value does not affect the real time taken to run a program. Instead, it affects the values that the semihosting `time()` functions return to the program.

**Debug Endian**

Select the byte order of the target processor. This setting:

- sets RealView Debugger to work with the appropriate byte order
- sets the byte order of models that do not have a CP15 coprocessor
- sets the byte order of models that do have a CP15 coprocessor if the Start Endian option is set to **Debug Endian**.

**Start Endian**

Select the way in which the byte order of RVISS models that have a CP15 coprocessor is determined:

- Select the **Debug Endian** radio button to instruct the model to use the byte order set in the Debug Endian group.
- Select the **Hardware Endian** radio button to instruct the model to simulate the behavior of real hardware.

Table 2-3 shows the possible combinations of Debug Endian and Start target Endian.

**Table 2-3 RealView ARMulator ISS Endian settings**

Usage	Debug Endian	Start target Endian
A target that is always little-endian. This is the default.	Little	Debug Endian
A target that is always big-endian	Big	Debug Endian
A big-endian target where the code and the processor start in little-endian mode, and switches to big-endian in the initialization code	Big	Hardware Endian

### Floating Point

Use the radio buttons to specify the VFP coprocessor included with some ARM CPUs. The default is No FPU.

### MMU/MPU Initialization

If you are simulating a processor with an active *Memory Management Unit* (MMU) or *Memory Protection Unit* (MPU), select **Default Page-Tables**.

### See also

- *Using VFP options* on page 2-14
- the following in the *RealView Debugger Essentials Guide*:
  - *Supported Debug Interfaces* on page 1-4
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
- *RealView ARMulator ISS User Guide*
- *ARM Architecture Reference Manual*.

## 2.5.2 Considerations when customizing RVISS Debug Interface configurations

Be aware of the following features when using the ARMulator Configuration dialog box:

- Some processors are not supported when you are connected using RVISS, for example the Cortex™ family of processors.
- There is no simulated ETM functionality. Therefore, you cannot use the autoconnect analyzer feature
- Use the Additional Modules control group to define *Floating Point Accelerator* (FPA) and VFP options.

#### ———— **Note** ————

VFPv1 is no longer supported.

- If you choose **Real-time**, the default clock frequency specified in the configuration file default.amt is used, that is DEFAULT\_CPUSPEED=20MHz.
- If you want to use features of RVISS such as the Tracer and memory map features, you must set these up in the RVISS configuration files (see *RVISS configuration files* on page 1-10):
  - To enable the Tracer feature, see in the *RealView ARMulator ISS User Guide*.

- If you are using the RVISS memory map feature, the memory map is displayed in the **Mapfile** tab of the Registers view when you connect to an RVISS target.

See the *RealView ARMulator ISS User Guide* for more details.

## Using VFP options

Be aware of the following when selecting the Floating Point options:

- The ARM1136JF-S™, ARM1156T2F-S™ and ARM1176JZF-S models already include a VFP11. Any VFP option you choose in RealView Debugger is ignored. If you choose the **FPA (limited support)** option then it is rejected.
- The ARM1136J-S, ARM1156T2-S and ARM1176JZ-S models do not include a VFP. Therefore, you must not use any Floating Point options with these models, because they might produce incorrect results. In addition:
  - A target configured with the **VFPv2 (Fast-mode)** option fails to connect with a message claiming (incorrectly) that it is trying to connect to an ARM7 processor.
  - A target configured with the **VFPv2** option connects but might result in incorrect operation.
  - If you choose the **FPA (limited support)** option then it is rejected.

## 2.6 Customizing an ISSM Debug Interface configuration

The ISSM Debug Interface enables you to connect to the supported ARM Cortex models. These are instruction level models with a fixed set of system components (such as UART and Timer) and a preconfigured memory map.

A Debug Configuration is provided for the Cortex-A8 model. You can:

- use the Debug Configuration without any additional modification
- modify the Debug Configuration to use a different target
- create a new Debug Configuration.

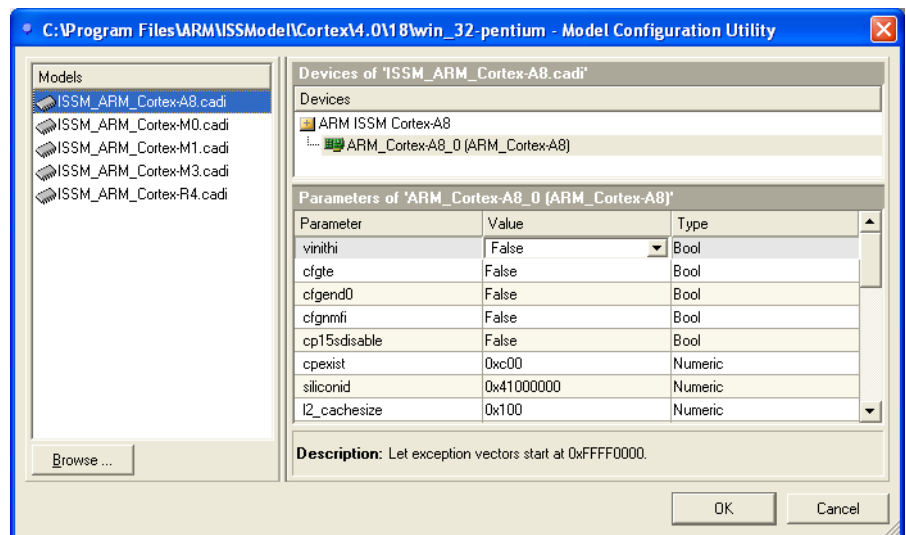
See also:

- *Procedure for customizing an ISSM Debug Interface configuration*
- *Considerations when customizing an ISSM Debug Interface configuration* on page 2-16.

### 2.6.1 Procedure for customizing an ISSM Debug Interface configuration

To customize an ISSM Debug Interface configuration:

1. Open the ISSM Debug Interface configuration dialog box:
  - a. Select the **Configuration** grouping from the Grouped By list.
  - b. Expand the Instruction Set System Model (ISSM) Debug Interface to see the existing Debug Configurations.
  - c. Right-click on the Debug Configuration to be customized to display the context menu.
  - d. Select **Configure...** from the context menu to display the Model Configuration Utility dialog box.
  - e. Select a model from the Models list. Figure 2-3 shows an example:



**Figure 2-3 Model Configuration Utility dialog box**

2. If the required models are listed, then skip this step. Otherwise:
  - a. Click **Browse...** to display the Browse for Folder dialog box.
  - b. Locate the directory containing your ISSM models.
  - c. Click **OK**.

The models are listed in the Models pane.



3. Select the required model in the Models pane.
4. Configure the settings in the Parameters of *model\_name* pane.  
The parameters available depend on the model you have selected.
5. Click **OK** to save your changes and close the dialog box.

**See also**

- *Cortex-A8 model configuration* on page B-2
- *Cortex-M0 model configuration* on page B-10
- *Cortex-M1 model configuration* on page B-17
- *Cortex-M3 model configuration* on page B-25
- *Cortex-R4 model configuration* on page B-33
- the following in the *RealView Debugger Essentials Guide*:  
— *Supported Debug Interfaces* on page 1-4
- the following in the *RealView Debugger User Guide*:  
— *About creating a Debug Configuration* on page 3-8.

**2.6.2 Considerations when customizing an ISSM Debug Interface configuration**

Be aware of the following when customizing an ISSM Debug Interface configuration:

- You cannot pass arguments to the function main using the ARGUMENTS command, LOAD command, or the Load Image dialog box. If you want to pass arguments to your image, use the *semihosting-cmd\_line* configuration parameter for the processor model you are configuring.

**See also**

- *Cortex-A8 model configuration* on page B-2
- *Cortex-M0 model configuration* on page B-10
- *Cortex-M1 model configuration* on page B-17
- *Cortex-M3 model configuration* on page B-25
- *Cortex-R4 model configuration* on page B-33

## 2.7 Customizing an RTSM Debug Interface configuration

If you want to debug your application on a *Real-Time System Model* (RTSM), then you must add an RTSM Debug Configuration. You can then customize your RTSM Debug Configuration as required.

### ———— Note ————

Some RTSMs are provided with *RealView Development Suite* (RVDS) Professional edition. You must purchase the RealView System Generator software to create your own RTSMs.

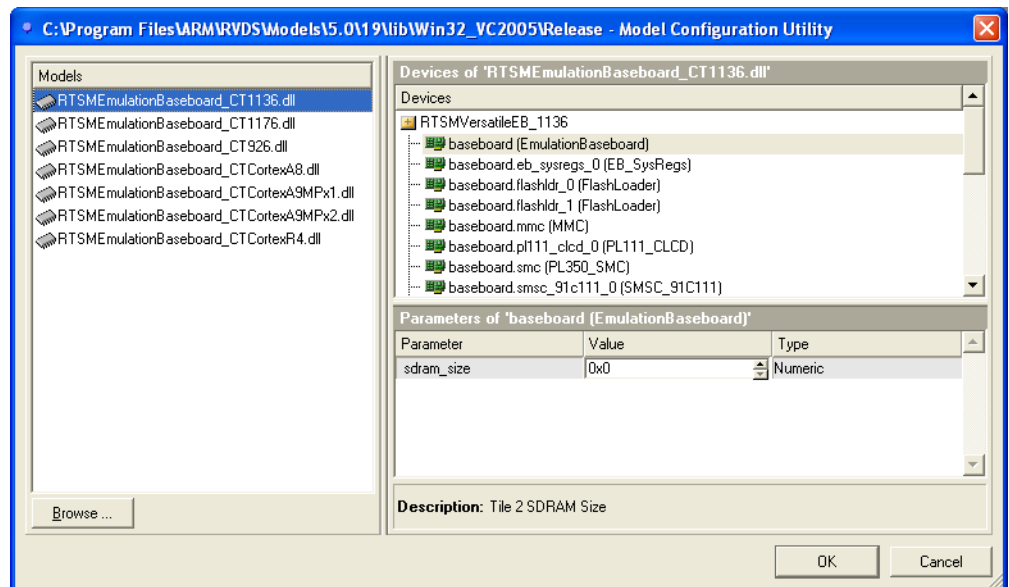
See also:

- *Procedure for customizing an RTSM Debug Interface configuration.*

### 2.7.1 Procedure for customizing an RTSM Debug Interface configuration

To customize an RTSM Debug Interface configuration:

1. Open the Model Configuration Utility dialog box:
  - a. Select the **Configuration** grouping from the Grouped By list.
  - b. Expand the Real-Time System Model (RTSM) Debug Interface to see the existing Debug Configurations.
  - c. Right-click on the RTSM Debug Configuration to be customized to display the context menu.
  - d. Select **Configure...** from the context menu to display the Model Configuration Utility dialog box.
  - e. Select a model from the Models list. Figure 2-4 shows an example:



**Figure 2-4 Model Configuration Utility dialog box (RTSM)**

2. If the required RTSMs are listed, then skip this step. Otherwise:
  - a. Click **Browse...** to display the Browse for Folder dialog box.
  - b. Locate the directory containing the shared library for your RTSM.
  - c. Click **OK**.

The models are listed in the Models pane.

3. Select the required RTSM in the Models pane.
4. Configure the settings as required for each component of the RTSM in the Parameters of 'device\_name' pane.
5. Click **OK** to save your changes and close the dialog box.

**See also**

- the following in the *RealView Debugger Essentials Guide*:
  - *Supported Debug Interfaces* on page 1-4
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8.
- *RealView Development Suite Real-Time System Model User Guide* for details of the RTSM configuration parameters.

## 2.8 Customizing a Model Library Debug Interface configuration

If you have made any changes to a Model Library model, and you have an existing Debug Configuration for that model, then you must customize the Debug Interface configuration to reflect the changes.

See also:

- *Procedure for customizing a Model Library Debug Interface configuration*
- *Considerations when customizing an Model Library Debug Interface configuration on page 2-20*

### 2.8.1 Procedure for customizing a Model Library Debug Interface configuration

To customize a Model Library Debug Interface configuration:

1. Open the Model Library Debug Interface configuration dialog box:
  - a. Select the **Configuration** grouping from the Grouped By list.
  - b. Expand the Model Library Debug Interface to see the existing Debug Configurations.
  - c. Right-click on the Debug Configuration to be customized to display the context menu.
  - d. Select **Configure...** from the context menu to display the Model Configuration Utility dialog box.
  - e. Select a model from the Models list. Figure 2-5 shows an example:

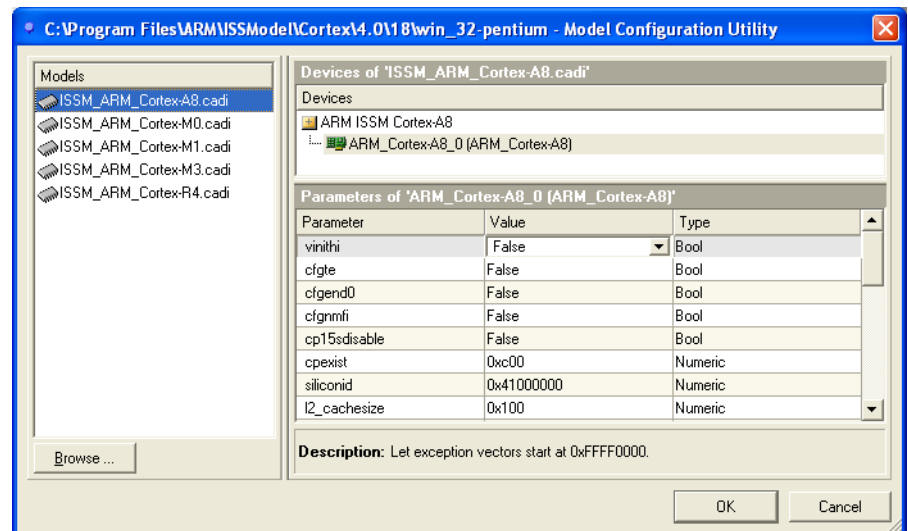


Figure 2-5 Model Configuration Utility dialog box

2. If the required models are listed, then skip this step. Otherwise:
  - a. Click **Browse...** to display the Browse for Folder dialog box.
  - b. Locate the directory containing your CADI models.
  - c. Click **OK**.

The models are listed in the Models pane.
3. Select the required model in the Models pane.
4. Configure the settings in the Parameters of *model\_name* pane.

The parameters available depend on the model you have selected.

5. Click **OK** to save your changes and close the dialog box.

**See also**

- the following in the *RealView Debugger Essentials Guide*:
  - *Supported Debug Interfaces* on page 1-4
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8.

## 2.8.2 Considerations when customizing an Model Library Debug Interface configuration

You cannot pass arguments to the function main using the ARGUMENTS command, LOAD command, or the Load Image dialog box. If you want to pass arguments to your image, use the `semihosting-cmd_line` configuration parameter for the processor model you are configuring.

———— **Note** —————

Be aware that this interface enables you to add an arbitrary CADI model that might not implement this parameter or might provide a different method of passing arguments.

**See also**

- *Customizing a Model Library Debug Interface configuration* on page 2-19.

## 2.9 Customizing a Model Process Debug Interface configuration

If you have made any changes to a Model Process model, and you have an existing Debug Configuration for that model, then you must customize the Debug Interface configuration to reflect the changes.

See also:

- *Procedure for customizing a Model Process Debug Interface configuration*
- *Considerations when customizing an Model Process Debug Interface configuration on page 2-22.*

### 2.9.1 Procedure for customizing a Model Process Debug Interface configuration

To customize a Model Process Debug Interface configuration:

1. Open the Model Process Debug Interface configuration dialog box:
  - a. Select the **Configuration** grouping from the Grouped By list.
  - b. Expand the Model Process Debug Interface to see the existing Debug Configurations.
  - c. Right-click on the Debug Configuration to be customized to display the context menu.
  - d. Select **Configure...** from the context menu to display the Model Configuration Utility dialog box.  
The Model Configuration Utility dialog box displays any running simulations.
  - e. Select running simulations from the Model Configuration Utility dialog box.

———— **Note** ————

You cannot configure parameters for running simulations.
2. If the required models are listed, then skip this step. Otherwise:
  - a. Click **Browse...** to display the Browse for Folder dialog box.
  - b. Locate the directory containing your CADI models.
  - c. Click **OK**.  
The models are listed in the Models pane.
3. Select the required model in the Models pane.
4. Configure the settings in the Parameters of *model\_name* pane.  
The parameters available depend on the model you have selected.
5. Click **OK** to save your changes and close the dialog box.

#### See also

- the following in the *RealView Debugger Essentials Guide*:  
— *Supported Debug Interfaces* on page 1-4
- the following in the *RealView Debugger User Guide*:  
— *About creating a Debug Configuration* on page 3-8.

## 2.9.2 Considerations when customizing an Model Process Debug Interface configuration

Be aware of the following when customizing an Model Process Debug Interface configuration:

- You cannot pass arguments to the function main using the ARGUMENTS command, LOAD command, or the Load Image dialog box. If you want to pass arguments to your image, use the `semihosting-cmd_line` configuration parameter for the processor model you are configuring.

### See also

- *Customizing a Model Process Debug Interface configuration* on page 2-21.

## 2.10 Customizing a SoC Designer Debug Interface configuration

If you have made any changes to a SoC Designer model, and you have an existing Debug Configuration for that model, then you must customize the Debug Interface configuration to reflect the changes.

---

### Note

---

You must purchase the Carbon SoC Designer Plus software separately.

---

See also:

- *Procedure for customizing a SoC Designer Debug Interface configuration (SoC Designer not running)*
- *Procedure for customizing a SoC Designer Debug Interface configuration (SoC Designer running) on page 2-24*
- Carbon SoC Designer Plus, <http://carbondesignsystems.com/Products/SoCDesigner.aspx>.

### 2.10.1 Procedure for customizing a SoC Designer Debug Interface configuration (SoC Designer not running)

To customize a SoC Designer Debug Interface configuration when Carbon SoC Designer Simulator is not currently running:

1. Launch Carbon SoC Designer Simulator with the currently configured model:
  - a. Select the **Configuration** grouping from the Grouped By list.
  - b. Expand the SoC Designer Debug Interface to see the existing Debug Configurations.
  - c. Right-click on the Debug Configuration to be customized to display the context menu.
  - d. Select **Configure...** from the context menu to display the Select SoC Designer Session dialog box. This states that the saved session is not running. Also, the Saved Session SocDesigner dialog box is displayed.
  - e. Click **Yes** on the prompt dialog box to launch Carbon SoC Designer Simulator. Carbon SoC Designer Simulator starts, and the configured model is loaded. You might have to respond to any prompts that SoC Designer displays.
2. Make any changes to the model object parameters, if required.
3. Change focus to the Select SoC Designer Session dialog box.
4. Click **OK** to save the configuration.

### See also

- the following in the *RealView Debugger Essentials Guide*:
  - *Supported Debug Interfaces* on page 1-4
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
- Carbon SoC Designer Plus, <http://carbondesignsystems.com/Products/SoCDesigner.aspx>.



## 2.10.2 Procedure for customizing a SoC Designer Debug Interface configuration (SoC Designer running)

If you have made changes to a SoC Designer model, and a RealView Debugger Debug Interface configuration currently exists for that model, then you can customize the configuration to reflect the changes you have made.

To customize a SoC Designer Debug Interface configuration when Carbon SoC Designer Simulator is currently running:

1. Start Carbon SoC Designer Simulator.
2. Load the modified model into Carbon SoC Designer Simulator.
3. Customize the associated SoC Designer Debug Interface configuration with the currently configured model:
  - a. Select the **Configuration** grouping from the Grouped By list.
  - b. Expand the SoC Designer Debug Interface to see the existing Debug Configurations.
  - c. Right-click on the Debug Configuration to be updated to display the context menu.
  - d. Select **Configure...** from the context menu to display the Select SoC Designer Session dialog box. The next step depends on the model that is loaded in Carbon SoC Designer Simulator:
    - If the model loaded in the simulator is the same as the model in this Debug Configuration, then continue at step 4.
    - If the model loaded in the simulator is different from the model configured in this Debug Configuration, the Saved Session SocDesigner dialog box is displayed.
  - e. Do one of the following:
    - Click **Yes** on the prompt dialog box to launch another Carbon SoC Designer Simulator with the model configured in the current Debug Configuration. You might have to respond to any prompts that SoC Designer displays.
    - Click **No** to change the Debug Configuration to the model loaded in the current Carbon SoC Designer Simulator session. Click **Scan**. The model is displayed in the Running SoC Designer sessions list.
4. Click **OK** to save the configuration.

### See also

- the following in the *RealView Debugger Essentials Guide*:
  - *Supported Debug Interfaces* on page 1-4
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
- Carbon SoC Designer Plus, <http://carbondesignsystems.com/Products/SoCDesigner.aspx>.

# Chapter 3

## Customizing a Debug Configuration

This chapter describes in detail how to customize a Debug Configuration to define a specific debugging environment. It includes:

- *About customizing a Debug Configuration* on page 3-2
- *Viewing the Connection Properties* on page 3-5
- *Changing connection settings* on page 3-10
- *Loading a different board file* on page 3-15
- *Hiding a Debug Configuration* on page 3-18
- *Specifying connect and disconnect mode* on page 3-19
- *Creating a target-specific Advanced \_Information group* on page 3-23
- *Configuring vector catch* on page 3-24
- *Configuring Semihosting* on page 3-29
- *Configuring the CLI commands for hardware cross-triggering* on page 3-33
- *Configuring a connection sequence for multiple targets* on page 3-36
- *Running CLI commands automatically on connection* on page 3-41
- *Configuring RealMonitor for connections through DSTREAM or RealView ICE* on page 3-43
- *Flash programming* on page 3-50
- *Using the Thumb-2EE helper macro* on page 3-51
- *Restoring the default connections and configurations* on page 3-55
- *Preparing Debug Configurations for distribution* on page 3-56
- *Distributing Debug Configurations to other machines and users* on page 3-58
- *Example of setting up an Integrator board and processor core module* on page 3-60
- *Troubleshooting Debug Configurations* on page 3-65.

### 3.1 About customizing a Debug Configuration

A Debug Configuration is defined by a CONNECTION= group within the board file (.brd) in your ARM® RealView® Debugger home directory. Each Debug Configuration in the Connect to Target window has a corresponding entry in the board file. For example, CONNECTION=RealView-ICE, corresponds to the RVI Debug Configuration for the RealView ICE Debug Interface.

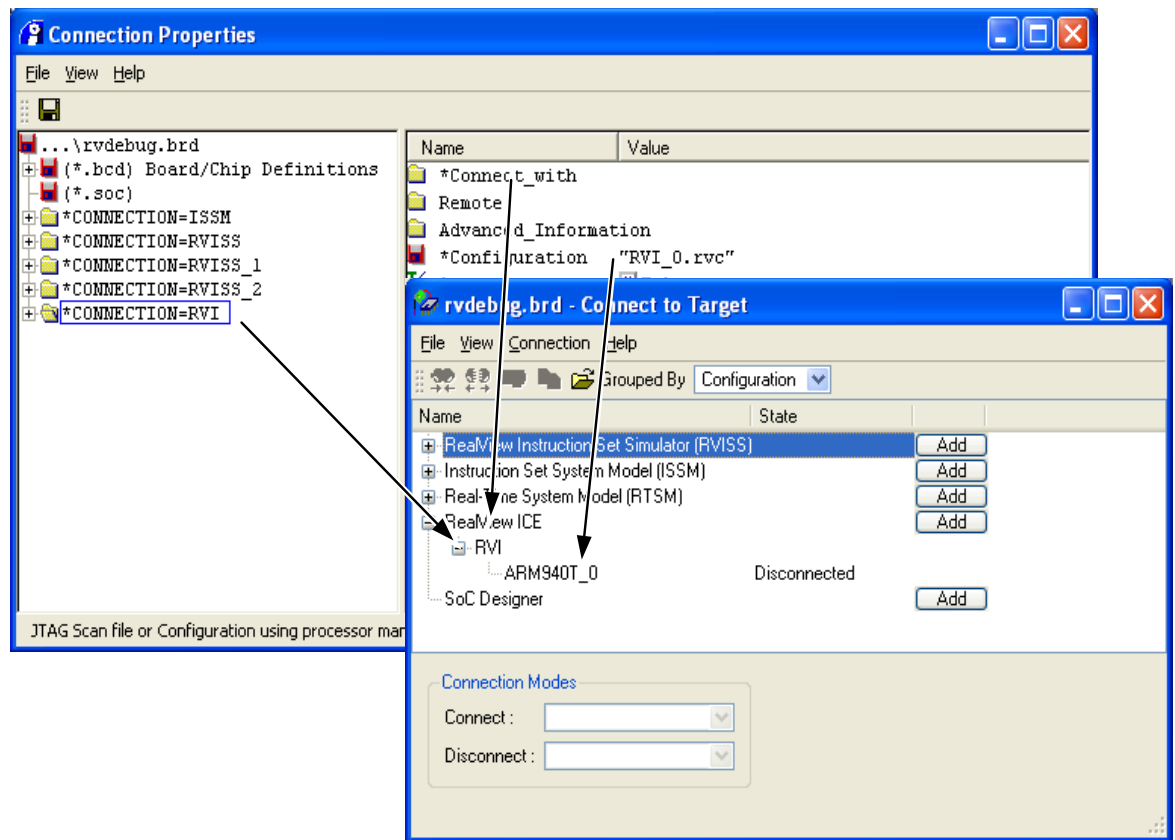
To view and configure the connection properties, you use the Connection Properties window.

See also:

- *Relationship between connection properties and Debug Configurations*
- *Managing configuration settings on page 3-3*
- *Using the examples on page 3-3.*

#### 3.1.1 Relationship between connection properties and Debug Configurations

Debug Configuration entries that are enabled form the basis of the information displayed in the Connect to Target window. Figure 3-1 shows the relationship between Connection Properties and the Connect to Target window.



**Figure 3-1 Relationship between Connection Properties and the Connect to Target window**

In Figure 3-1, a CONNECTION=name group in left pane of the Connection Properties window defines a Debug Configuration. For a Debug Configuration:

- The Configuration setting in the right pane identifies the Debug Interface configuration file that is used by the Debug Configuration. The Debug Interface configuration file identifies the target information for the Debug Configuration.

- The Manufacturer setting in the Connect\_with group identifies the Debug Interface to which the Debug Configuration belongs.

For the example shown in Figure 3-1 on page 3-2:

- \*CONNECTION=RealView-ICE defines the RealView-ICE Debug Configuration. A Debug Configuration entry must have unique name.
- \*Configuration "rvi.rvc" specifies a RealView ICE Debug Interface configuration file that identifies the targets (that is ARM940T\_0).

In the \*Connect\_with group, the setting \*Manufacturer ARM-ARM-NW indicates that this Debug Configuration is for the RealView ICE Debug Interface.

#### See also

- *Debug Configuration Advanced\_Information settings reference* on page A-10.

### 3.1.2 Managing configuration settings

You configure your debug target by amending Debug Configuration entries using the Connection Properties window. This enables you to specify connection behavior, target visibility, image loading parameters, and disconnect options.

RealView Debugger provides great flexibility in how to configure these settings so that you can control your debug target and any custom hardware that you are using. This means that some settings can be defined in the top-level board file so that they apply:

- to Debug Configurations, for example CONNECTION=RealView-ICE
- on a per-board (or per-chip) basis using groups in one or more linked BCD files, for example CHIP=KS32C50100 in the file Eval7T.bcd.

#### ———— Note ————

To avoid conflicts between settings when you reference multiple boards, follow the guidelines given in *Avoiding conflicts between linked board groups* on page 3-14.

#### See also

- *Avoiding conflicts between linked board groups* on page 3-14
- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals*.

### 3.1.3 Using the examples

The examples in the rest of this chapter, modify the board file stored in your RealView Debugger home directory. By default, this is called rvdebug.brd. Target configuration files might also be stored in this directory, for example .rvc files.

It is recommended that you back up your RealView Debugger home directory before starting the examples, so that you can restore your original configuration later.

In these examples:

- It is assumed that you are starting with the default board file as installed with the base product. Depending on the type of installation you choose, and your other ARM products, the contents of this file might be different from the one shown here. However, the methods described can be applied to any board file.

- Board file entries are created and renamed. The names used are for illustration only and you can change them as you require. However, it is recommended that you avoid duplicates.

---

**Note**

---

Before you configure a Debug Configuration, you must disconnect from all the targets in that configuration.

---

**See also**

- *Saving and restoring connection properties* on page 1-17
- *Restoring the default connections and configurations* on page 3-55
- *Troubleshooting Debug Configurations* on page 3-65.

## 3.2 Viewing the Connection Properties

RealView Debugger provides a Connection Properties dialog box and a Connection Properties window to enable you to examine, and change, Debug Configuration details stored in a board file. Settings in the Connection Properties can be applied to all target connections in a Debug Configuration or to a specific target connection.

See also:

- *Displaying the Connection Properties dialog box*
- *Displaying the Connection Properties window on page 3-6*
- *Identifying groups in the List of Entries pane on page 3-6*
- *Identifying settings in the Settings Values pane on page 3-6*
- *Debug Configuration entries on page 3-7*
- *The Debug Configuration Advanced\_Information block on page 3-8.*

### 3.2.1 Displaying the Connection Properties dialog box

The Connection Properties dialog box contains the more commonly used Debug Configuration settings. It enables you to examine and edit these settings.

To display the Connection Properties dialog box:

1. Locate a Debug Configuration in the Connect to Target window.
2. Right-click on the Debug Configuration to display the context menu.
3. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed. Figure 3-2 shows an example:

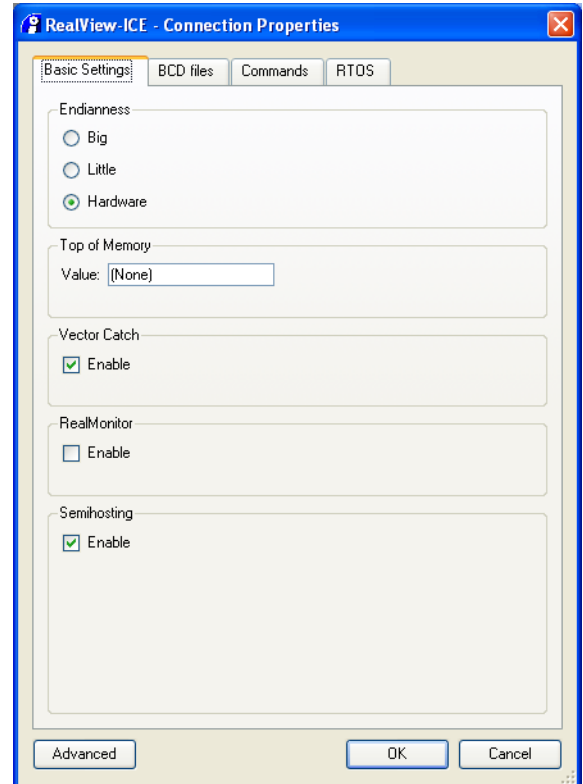


Figure 3-2 Connection Properties dialog box

### 3.2.2 Displaying the Connection Properties window

The Connection Properties window enables you to examine your current Debug Configuration settings and edit these settings to modify the debugging environment for that configuration.

To display the Connection Properties window, click the **Advanced** button on the Connection Properties dialog box. Figure 3-3 shows an example Connection Properties window.

#### ———— Note ————

An asterisk (\*) placed at the front of a group name shows that at least one setting in the group has changed from the default or that it was changed by RealView Debugger.

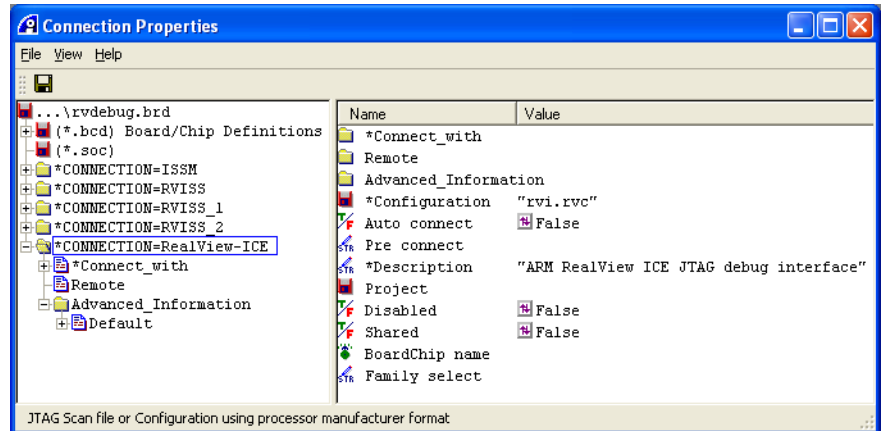





Figure 3-3 Connection Properties window

### 3.2.3 Identifying groups in the List of Entries pane

The left pane of the Connection Properties window, the List of Entries pane, shows configuration entries as a hierarchical tree with node controls.

Groups of settings are associated with an icon to explain their function:

-  **Red disk** This is a container disk file.  
For example, the (\*bcd) Board/Chip Definitions group lists all BCD files that RealView Debugger has found.
-  **Yellow folder** This is a parent group containing other groups (*rules pages*) and/or entries. For example, a Debug Configuration group.
-  **Rules page** A rules page is a container for settings values that you can change in the right pane. This icon only appears in the left pane.

#### See also

- *The RealView Debugger search path on page 1-17.*

### 3.2.4 Identifying settings in the Settings Values pane

If you click on an entry in the left pane, a blue box is drawn around it and the Description field is updated. At the same time, the right pane, the Settings Values pane, is updated to show the contents of the highlighted group.

Settings are associated with an icon to explain their function:



**Red disk**

This specifies a disk file.

For example, the Configuration setting specifies the Debug Interface configuration file used by the parent Debug Configuration.



**Yellow folder**

This is a parent group or rules page containing other groups (*rules pages*), settings, or both.



**String value**

This is a text string. If more than one value can be assigned to the setting, a new setting is created with the chosen value, and is colored blue. The original setting remains available for you to add other values if required.




**Numerical value**

This is a numerical value.



**True/False value**

This has a value that is either True or False. To change the value, either:

- click on the switch button  in the Value field
- right-click on the setting, and select True or False from the context menu.



**Preset selections value**

This enables you to select the value from a list that is defined by RealView Debugger. If more than one value can be assigned to the setting, a new setting is created with the chosen value, and is colored blue. The original setting remains available for you to add other values if required.

**See also**

- *Default configuration files* on page 1-7.

### 3.2.5 Debug Configuration entries

RealView Debugger sets up Debug Configuration entries in the board file based on targets it finds during installation. For example, if you have installed the DSTREAM and RealView ICE host software, RealView Debugger autodetects the appropriate .prc file. If you install the host software after installing RealView Debugger, the display list is automatically updated when you next start RealView Debugger.

The Connect to Target window uses the connection entries in the board file to create a tree of possible connections. Each connection entry in the board file references a device configuration file (such as .rvc). The device configuration file specifies the processors that can be reached using that connection, for example ARM940T\_0.

To see some connection entries you must install other ARM products, such as the DSTREAM and RealView ICE host software, because RealView Debugger does not include any connection software of its own. However, RealView Debugger does include connection interface components, or Debug Interfaces, to provide the interface to each target:

- the DSTREAM Debug Interface is used to connect to hardware target processors through DSTREAM.
- the RealView Instruction Set Simulator (RVISS) Debug Interface is used to connect to simulated ARM processors using *RealView Instruction Set Simulator* (RVISS)



- the Instruction Set System Model (ISSM) Debug Interface is used to connect to simulated ARM Cortex™ models
- the Model Library Debug Interface is used to connect to simulated targets in one or more CADI model library files
- the Model Process Debug Interface is used to connect to simulated targets in one or more processes running CADI models, such as Exported Virtual Systems
- the Real-Time System Model (RTSM) Debug Interface is used to connect to simulated RTSM targets
- the SoC Designer Debug Interface is used to connect to targets on simulated development platforms created with Carbon SoC Designer Plus
- the RealView ICE Debug Interface is used to connect to hardware target processors through RealView ICE.

You can add, copy, and rename Debug Configurations. Also, if you have created many Debug Configurations, you can hide a Debug Configuration without having to delete it.

#### See also

- *Hiding a Debug Configuration* on page 3-18
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
  - *Changing the name of a Debug Configuration* on page 3-17
  - *Copying an existing Debug Configuration* on page 3-18
  - *Chapter 3 Target Connection*.

### 3.2.6 The Debug Configuration Advanced\_Information block

A special group of settings is available in each Debug Configuration (CONNECTION group), called the Advanced\_Information block. Figure 3-3 on page 3-6 shows the position of an Advanced\_Information block in the Connection Properties window. When you create a Debug Configuration, a Default group is provided in this block. Any settings you configure in the Default group are applied to all the targets in your Debug Configuration. However, if you want settings to be applied to specific targets, then you must create target-specific groups.

You cannot rename the top-level Default group of the Advanced\_Information block. For a Debug Configuration, it is recommended that you create your own named groups, but do not delete the Default group. The Default group is then available for creating future configurations, and for situations where a named group does not match a target on the related Debug Configuration.

#### Naming an Advanced\_Information block group

RealView Debugger enables you to create multiple groups in the Advanced\_Information block. Each group has a unique name that relates to the target to which the settings are to apply. The name can be one of the following:

- a vendor, such as ARM
- a processor family, such as ARM9 or Cortex
- a partial processor name, such as ARM92
- a processor name, such as ARM940T
- a complete target connection name, such as ARM940T\_0.

## Matching of Advanced\_Information block group names and targets

When you connect to a target, RealView Debugger attempts to match the target name with a group name in the Advanced\_Information block. The match is checked in the following sequence:

1. Check the complete target connection name.  
The settings in a group called ARM926EJ-S\_0 are used only for an ARM926EJ-S processor that is at the first position on a DSTREAM or RealView ICE scan chain.
2. Check the target processor name.  
The settings in a group called ARM926EJ-S are used for an ARM926EJ-S processor at any position on a DSTREAM or RealView ICE scan chain. If there is more than one ARM926EJ-S processor, then the settings are used for all those processors.
3. Check the partial processor name.  
The settings in a group called ARM92 are used for all ARM92xx processors at any position on a DSTREAM or RealView ICE scan chain. For example, both ARM926EJ-S\_0 and ARM920T\_1 match.
4. Check the processor family name.  
The settings in a group called ARM9 are used for all the ARM9 family of processors at any position on a DSTREAM or RealView ICE scan chain. For example, both ARM926EJ-S\_0 and ARM966E-S\_1 match.
5. Check the vendor name.  
The settings in a group called ARM are used for any ARM processor at any position on a scan chain.
6. If no target matches are found, use the settings in the Default group.

### See also

- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities.*

### 3.3 Changing connection settings

RealView Debugger provides built-in default values for all settings in the Connection Properties window. The absence of an asterisk character next to a setting name indicates that the built-in default value is used, as shown in the Value column.

You can override the default value for a setting by providing a specific value in the Connection Properties window. If you provide a specific value for a setting, the name of the setting is prefixed with an asterisk character to indicate that the value has changed.

See also:

- *Changing entries containing user-information values*
- *Changing a specific value back to the default value* on page 3-11
- *Changing the order of settings that have multiple values* on page 3-12
- *Removing instances from multi-value settings* on page 3-13
- *Avoiding conflicts between linked board groups* on page 3-14.

#### 3.3.1 Changing entries containing user-information values

To customize an entry in the connection properties that contains user-information values:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView Instruction Set Simulator (RVISS).
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the Debug Configuration to be displayed to display the context menu. For example, right-click on RVISS.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Right-click on the Description entry in the right pane to display the context menu.
7. Select **Edit Value...** from the context menu.  
The text defined by this entry is selected, and in-place editing is enabled.
8. Enter a new description and press Enter.  
For example, enter **ARM7TDMI simulated processor**.
9. Select **Save and Close** from the **File** menu to close the Connection Properties window and save this change.

———— **Note** —————

If you close the window without saving your changes, a warning dialog box is displayed indicating that the contents have changed, and gives you the option of saving them.

10. Click the **OK** button to close the Connection Properties dialog box.

This example demonstrates the following:

- the Description setting has been changed in your user-specific copy of the board file, for example:  
C:\Documents and Settings\userID\Application Data\ARM\rvdebug\version\rvdebug.brd
- the default board file in your default setting directory is not changed:  
C:\Documents and Settings\userID\Local Settings\Application Data\ARM\rvdebug\version\shadowbase\etc\rvdebug.brd
- because the default value for the setting has been changed, an asterisk marks the setting and the Debug Configuration entry in the Connection Properties window.

### 3.3.2 Changing a specific value back to the default value

To change a specific value back to the default value:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView Instruction Set Simulator (RVISS).
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RVISS.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Locate the setting that you want to reset to the default value (the setting name is prefixed with an asterisk). For example:  
\*Description "ARM7TDMI simulated processor"
7. Right-click on the required setting in the right pane to display the context menu.
8. Select the appropriate option from the context menu:
  - **Reset to Default** for entries that have a default value in the default board file, such as Disabled. This sets the value of the setting to that defined in the default board file.
  - **Reset to Empty** for entries that have no value by default, such as Description. This clears the value for the entry.

The asterisk is also removed from the setting name.

For settings where new instances are created that are colored blue, such as the BoardChip\_name setting:

---

**Note**

---

Be aware that if you change a setting to a value that corresponds to the default value, and the setting name is still prefixed with an asterisk, then RealView Debugger does not treat the setting as a default setting. Therefore, the setting might conflict with the equivalent setting in another file.

---

9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
10. Click the **OK** button to close the Connection Properties dialog box.  
The original contents of the Connect to Target window are restored. In this example, the RVISS Debug Configuration is visible.

### 3.3.3 Changing the order of settings that have multiple values

Some settings, such as Pre\_connect, enable you to enter multiple values. Each value you enter creates a new instance below the main setting that is colored blue. The last value you enter is listed first. For some settings, such as BoardChip\_Name, this order is not important. However, for settings such as the Pre\_connect setting the order is important. Therefore, if you discover that you have entered the values in the wrong order you might want to re-order the list.

To re-order the list of values for a setting that has multiple values:

1. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RVISS.
2. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
3. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
4. Locate the multiple-value setting that you want to reorder, such as the Pre\_connect setting.
5. Right-click on any instance of the setting to display the context menu.  
For example, right-click on an instance of the Pre\_connect setting.
6. Select **Manage List...** from the context menu to display the Settings: List Manager dialog box.
7. Select the check box for a group or setting that you want to move.
8. Click either **Move Up** or **Move Down** to move the selected group or setting to the required position.
9. Click **OK** to close the Settings: List Manager dialog box.  
The list is reordered (the Pre\_connect list in this example).
10. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
11. Click **OK** to close the Connection Properties dialog box.

#### See also

- *Identifying settings in the Settings Values pane* on page 3-6
- *Pre\_connect* on page A-16.

### 3.3.4 Removing instances from multi-value settings

Some settings, such as Pre\_connect, enable you to enter multiple values. Each value you enter creates a new instance below the main setting that is colored blue. You can remove a specific instance directly, or multiple instances using a Settings: List Manager dialog box.

#### Removing a specific instance

To remove a specific instance:

1. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RVISS.
2. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
3. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
4. Locate the multiple-value setting that you want to reorder, such as the Pre\_connect setting.
5. Right-click on the required setting instance to display the context menu.
6. Select **Delete** from the context menu to delete that instance.
7. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
8. Click **OK** to close the Connection Properties dialog box.

#### Removing multiple instances

To remove multiple instances:

1. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RVISS.
2. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
3. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
4. Locate the multiple-value setting that you want to reorder, such as the Pre\_connect setting.
5. Right-click on any instance of the setting to display the context menu.
6. Select **Manage List...** from the context menu to display the Settings: List Manager dialog box.
7. Select the check box the instances that you want to move.
8. Click **Remove** to delete the selected instances.
9. Click **OK** to close the Settings: List Manager dialog box.
10. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

11. Click **OK** to close the Connection Properties dialog box.

**See also**

- *Identifying settings in the Settings Values pane on page 3-6.*

### 3.3.5 Avoiding conflicts between linked board groups

If a setting in the BCD file conflicts with a connection-level setting, then the connection-level setting is used. For these settings a warning message is displayed in the Output view, for example:

Warning: Semihosting has conflicting settings (0x01 instead of 0x00).

To avoid this situation, it is recommended that you only change settings in one place in the debugger, and leave all other settings with the same name with their default value. This simplifies changing the settings because it avoids having to search through the settings elsewhere in the debugger looking for conflicting values.

———— **Note** ————

It is recommended that BCD files are used only for configuring memory map related information.

—————

**See also**

- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals.*

## 3.4 Loading a different board file

When you configure your Debug Interface and connection details, you can make a copy of the default board file (rvdebug.brd) in your home directory, or any other location, to create different target connection views. If you do this, then you might want to load the copy of the board file to access the Debug Interface and Debug Configurations in that board file.

See also:

- *Manually loading a board file*
- *Forcing the load of a specific board file on startup (global configuration)* on page 3-16
- *Forcing the load of a specific board file on startup (workspace configuration)* on page 3-16.

### 3.4.1 Manually loading a board file

To manually load a board file:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Disconnect all your current target connections. To do this, enter the following CLI command in the **Cmd** tab of the Output view:  
`disconnect,all`  
All connections on all Debug Configurations are disconnected.
3. Select **Load Board File...** from the Connect to Target window **File** menu to display the Load Board File dialog box.
4. Locate the board file that you want to load.
5. Click **Open**.
6. Click **Yes** when prompted to load the chosen board file.

This replaces your current Debug Interface list and Debug Configurations with those in the chosen board file.

#### ———— **Note** ————

Your original board file is not modified. To restore your original Debug Interface list and Debug Configurations, load the default board file (rvdebug.brd) in your home directory.



### 3.4.2 Forcing the load of a specific board file on startup (global configuration)

You can force a specific board file to be loaded on startup. To do this:

1. Select **Options...** from the Code window **Tools** menu to display the global configuration Options window.
2. Select **DEBUGGER** in the left pane. The settings for this group are displayed in the right pane.
3. Right-click on **Board\_file** in the right pane to display the context menu.
4. Select **Edit as filename...** from the context menu to display the Enter New Filename dialog box.
5. Locate the required board file that you want to load on startup.
6. Click **Save** to store the location in the **Board\_file** setting.
7. Select **Save and Close** from the **File** menu to save the settings and close the window.
8. Exit and restart RealView Debugger for the change to take effect. The specified board file is used for all subsequent restarts of RealView Debugger.

#### See also

- the following in the *RealView Debugger User Guide*:  
— **DEBUGGER** on page A-2.

### 3.4.3 Forcing the load of a specific board file on startup (workspace configuration)

You can force a specific board file to be loaded on startup. To do this:

1. Disconnect all your current target connections. To do this, enter the following CLI command in the **Cmd** tab of the Output view:  
`disconnect,all`  
All connections on all Debug Configurations are disconnected.
2. Select **File** → **Workspace** → **Workspace Options...** from the Code window main menu to display the Workspace Options window.
3. Select **DEBUGGER** in the left pane. The settings for this group are displayed in the right pane.
4. Right-click on **Board\_file** in the right pane to display the context menu.
5. Select **Edit as filename...** from the context menu to display the Enter New Filename dialog box.
6. Locate the required board file that you want to load on startup.
7. Click **Save** to store the location in the **Board\_file** setting.
8. Select **Save and Close** from the **File** menu to save the settings and close the window.  
The change takes immediate effect. The specified board file is loaded and the Connect to Target window lists the Debug Interfaces and Debug Configurations in the chosen board file.  
The specified board file is used for all subsequent restarts of RealView Debugger.

**See also**

- the following in the *RealView Debugger User Guide*:
  - *DEBUGGER* on page A-2.

### 3.5 Hiding a Debug Configuration

By default, the Connect to Target window lists all Debug Configurations that have been created for each Debug Interface installed on your workstation. That is, all Debug Configurations are enabled by default. However, you can hide a Debug Configuration by disabling the corresponding entry in the connection properties.

To disable a Debug Configuration:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView Instruction Set Simulator (RVISS).
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the Debug Configuration to be displayed to display the context menu. For example, right-click on RVISS.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
 The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.  
 Enabled entries in the left pane are displayed in regular type, and those that are currently disabled are shown as gray text.
6. Left-click on the value for the Disabled setting in the right pane.
7. The value changes to **True**.
8. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.  
 In the Connect to Target window the Debug Configuration RVISS is no longer visible.
9. Click the **OK** button to close the Connection Properties dialog box.

See also:

- *Disabled* on page A-8.

## 3.6 Specifying connect and disconnect mode

If you want to specify how RealView Debugger connects to or disconnects from a target processor, you must configure this at the CONNECTION level. These definitions are contained in the Advanced\_Information group for the target processor.

---

### Note

---

The connect or disconnect mode that is used depends on the target hardware, the Debug Interface, and the associated interface software that manages the connection. For example, if you are using DSTREAM or RealView ICE, the unit configuration can specify the disconnect mode. Therefore, the unit configuration might override any settings that you specify in your board file.

---

See also:

- *Configuring connect mode*
- *Configuring disconnect mode* on page 3-20.

### 3.6.1 Configuring connect mode

The difference between configuring the connect mode for a connection through DSTREAM or RealView ICE is the Debug Interface that you use. The following procedure uses the RealView ICE Debug Interface as an example.

To configure connect mode for any hardware debug target:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RealView-ICE.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window. The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Expand the following entries in turn:
  - a. CONNECTION=RealView-ICE
  - b. Advanced\_Information
7. Select the Default group.

---

### Note

---

You can also configure a target-specific connect mode by creating target-specific Advanced\_Information groups.

---

8. Right-click on the `Connect_mode` setting in the right pane to display the context menu.  
The connection options on the context menu are fixed, and so might include options that are not supported by your Debug Interface. If you specify such an option, the debugger prompts you to select an appropriate connection mode when you try to connect.
9. Select the required disconnect mode (see *Connect\_mode* on page A-17). For example, select **no\_reset\_and\_no\_stop** so that the target processor is not reset when you connect. The running state of the target is unchanged.
10. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
11. Click the **OK** button to close the Connection Properties dialog box.

### Considerations for DSTREAM or RealView ICE

For connections through DSTREAM or RealView ICE, what happens when a processor comes out of reset depends on the combination of:

- the RealView Debugger connection mode
- the Default Post Reset State setting in the RVConfig utility.

Table 3-1 summarizes this action.

**Table 3-1 Action performed after reset**

Connection mode	Action performed after reset
Reset and Stop	Stops the target.
Reset and No Stop	Target state determined by the RVConfig Default Post Reset State setting.
No Reset and Stop	Specified by the RealView Debugger <code>connect_mode</code> setting or <b>Connect Mode</b> on the Connect to Target window.
No Reset and No Stop	Specified by the RealView Debugger <code>connect_mode</code> setting or <b>Connect Mode</b> on the Connect to Target window.

### See also

- *About customizing a DSTREAM or RealView ICE Debug Interface configuration* on page 2-3 for details of the Default Post Reset State setting
- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Connect\_mode* on page A-17
- the following in the *RealView Debugger User Guide*:
  - *Connecting to a target using different modes* on page 3-43
  - *Chapter 3 Target Connection*
  - *Appendix A Workspace Settings Reference*.

## 3.6.2 Configuring disconnect mode

The difference between configuring the disconnect mode for a connection through DSTREAM or RealView ICE is the Debug Interface that you use. The following procedure uses the RealView ICE Debug Interface as an example.

To configure disconnect mode for any hardware debug target:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RealView-ICE.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window. The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Expand the following entries in turn:
  - a. CONNECTION=RealView-ICE
  - b. Advanced\_Information
7. Select the Default group.

---

**Note**

---

You can also configure a target-specific connect mode by creating target-specific Advanced\_Information groups.

---

8. Right-click on the Disconnect\_mode setting in the right pane, to display the context menu. The disconnection options on the context menu are fixed, and so might include options that are not supported by your Debug Interface. If you specify such an option, the debugger displays a warning when you try to disconnect.
9. Select the required disconnect mode. For this example, select **prompt** so that RealView Debugger displays a prompt when you disconnect. This enables you to choose what disconnect mode to use at the time you disconnect.
10. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
11. Click the **OK** button to close the Connection Properties dialog box.
12. To test your changes:
  - a. Connect to your target and load an image, for example dhrystone.axf.
  - b. Run your image.
  - c. Disconnect from your target to view the Disconnect Mode prompt.

---

**Note**

---

All vector catch breakpoints are removed from the target on disconnection, irrespective of the disconnect mode.

---

**See also**

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Disconnect\_mode* on page A-17
- the following in the *RealView Debugger User Guide*:
  - *Disconnecting from a target using different modes* on page 3-54.

### 3.7 Creating a target-specific Advanced\_Information group

If your development platform has multiple targets, then you might want to specify different connection attributes for each target. You do this by creating target-specific groups in the Advanced\_Information block of the Debug Configuration. For example, you might want to perform OS-aware debugging on one target only.

The development platform used as the example in the following procedure contains an:

- ARM Integrator®/AP development board
- ARM920T processor at the first position on the scan chain
- ARM7TDMI® processor at the second position on the scan chain.

A DSTREAM or RealView ICE Debug Interface unit is used to connect to the development platform. Also, semihosting is to be enabled on the ARM920T only. The Debug Configuration defining the debugging environment is called AP-920T-7TDMI.

To create target-specific Advanced\_Information groups:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For this example, right-click on AP-920T-7TDMI.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Create a target-specific Advanced\_Information block for the target. For example, create a block called **ARM920T\_0**.

———— **Note** ————

Because the two processors used in the example are from two different families, you can set the group name to ARM9 instead. However, if you do this, and later add another processor from the ARM9 family to your development platform, the settings in the ARM9 group are used when you connect to both ARM9 processors.

7. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
8. Click the **OK** button to close the Connection Properties dialog box.

See also:

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Matching of Advanced\_Information block group names and targets* on page 3-9
- *Debug Configuration Advanced\_Information settings reference* on page A-10.



## 3.8 Configuring vector catch

Vector catch is a mechanism used to trap processor exceptions. This feature is typically used in the early stages of development to trap processor exceptions before the appropriate handlers are installed. You select the vectors to trap by setting the `vector_catch` value to `True`, and configuring the required exceptions in the Vectors group.

You can set the vector catch settings from various places in the debugger, including the Connection Properties window. However, see *Avoiding conflicts between linked board groups* on page 3-14 for the implications of changing settings in more than one place.

---

### Note

---

To make sure that the correct vector catch setting is used, you must set the vector catch only at the CONNECTION level.

---

See also:

- *Default settings for processor exceptions*
- *Setting vector catch for all targets* on page 3-25
- *Setting vector catch for individual targets* on page 3-26
- *Temporarily overriding the vector catch for an existing connection* on page 3-27
- *Considerations when setting SVC vector catch (hardware targets)* on page 3-27
- the following in the *RealView Debugger User Guide*:
  - *Customizing a Debug Configuration* on page 3-20.

### 3.8.1 Default settings for processor exceptions

Table 3-2 lists the default settings for the exceptions in the Connection Properties.

**Table 3-2 Default settings for Processor Exceptions**

Exception	Default	Comment
Reset	True	Catch Reset exceptions.
Undefined	True	Catch Undefined/Illegal Instructions.
SVC	False	Set to True to catch <i>SuperVisor Call</i> (SVC) interrupts. The SVC exception can also be trapped by the debugger to enable standard semihosting.
<hr/> <p style="text-align: center;"><b>Note</b></p> <p>For non ARMv7-M processors with the semihosting vector set to the default (0x8), you cannot enable the SVC vector catch if semihosting is enabled.</p> <hr/>		
Prefetch Abort	True	Catch Prefetch abort (instruction fetch memory fault) exceptions.
Data Abort	True	Catch Data abort (data access memory fault) exceptions.
Address	True	Catch Address exceptions. Used only by the obsolete 26-bit ARM processor architectures.

**Table 3-2 Default settings for Processor Exceptions (continued)**

Exception	Default	Comment
IRQ	False	Set to True to catch interrupt requests.
FIQ	False	Set to True to catch fast interrupt requests.
Error	True	Catch Error exceptions. Supported only for RVISS targets.

### 3.8.2 Setting vector catch for all targets

To set the state of individual vectors for all targets in a Debug Configuration:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RealView-ICE.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. To set the state of the vectors:
  - a. Expand the following entries:
    - CONNECTION=*name* (*name* is RealView-ICE in this example)
    - Advanced\_Information
    - Default
    - ARM\_config
  - b. Select Vectors in the left pane.
  - c. Set each vector in the right pane to the required state (see Table 3-2 on page 3-24).
7. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
8. Click the **OK** button to close the Connection Properties dialog box.

#### **Note**

When you connect to a target in your Debug Configuration, the state of each vector is applied to that target.

#### **See also**

- *The Debug Configuration Advanced\_Information block* on page 3-8

- *Avoiding conflicts between linked board groups* on page 3-14
- *Creating a target-specific Advanced\_Information group* on page 3-23
- *Default settings for processor exceptions* on page 3-24
- *ARM\_config settings for a Debug Configuration* on page A-11.

### 3.8.3 Setting vector catch for individual targets

To set the state of individual vectors for individual targets in a Debug Configuration:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window is displayed.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RealView-ICE.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window. The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Create a target-specific Advanced\_Information block for the target. For example, create a block called **ARM940T**.
7. For each individual target group in the Advanced\_Information block:
  - a. Expand the following entries:
    - *group\_name* (for example ARM940T)
    - *ARM\_config*
  - b. Click Vectors in the left pane.
  - c. Set each vector in the right pane to the required state.
8. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
9. Click the **OK** button to close the Connection Properties dialog box.

#### ———— Note ————

When you connect to a target in your Debug Configuration, the state of each vector is applied only to the target that matches the Advanced\_Information block name (an ARM940T™ processor in this example).

#### See also

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Avoiding conflicts between linked board groups* on page 3-14
- *Creating a target-specific Advanced\_Information group* on page 3-23
- *Default settings for processor exceptions* on page 3-24

- *ARM\_config settings for a Debug Configuration* on page A-11.

### 3.8.4 Temporarily overriding the vector catch for an existing connection

You can temporarily override the value of `vector_catch` for an existing connection as follows:

1. Make sure that the details for the required connection are shown in the Code window.

**————— Note —————**

If you have multiple connections, you might have to change the current connection.

2. Select **Processor Exceptions...** from the Code window **Debug** menu to display the Processor Exceptions dialog box.
3. Select the processor exceptions that you want enabled.
4. Click **OK**.

**————— Note —————**

Although setting vector catches in this way overrides those set using the other methods, the settings are not retained after you disconnect from the target. To make permanent changes to settings, use one of the other methods described in this section.

#### See also

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Avoiding conflicts between linked board groups* on page 3-14
- *Creating a target-specific Advanced\_Information group* on page 3-23
- *Default settings for processor exceptions* on page 3-24
- *ARM\_config settings for a Debug Configuration* on page A-11
- the following in the *RealView Debugger User Guide*:
  - *Changing the current target connection* on page 3-50.

### 3.8.5 Considerations when setting SVC vector catch (hardware targets)

For connections to non ARMv7-M hardware processors, semihosting uses the SVC vector by default. For these processors, you must not enable the SVC vector catch with semihosting enabled:

- If you enable both in the connection properties, the following warning is displayed when you connect:  
Warning: ARM\_config: cannot have vector catch of SVC and semi-hosting enabled.
- If you attempt to enable the SVC vector catch using the `BGLOBAL` command or the Processor Exceptions dialog box with semihosting enabled, the following error is displayed:  
Error V2801C (Vehicle): vector-catch-svc not available. This resource may be in use for semihosting.  
The SVC vector catch is not enabled in this case.

**See also**

- *Configuring Semihosting* on page 3-29
- the following in the *RealView Debugger User Guide*:
  - *Specifying processor exceptions (global breakpoints)* on page 11-65.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12 for details of the BGLOBAL command.

## 3.9 Configuring Semihosting

Semihosting enables your applications to communicate with the host workstation. In this way you can view messages from your application, and respond to any prompts.

See also:

- *Configuring semihosting for all targets*
- *Configuring semihosting for individual targets* on page 3-30
- *Considerations when configuring semihosting* on page 3-31
- the following in the *RealView Debugger User Guide*:
  - *Customizing a Debug Configuration* on page 3-20.

### 3.9.1 Configuring semihosting for all targets

To configure semihosting for all target in a Debug Configuration:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RealView-ICE.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window. The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Expand the following entries for the connection:
  - Advanced\_Information
  - Default
  - ARM\_config
7. Click Semihosting in the left pane.
8. Set Enabled in the right pane to the required value (True is the default).
9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
10. Click the **OK** button to close the Connection Properties dialog box.

---

**Note**

When you connect to any target in your Debug Configuration, the semihosting setting is applied that target.

---

**See also**

- *The Debug Configuration Advanced\_Information block on page 3-8*
- *Avoiding conflicts between linked board groups on page 3-14*
- *ARM\_config settings for a Debug Configuration on page A-11.*

**3.9.2 Configuring semihosting for individual targets**

To configure semihosting for individual targets in a Debug Configuration:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RealView-ICE.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Create a target-specific Advanced\_Information block for the target. For example, create a block called **ARM940T**.
7. For each individual target group in the Advanced\_Information block:
  - a. Expand the following entries:
    - *group\_name* (for example ARM940T)
    - ARM\_config
    - Semihosting
  - b. Set Enabled in the right pane to the required value (True is the default).
8. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
9. Click the **OK** button to close the Connection Properties dialog box.

**Note**

When you connect to a target in your Debug Configuration, the semihosting setting is applied only to the target that matches the Advanced\_Information block name (an ARM940T processor in this example).

**See also**

- *The Debug Configuration Advanced\_Information block on page 3-8*
- *Avoiding conflicts between linked board groups on page 3-14*
- *Creating a target-specific Advanced\_Information group on page 3-23*

- *Debug Configuration Advanced Information settings reference* on page A-10
- *ARM\_config settings for a Debug Configuration* on page A-11
- the following in the *RealView Debugger User Guide*:
  - *Changing the current target connection* on page 3-50.

### 3.9.3 Considerations when configuring semihosting

Be aware of the following when configuring semihosting:

- For connections to non ARMv7-M hardware processors, semihosting uses the SVC vector by default. For these processors, you must not enable semihosting with the SVC vector catch enabled:
  - If you enable both in the connection properties, the following warning is displayed when you connect:  
Warning: ARM\_config: cannot have vector catch of SVC and semi-hosting enabled.
  - If you attempt to enable semihosting with the SVC vector catch enabled, the following error is displayed:  
Error V2801C (Vehicle): vector-catch-svc not available. This resource may be in use for semihosting.  
Semihosting is not enabled in this case.

- You can configure semihosting using different methods, depending on the Debug Interface:

#### DSTREAM

After connecting to a target through DSTREAM, you can configure semihosting for that target using the **Debug** tab in the Registers view. This temporarily overrides the setting in the Connection Properties of the Debug Configuration.

#### Instruction Set System Model (ISSM)

Before connecting to an ISSM target, you can configure semihosting for that target using the Model Configuration Utility dialog box. The ISSM model configuration setting overrides the setting in the Connection Properties of the Debug Configuration.

#### Real-Time System Model (RTSM)

Before connecting to an RTSM target, you can configure semihosting for that target using the Model Configuration Utility dialog box. The RTSM configuration setting overrides the setting in the Connection Properties of the Debug Configuration.

#### RealView ICE

After connecting to a target through RealView ICE, you can configure semihosting for that target using the **Debug** tab in the Registers view. This temporarily overrides the setting in the Connection Properties of the Debug Configuration.

#### RealView Instruction Set Simulator (RVISS)

After connecting to an RVISS target, you can configure semihosting for that target using the **Semihost** tab in the Registers view. This temporarily overrides the setting in the Connection Properties of the Debug Configuration.



## SoC Designer

Before connecting to an SoC Designer target, you can configure semihosting for that target in the SoC Designer Component Parameters dialog box.

The SoC Designer model configuration setting overrides the setting in the Connection Properties of the Debug Configuration.

**See also**

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Avoiding conflicts between linked board groups* on page 3-14
- *Creating a target-specific Advanced\_Information group* on page 3-23
- *Configuring vector catch* on page 3-24
- *Debug Configuration Advanced\_Information settings reference* on page A-10
- *ARM\_config settings for a Debug Configuration* on page A-11
- *Chapter 2 Customizing a Debug Interface configuration*
- *Appendix B ISSM Configuration Reference*
- the following in the *RealView Debugger User Guide*:
  - *Specifying processor exceptions (global breakpoints)* on page 11-65
  - *Enabling or disabling semihosting (hardware targets)* on page 14-10
  - *Enabling or disabling semihosting (RVISS)* on page 14-11.
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*
- Carbon SoC Designer Plus, <http://carbondesignsystems.com/Products/SoCDesigner.aspx>
- *RealView Development Suite Real-Time System Model User Guide*.

### 3.10 Configuring the CLI commands for hardware cross-triggering

To set up hardware cross-triggering on your target, you must specify the appropriate CLI commands to use to enable and disable the In and Out triggers. Because these are target-specific, it is suggested that you set these up in a target-specific BCD file.

---

**Note**

---

The following procedure uses the ARM966E-S as an example, with the G\_CM\_DBGXTRIG register Start address specified as an absolute value. Alternatively, you can examine the equivalent definitions in the CM966ES.bcd file. In this file, the G\_CM\_DBGXTRIG register Start address is specified relative to a memory map block.

---

To configure the CLI commands to use for hardware cross-triggering:

1. Create a BCD file to use as a template.  
The rest of this procedure assumes that you have created the `template.bcd` file as described in *Creating a BCD file to use as a template* on page 4-13.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RealView-ICE.
4. Select **Connection Properties...** from the **Target** menu to display the Connection Properties dialog box.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Expand the (\*.bcd) Board/Chip Definitions group in the left pane. A list of existing BCD files is displayed.
7. Right-click on the `...\template.bcd` entry.
8. Select **Save As...** from the context menu to display the Enter New Name dialog box. The location displayed in this dialog box is the directory you last selected in the dialog box.
9. Locate your RealView Debugger home directory.  
By default, RealView Debugger searches for \*.bcd files in the current working directory, then in your home directory, and then in your default settings directory:  
C:\Documents and Settings\userID\Local Settings\Application Data\ARM\rvdebug\version\shadowbase\etc
10. Enter the new filename. You must use the .bcd file extension when saving the file.  
For this example, enter **ARM966ES.bcd**.
11. Click **Save**.

The New Name dialog box closes and the new name is displayed in the \*.bcd list. Although the new BCD entry replaces the BCD entry you used to make the copy, the original file still exists. To restore the complete list of BCD files:

- a. Select **Save Changes** from the **File** menu to save the changes.
  - b. Select **Refresh** from the **File** menu. This restores the list of all current BCD files, including the new file you have created.
  - c. Expand the group (\*.bcd) Board/Chip Definitions to display the current list of BCD files. The list includes the new BCD file, that is ARM966ES.bcd.
12. Rename the BOARD= group as appropriate:
    - a. Expand the entry for the new BCD file, ... \ARM966ES.bcd in this example.
    - b. Right-click on the BOARD=ARM\_TEMPLATE group in the left pane to display the context menu.
    - c. Select **Rename** from the context menu to display the Group/Type Name Selector dialog box.
    - d. Enter an appropriate name, for example **ARM966ES-1XTRIG**.
    - e. Click **OK** to close the dialog box.
  13. Create a new target-specific Advanced\_Information group. In the left pane:
    - a. Expand the BOARD=ARM966ES-1XTRIG group.
    - b. Expand the Advanced\_Information group.
    - c. Right-click on Advanced\_Information to display the context menu.
    - d. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default inserted.
    - e. Enter a name that is appropriate to the target for which this BCD file is to be used. For this example, enter **ARM966**.
    - f. Click **Create** to close the dialog box.
  14. Delete the Default group:
    - a. Right-click on the Default group in the left pane to display the context menu.
    - b. Select **Delete** from the context menu. The Default group is deleted.
  15. Locate the settings for cross-triggering. In the left pane:
    - a. Expand the ARM966 group you created.
    - b. Select the Cross\_trigger group. The cross-triggering settings are displayed in the right pane.
  16. Set up the CLI commands to enable and disable the In and Out triggers. Table 3-3 shows the commands to use for the ARM966E-S.

**Table 3-3 CLI commands to enable and disable the ARM966E-S In and Out triggers**

Setting	Command
Trig_in_ena	ce @B_DBGREQINEN  = 1
Trig_in_dis	ce @B_DBGREQINEN &= ~1
Trig_out_ena	ce @B_DBGACKOUTEN  = 1
Trig_out_dis	ce @B_DBGACKOUTEN &= ~1

In the commands shown in Table 3-3 on page 3-34:

- `ce` is the short form of the `CEXPRESSION` command.
- `B_DBGREQINEN` and `B_DBGACKOUTEN` are bit fields in a custom memory mapped register, `G_CM_DBGXTRIG`. To create the register and bit fields, follow the steps described in *Creating a custom memory mapped register* on page 4-37. Table 3-4 shows the attributes to specify for the ARM966E-S.

**Table 3-4 Bit fields use for cross-triggering on ARM966E-S**

Setting	Value
Register symbol name	<code>G_CM_DBGXTRIG</code>
Register Start	<code>0x10000070</code>
Register Length	4
Register Base	Absolute (the default)
In bit field symbol name	<code>B_DBGREQINEN</code>
In bit field Position	12
In bit field Size	4
In bit field Gui_name	<code>DBGREQINEN</code>
Out bit field symbol name	<code>B_DBGACKOUTEN</code>
Out bit field Position	8
Out bit field Size	4
Out bit field Gui_name	<code>DBGACKOUTEN</code>

17. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
18. Click the **OK** button to close the Connection Properties dialog box.

See also:

- *Creating a BCD file to use as a template* on page 4-13
- *Creating a custom memory mapped register* on page 4-37
- *Cross\_trigger* on page A-14
- the following in the *RealView Debugger User Guide*:
  - *Redefining the RealView Debugger directories* on page 2-9.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

### 3.11 Configuring a connection sequence for multiple targets

If you are debugging a development platform that contains multiple targets, then you might want one target to be connected before another is connected. Your development platform might contain:

- multiple processors
- one or more processors with *Embedded Trace Macrocell™* (ETM™) or *Embedded Trace Buffer™* (ETB™) trace devices
- one or more processors with CoreSight™ devices.

RealView Debugger enables you to specify the order in which these targets are connected.

---

**Note**

---

You can use this mechanism only when the targets are in the same Debug Configuration.

---

See also:

- *Setting a generic connection sequence for a Debug Configuration*
- *Specifying a target-specific connection sequence on page 3-38*

#### 3.11.1 Setting a generic connection sequence for a Debug Configuration

If you want to connect to all targets of a Debug Configuration in a specific sequence, then you can specify this sequence using the generic Pre\_connect setting for that configuration.

To specify the generic connection sequence for a Debug Configuration:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on RealView-ICE.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Add each target to the Pre\_connect sequence:
  - a. In the right pane, right-click on the black Pre\_connect setting to display the context menu.
  - b. Select **Edit Value...** from the context menu.
  - c. Enter the name of the target. A new Pre\_connect setting is added for the target, colored blue.

---

**Note**

---

The target name must be the full name that appears in the Connect to Target window for the Debug Configuration, for example, ARM940T\_1.

---

- d. Repeat these steps for each target to be added.

---

**Note**

---

You must add the targets in the reverse order that they are to connect. However, you can change the order if required.

---

7. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
8. Click the **OK** button to close the Connection Properties dialog box.

### Example connection sequence with multiple processors

For example, your development platform might have the processors ARM920T™, ARM940T, and ARM966EJ-S™. These might be shown as ARM920T\_0, ARM940T\_1, and ARM966EJ-S\_2 in the Connect to Target window. Therefore, if you want to make sure that the processors are connected in the order ARM966EJ-S, ARM920T, and ARM940T, then you must specify the Pre\_connect sequence for a Debug Configuration in the order ARM940T\_1, ARM920T\_0, and ARM966EJ-S\_2.

How RealView Debugger connects to the targets depends on the target you attempt to connect first. Table 3-5 summarizes the connect sequence for this example.

**Table 3-5 Connect sequence using the generic Pre\_connect setting**

Target you try to connect to	Target connect sequence
ARM920T	ARM966EJ-S ARM920T
ARM940T	ARM966EJ-S ARM920T ARM940T
ARM966EJ-S	ARM966EJ-S
Connect all operation	ARM966EJ-S ARM920T ARM940T

### See also

- *Changing the order of settings that have multiple values* on page 3-12
- *Pre\_connect* on page A-8
- the following in the *RealView Debugger User Guide*:
  - *Connecting to all targets for a specific Debug Configuration* on page 3-48.

### 3.11.2 Specifying a target-specific connection sequence

If a target depends on one or more targets being connected first, then you can use the `Pre_connect` setting in a target-specific `Advanced_Information` group.

To configure a target-specific connection sequence:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand `RealView ICE`.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on `RealView-ICE`.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Create a target-specific `Advanced_Information` block for the target. For example, create a block called **ARM940T**.
7. Click the group name you have created. For example, click `ARM940T`.
8. For each target that this target depends on, do the following:
  - a. In the right pane, right-click on the black `Pre_connect` setting to display the context menu.
  - b. Select **Edit Value...** from the context menu.
  - c. Enter the name of the target. A new `Pre_connect` setting is added for the target, colored blue.

———— **Note** ————

The target name must be the full name that appears in Connect to Target window for the Debug Configuration, for example, `ARM940T_1`.

  - d. Repeat these steps for each target to be added.

———— **Note** ————

You must add the targets in the reverse order that they are to connect. However, you can change the order if required.
9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
10. Click the **OK** button to close the Connection Properties dialog box.

### Example using target-specific Advanced\_Information blocks with multiple processors

Your development platform might contain the processors ARM920T, ARM940T, and ARM966EJ-S. These might be shown as ARM920T\_0, ARM940T\_1, and ARM966EJ-S\_2 in the Connect to Target window. You can create an Advanced\_Information block for each target, called ARM920, ARM940, and ARM966. To make sure that the targets are connected in the order ARM966EJ-S, ARM920T, and ARM940T, then set up the Advanced\_Information block for each target as shown in Table 3-6.

**Table 3-6 Pre\_connect settings in Advanced\_Information block**

Advanced_Information block	Pre_connect setting
ARM920	ARM966EJ-S_2
ARM940	ARM966EJ-S_2 ARM920T_0
ARM966	Empty, because this target does not rely on another target being connected.

### Connection sequence that occurs for each target

How RealView Debugger connects to the targets depends on the target you attempt to connect to first. Table 3-7 summarizes the connect sequence for this example.

**Table 3-7 Connection sequence using target-specific Pre\_connect setting**

Target you connect to first	Target connect sequence
ARM920T	ARM966EJ-S ARM920T
ARM940T	ARM966EJ-S ARM920T ARM940T
ARM966EJ-S	ARM966EJ-S
Connect all operation	ARM966EJ-S ARM920T ARM940T

### Example using target-specific Advanced\_Information blocks for a CoreSight system

Your CoreSight system might contain the following:

- a Cortex-A8 processor
- a CoreSight *Trace Port Interface Unit* (TPIU)
- a CoreSight ETM
- a CoreSight Trace Funnel
- two CoreSight *Cross Trigger Interfaces* (CTIs).



These might be shown as Cortex-A8\_0, CSTPIU\_0, CSETM\_0, CSTFunnel\_0, CSCTI\_0, and CSCTI\_1 in the Connect to Target window. You can create an Advanced\_Information block for the processor, called Cortex-A8. To make sure that the CoreSight devices are connected before the Cortex-A8 processor, then set up the Advanced\_Information block for each target as shown in Table 3-8.

**Table 3-8 Pre\_connect settings in Advanced\_Information block**

Advanced_Information block name	Pre_connect setting
Cortex-A8	CSTPIU_0 CSTFunnel_0 CSCTI_0 CSCTI_1 CSETM_0

This ensures that the connection to the Cortex-A8 processor is the last connection that is established, and is therefore the current connection.

### See also

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Changing the order of settings that have multiple values* on page 3-12
- *Creating a target-specific Advanced\_Information group* on page 3-23
- *Pre\_connect* on page A-8
- the following in the *RealView Debugger User Guide*:
  - *Connecting to all targets for a specific Debug Configuration* on page 3-48.

### 3.12 Running CLI commands automatically on connection

There are situations when you might want to perform various actions automatically when you connect to a target. For example, you might want load an image and stop execution at a specific point ready for debugging. To do this, you can define a sequence of CLI commands that are executed automatically after connecting to the target.

Before you define commands that run automatically on connection, be aware of the following:

- If you define commands in the **Default** group of an **Advanced\_Information** block, then the commands are run on each target in the Debug Configuration as you connect to that target. Usually, commands are target-specific, so you must create target-specific groups. The procedure described in the next section shows how to do this.
- Commands can be provided in a script file if required.
- If you want to specify user-defined macros, then you must put the macro definitions in a script file. The macro definitions must be defined before any commands that use them are executed.

To run CLI commands when you connect to a specific target:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand **RealView ICE**.
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on **RealView-ICE**.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window. The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Create a target-specific **Advanced\_Information** block for the target. For example, create a block called **ARM940T**.
7. Click the group name you have created. For example, click **ARM940T**.
8. Enter the commands to be run on the target:
  - a. In the right pane, right-click on the black **Commands** setting to display the context menu.
  - b. Select **Edit Value...** from the context menu.
  - c. Enter the required CLI command. A new **Commands** setting is added for the command, colored blue.
  - d. Repeat these steps for each command to be added.

---

**Note**

---

You must add the commands in the reverse order that they are to execute. However, you can change the order if required.

---

Alternatively, you can define the command sequence in a script file:

- a. Specify the CLI commands and any macro definitions in a command script.
- b. Add an **INCLUDE** command in the **Commands** setting to run the command script.  
For example, to run commands in the script `C:\rvd_scripts\myscript.inc`, add the command:  
`include 'C:\rvd_scripts\myscript.inc'`

9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
10. Click the **OK** button to close the Connection Properties dialog box.

See also:

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Changing the order of settings that have multiple values* on page 3-12
- *Creating a target-specific Advanced\_Information group* on page 3-23
- the following in the *RealView Debugger User Guide*:
  - *Customizing a Debug Configuration* on page 3-20
  - *Chapter 15 Debugging with Command Scripts*.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

### 3.13 Configuring RealMonitor for connections through DSTREAM or RealView ICE

This section describes how to configure RealView Debugger with DSTREAM or RealView ICE to run RealMonitor-integrated applications.

---

**Note**

---

RealMonitor is compatible with DSTREAM and with RealView ICE v1.1 and later.

---

See also:

- *Restrictions on using RealMonitor with DSTREAM or RealView ICE*
- *Rules for connecting DSTREAM or RealView ICE to a running target*
- *Basic procedure* on page 3-44
- *Customizing the Debug Configuration used for enabling RealMonitor* on page 3-46
- *Customizing the Debug Configuration used for debugging with RealMonitor* on page 3-47
- *Loading an image before using RealMonitor* on page 3-48
- *Working with RealMonitor* on page 3-48.

#### 3.13.1 Restrictions on using RealMonitor with DSTREAM or RealView ICE

RealView Debugger does not support the following RealMonitor functionality:

- Semihosting.
- Synchronizing the caches after memory writes.
- On-the-fly switching between run-mode (through RealMonitor) and stop-mode debug (halting the processor through DSTREAM or RealView ICE) is not possible. In particular, RealView Debugger makes no attempt to recover if the RealMonitor connection is lost during a debug session.

#### 3.13.2 Rules for connecting DSTREAM or RealView ICE to a running target

Because of floating signal levels on the JTAG connector, you must be aware of the following rules when using a DSTREAM or RealView ICE unit with running targets:

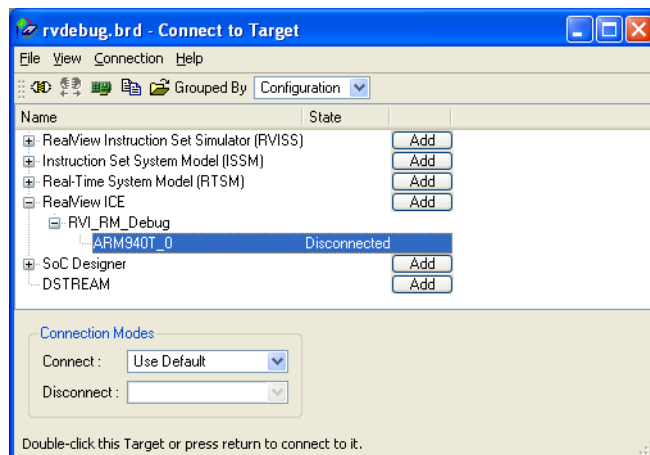
- Do not do any of the following actions, because they can cause the target to stop:
  - power up the DSTREAM or RealView ICE unit when it is connected to a running target
  - connect a running DSTREAM or RealView ICE unit to a running target
  - power down the DSTREAM or RealView ICE unit when it is connected to a running target.
- Configuring the DSTREAM or RealView ICE unit might stop the target you are connected to. In particular auto-configuring does this.
- Incorrect DSTREAM or RealView ICE settings can result in your target being stopped or staying in halt-mode debug. As a result, hardware breakpoints stop the target. RealView Debugger does not warn you if this happens.
- If your target has stopped when the DSTREAM or RealView ICE unit is configured or booted, reset your target before connecting to the RealMonitor connection with RealView Debugger.

RealMonitor support for RealView Debugger is not enabled if you connect to a halted target.

### 3.13.3 Basic procedure

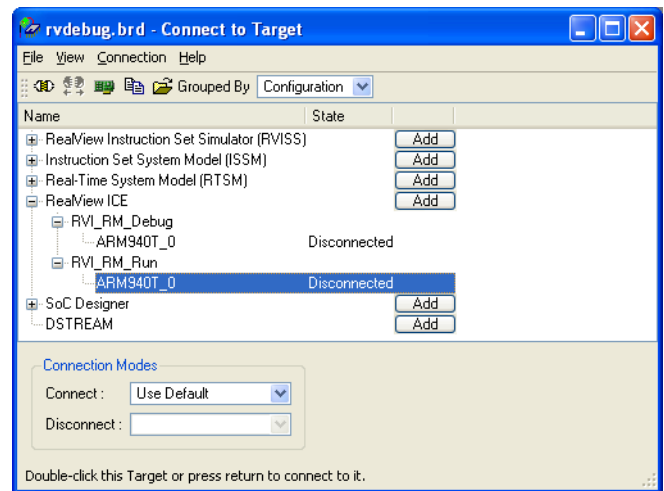
Debugging a RealMonitor-integrated application with DSTREAM or RealView ICE requires that you connect with the RMHost controller while the program is running. To debug a RealMonitor-integrated application with DSTREAM or RealView ICE you must perform the following sequence of actions:

1. Review the information in:
  - *Restrictions on using RealMonitor with DSTREAM or RealView ICE* on page 3-43
  - *Rules for connecting DSTREAM or RealView ICE to a running target* on page 3-43.
2. If you have an existing DSTREAM or RealView ICE Debug Configuration for your target, then continue at the next step. Otherwise, create a new Debug Configuration. This is to be the Debug Configuration you use for debugging your RealMonitor-integrated application.
3. Change the name of the Debug Configuration. For this example, change the name of the RealView ICE configuration to **RVI\_RM\_Debug**. Figure 3-4 shows an example:



**Figure 3-4 RealMonitor Debug Configuration for debugging**

4. Make a copy of the Debug Configuration. This new Debug Configuration is the one you use to enable RealMonitor on the target. For this example, change the name of the RealView ICE configuration to **RVI\_RM\_Run**. Figure 3-5 on page 3-45 shows an example:



**Figure 3-5 RealMonitor Debug Configuration for running target**

5. Customize the original (RVI\_RM\_Run) Debug Configuration as described in *Customizing the Debug Configuration used for enabling RealMonitor* on page 3-46.
6. Customize the new (RVI\_RM\_Debug) Debug Configuration as described in *Customizing the Debug Configuration used for debugging with RealMonitor* on page 3-47.
7. For the Debug Configuration used to enable RealMonitor:
  - a. Expand the Debug Configuration (RVI\_RM\_Run in this example).
  - b. Connect to the target.
  - c. Load the image.
  - d. Run the image.
  - e. Disconnect from the running target.
8. For the Debug Configuration used to debug your application:
  - a. Expand the Debug Configuration (RVI\_RM\_Debug in this example).
  - b. Connect to the running target.
  - c. Load only the symbols for the image.
  - d. Stop execution.  
This enables RealView Debugger to get the context so that you can begin debugging. RealView Debugger might prompt you for the location of the source file that contains the current context.
  - e. Restart execution.
9. You can debug your application as described in *Working with RealMonitor* on page 3-48.

### See also

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Changing the order of settings that have multiple values* on page 3-12
- *Creating a target-specific Advanced\_Information group* on page 3-23
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
  - *Changing the name of a Debug Configuration* on page 3-17

- *Copying an existing Debug Configuration* on page 3-18
- *Loading symbols only for an image* on page 4-16
- *Chapter 15 Debugging with Command Scripts.*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities.*

### 3.13.4 Customizing the Debug Configuration used for enabling RealMonitor

To configure the Debug Configuration used for enabling RealMonitor:

1. Right-click on the new Debug Configuration (RVI\_RM\_Run) to display the context menu.
2. Select **Configure...** from the context menu to display the RVConfig utility.
3. Select your debug hardware from the list of devices.
4. Click **Connect...** to connect to the hardware unit.
5. If your Debug Interface has targets that:
  - can be halted, then click **Auto Configure Scan Chain**
  - must not be halted, then add the devices manually.
6. Click **Advanced**, and change the settings as shown in Table 3-9. Leave all other settings unchanged.

**Table 3-9 DSTREAM or RealView ICE settings for RealMonitor**

Setting	Value
Reset Type	nTRST
Default Post Reset State	Running
TAP reset via State Transitions	deselected

#### Note

The intention here is to make sure that your development platform is not reset, and that the target is always running. Some targets might require different settings.

7. Select **Save** from the **File** menu to save the changes.
8. Select **Exit** from the **File** menu to close the RVConfig utility.

#### See also

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Changing the order of settings that have multiple values* on page 3-12
- *Creating a target-specific Advanced\_Information group* on page 3-23
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
  - *Changing the name of a Debug Configuration* on page 3-17
  - *Copying an existing Debug Configuration* on page 3-18
  - *Loading symbols only for an image* on page 4-16
  - *Chapter 15 Debugging with Command Scripts*

- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities.*

### 3.13.5 Customizing the Debug Configuration used for debugging with RealMonitor

To configure the Debug Configuration that is to be used for debugging a RealMonitor-integrated application, there are some settings for the connection that you must configure. You must configure these settings at the CONNECTION level.

To configure the mandatory settings for a RealMonitor connection:

1. Right-click on the original Debug Configuration (RVI\_RM\_Debug) to display the context menu.
2. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
3. Click the **Advanced** button to display the Connection Properties window.

The settings group for the Debug Configuration entry is selected, and the contents are displayed in the right pane.

Figure 3-6 shows an example Connection Properties window.

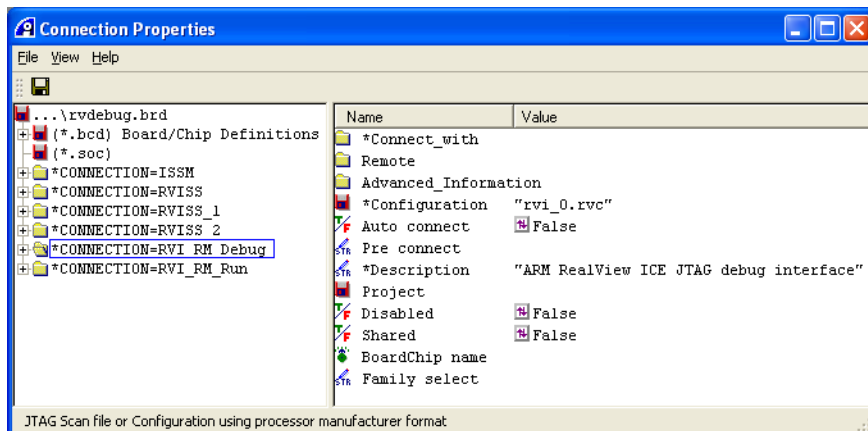


Figure 3-6 Debug Configuration settings

4. Expand the CONNECTION=RVI\_RM\_Debug entry.
5. Expand the Advanced\_Information entry.
6. Select Default in the left pane, and set the following in the right pane:
  - Connect\_mode to **no\_reset\_and\_no\_stop**
  - Disconnect\_mode to **as\_is\_without\_debug**.
  - Endianess to the endianness of your target.
7. Specify the monitor:
  - a. Expand the Default entry in the left pane.
  - b. Select Monitor in the left pane.
  - c. Set Type in the right pane to **RealMonitor**.
8. Disable vector catch:
  - a. Select ARM\_config in the left pane.
  - b. Set Vector\_catch in the right pane to **False**.



9. Disable semihosting:
  - a. Expand ARM\_config.
  - b. Select Semihosting in the left pane
  - c. Set Enabled in the right pane to **False**.
10. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
11. Click the **OK** button to close the Connection Properties dialog box.
12. You can connect to the target that is running RealMonitor. No additional configuration is required.

### 3.13.6 Loading an image before using RealMonitor

Typically, RealMonitor resides in Flash or ROM. However, if you have to upload your image, you must:

- Create a standard connection to upload the image to the target and set it running.
- Use the RVI\_RM\_Run connection you created in *Basic procedure* on page 3-44, and that you configured as described in *Customizing the Debug Configuration used for enabling RealMonitor* on page 3-46.

#### See also

- *Customizing the Debug Configuration used for enabling RealMonitor* on page 3-46
- *Changing the order of settings that have multiple values* on page 3-12
- *Creating a target-specific Advanced\_Information group* on page 3-23
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
  - *Chapter 4 Loading Images and binaries*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities*.

### 3.13.7 Working with RealMonitor

The main benefit of RealMonitor is that debugging functionality is available even when the foreground task is running. For example, you can insert breakpoints in the foreground application without the processor stopping the servicing of interrupts.

#### Periodic update of views

The Watch and Memory views have the following context menu options enabled when RealMonitor is enabled on the related target:

- **Timed Update When Running**
- **Timed Update Period...**

These options enable the view contents to be updated periodically with the foreground application running.

**See also**

- the following in the *RealView Debugger User Guide*:
  - Chapter 13 *Examining the Target Execution Environment*.
- *ARM RMHost User Guide*
- *ARM RMTarget Integration Guide*.

## 3.14 Flash programming

Programming the Flash with RealView Debugger involves:

- assigning the BCD file that describes the memory map and specifies a *Flash Method* (FME) file to the target connection.
- programming the image to the Flash device.

See also:

- *Requirements for programming Flash.*

### 3.14.1 Requirements for programming Flash

You can program Flash if you have:

- an ARM board that is supported by RealView Debugger
- a custom board that uses one of the Flash types supported by RealView Debugger
- a custom board that uses a Flash type not supported by RealView Debugger.

#### See also

- Chapter 6 *Programming Flash with RealView Debugger*
- the following in the *RealView Debugger User Guide*:
  - Chapter 6 *Writing Binaries to Flash.*

### 3.15 Using the Thumb-2EE helper macro

For targets that support Thumb<sup>®</sup>-2EE, such as Cortex-A8, you can use the Thumb-2EE helper macro defined in the thumb2ee.inc file to obtain addresses relative to the CP14\_THUMB2EE\_HANDLER\_BASE register.

See also:

- *Loading the Thumb-2EE helper macro for a single processor system*
- *Loading the Thumb-2EE helper macro for a multiprocessor system* on page 3-52
- *Using the Thumb-2EE helper macro with breakpoints and tracepoints* on page 3-53
- *Examining the definition of the handleraddr() macro* on page 3-54.

#### 3.15.1 Loading the Thumb-2EE helper macro for a single processor system

To load the Thumb-2EE helper macro in a single processor system:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand Instruction Set System Model (ISSM).
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the required Debug Configuration to display the context menu. For example, right-click on ISSM.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the ISSM Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Assign the thumb2ee.bcd file to the connection:
  - a. Right-click on the BoardChip\_name setting in the right pane to display the context menu.
  - b. Select **<More...>** from the context menu to display the List Selection dialog box. This lists the available board/chip definitions.
  - c. Select **Thumb2EE** from the list.
  - d. Click **OK** to close the List Selection dialog box.
  - e. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
  - f. Click the **OK** button to close the Connection Properties dialog box.
7. Connect to the required target. For example, connect to the ARM\_Cortex-A8 ISSM model.
8. Load the required image.  
RealView Debugger automatically loads the thumb2ee.inc file. This file defines the handleraddr(handlerIndex) macro, where handlerIndex is a Thumb-2EE handler number in the range 0 to 255.

**See also**

- *Summary of supplied BCD files* on page 1-19
- the following in the *RealView Debugger User Guide*:
  - Chapter 16 *Using Macros for Debugging*.

**3.15.2 Loading the Thumb-2EE helper macro for a multiprocessor system**

If your Thumb-2EE supported processor is part of a multiprocessor system, the method described in *Loading the Thumb-2EE helper macro for a single processor system* on page 3-51 might not be desirable. This is because RealView Debugger loads the macro for each processor to which an image is loaded.

To make sure the Thumb-2EE helper macro is loaded only for a Thumb-2EE supported processor in a multiprocessor system, do the following:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the Debug Interface containing the Debug Configuration of interest. For this example, expand Instruction Set System Model (ISSM).
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the Debug Configuration to display the context menu. For example, right-click on ISSM.
4. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
5. Click the **Advanced** button to display the Connection Properties window.  
The settings group for the ISSM Debug Configuration entry is selected, and the contents are displayed in the right pane.
6. Create an Advanced\_Information block for each:
  - a. Expand the Advanced\_Information group in the left pane.
  - b. Right-click on the Default group to display the context menu.
  - c. Select **Make New...** from the context menu to display the Enter Value dialog box.
  - d. Enter a processor name corresponding to the target processor. For example, enter **ARM\_Cortex-A8**.
  - e. Click **Create** to create the new group for the processor.
  - f. Repeat this procedure for other target processors as required.
7. For the Thumb-2EE supported target, select the target-specific group in the Advanced\_Information block. For example, select the ARM\_Cortex-A8 block.
8. Specify the commands to load the image and the macro:
  - a. Right-click on the black Commands setting in the right pane to display the context menu.
  - b. Select **Make New...** from the context menu.
  - c. Enter the following INCLUDE command to load the help macro:

```
include 'C:\Documents and Settings\userID\Local Settings\Application
Data\ARM\rvdebug\version\shadowbase\etc\thumb2ee.inc'
```

A new Commands setting is created, colored blue.

- d. Right-click on the main Commands setting in the right pane to display the context menu.
- e. Select **Make New...** from the context menu.
- f. Enter the required LOAD command to load your image, for example:  

```
load/pd/r
'install_directory\RVDS\Examples\...\cached_dhry\CortexA8dhry\CortexA8dhry.a
xf'
```

---

**Note**

You might have to build the example image in this directory first.

A new Commands setting is created, colored blue.

---

**Note**

The commands are added to the Commands list in reverse order. Therefore, the LOAD command runs before the INCLUDE command. You can re-order the commands if you have added them in the wrong order.

9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
10. Click the **OK** button to close the Connection Properties dialog box.

When you connect to a Thumb-2EE supported target, the image is automatically loaded first and then the helper macro is loaded. The macro is not loaded when you load an image to any other processor in the system.

### See also

- *Changing the order of settings that have multiple values* on page 3-12
- the following in the *RealView Debugger User Guide*:  
 — Chapter 16 *Using Macros for Debugging*.

### 3.15.3 Using the Thumb-2EE helper macro with breakpoints and tracepoints

You can use this macro when setting breakpoints and tracepoints, and for viewing disassembly and memory locations. For example, to set a breakpoint on Thumb-2EE handler number one, enter the command:

```
BREAKINSTRUCTION handleraddr(1)
```

---

**Note**

Be aware that if you enter a handler number that is not in the accepted range, the macro returns -1, and an error dialog is displayed. This means that if you are setting a breakpoint or tracepoint, RealView Debugger sets the breakpoint or tracepoint at address 0xFFFFFFFF.

### See also

- the following in the *RealView Debugger User Guide*:  
 — Chapter 16 *Using Macros for Debugging*.

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

### 3.15.4 Examining the definition of the handleraddr() macro

To see the definition of the handleraddr() macro after it is loaded, enter the CLI command:

```
SHOW handleraddr
```

#### See also

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

### 3.16 Restoring the default connections and configurations

If you have completed these examples and you want to return to the default installation settings, you can restore your board file.

To restore your target connections and configurations to the default installation:

1. Exit RealView Debugger.
2. Delete the `rvdebug.brd` file from your RealView Debugger home directory.

When you restart RealView Debugger it creates a new default `rvdebug.brd` configuration file for you.

Alternatively, start RealView Debugger with the `--cleanstart` argument, for example:

```
rvdebug --cleanstart
```

———— **Note** ————

This removes all files from your home directory, and populates it with new default files.

See also:

- *Syntax of the `rvdebug` command* on page 2-2.



### 3.17 Preparing Debug Configurations for distribution

You must prepare the RealView Debugger Debug Configurations that you want to distribute to other members of a development team.

If you intend to create Debug Configurations for use by multiple users, you are recommended to follow these steps:

1. Create a home directory that you want to use to hold the Debug Configuration files to be distributed, for example, C:\RVD\_home\DevTeam.

---

**Note**

The home directory can be on a remote machine that you access through a mapped drive, for example, H:\RVD\_home\DevTeam.

---

2. Start RealView Debugger using the following command (replace ... with the version and build number of your installation):

```
"C:\Program Files\ARM\RVD\Core\...\win_32-pentium\bin\rvdebug.exe"
--install="C:\Program Files\ARM\RVD\Core\...\win_32-pentium"
--home="C:\RVD_home\DevTeam"
```

RealView Debugger copies the default configuration files into the home directory. Table 3-10 lists the default configuration files.

**Table 3-10 Default configuration files in the home directory**

File	Description
armreg.sig	Required by the ARM simulators (ISSM, RVISS, and SoC Designer). Do not edit this file.
default.auc	Default configuration file for RVISS targets.
ISSM_0.smc	Target configuration file for ISSM Cortex-A8 target.
rvdebug.aws	The RealView Debugger user-specific workspace file.
rvdebug.brd	The RealView Debugger user-specific board file containing the Debug Configurations. This contains Debug Configurations for the following targets: <ul style="list-style-type: none"> <li>• ISSM Cortex-A8</li> <li>• RVISS ARM7TDMI</li> <li>• RVISS ARM926EJ-S</li> <li>• RVISS ARM1176JZF-S.</li> </ul>
rvdebug.ini	The RealView Debugger initialization file.
RVISS_0.auc	Target configuration file for the RVISS ARM7TDMI target.
RVISS_1.auc	Target configuration file for the RVISS ARM926EJ-S target.
RVISS_2.auc	Target configuration file for the RVISS ARM1176JZF-S target.

3. Create your own Debug Configurations.

---

**Note**

All users have the default configuration files listed in Table 3-10. It is recommended that you do not customize these default Debug Configurations, but rather create new ones.

---

4. If the default memory map BCD files supplied with *RealView Development Suite* (RVDS) do not meet your requirements, then create your own custom memory maps.

If you have to create your own custom memory maps for your development platforms, then:

- a. Store the BCD files containing the memory map definitions in the home directory.
- b. Assign the BCD files to the appropriate Debug Configurations.

5. If you want users to keep their own `rvdebug.brd` file, then change the name of the `rvdebug.brd` file. For example, change the name to `devteam.brd`. Each user can then switch between their own board file and the common board file.

If you want all users to use the same `rvdebug.brd` file, then you do not have to change the file name.

6. Exit RealView Debugger.

When you exit RealView Debugger, the following files might be created:

- `exphist.sav`
- `rvdebug.sav`
- `settings.sav`.

Remove these files if they exist. Also remove any `*.bak` files.

7. Remove any additional files that you do not want to distribute. These include the following user-specific files:

- `armreg.sig`
- `rvdebug.aws`
- `rvdebug.ini`.

———— **Note** ————

Be aware that these files are recreated when you next start RealView Debugger using this home directory. Also, if you have renamed the `rvdebug.brd` file, then a new default `rvdebug.brd` file is created.

See also:

- *Summary of supplied BCD files* on page 1-19
- *Changing connection settings* on page 3-10
- *Loading a different board file* on page 3-15
- *Distributing Debug Configurations to other machines and users* on page 3-58
- Chapter 2 *Customizing a Debug Interface configuration*
- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals*
- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
  - *Changing the name of a Debug Configuration* on page 3-17
  - *Copying an existing Debug Configuration* on page 3-18
  - *Deleting a Debug Configuration* on page 3-19.

### 3.18 Distributing Debug Configurations to other machines and users

You can distribute RealView Debugger Debug Configurations so that all members of a development team have access to the same targets and configurations.

To distribute RealView Debugger Debug Configurations to other users:

1. Prepare your Debug Configurations for distribution.
2. A Debug Configuration is stored in a .brd file and has a reference to a target configuration file. Therefore, as a minimum, you must distribute the .brd file, and any target configuration files that it references.

For example, you might have created an RVISS\_3 Debug Configuration that references the rviss\_3.auc target configuration file. Therefore, you must distribute both the .brd file and the rviss\_3\_0.auc file.

3. If you have created your own custom memory maps in BCD files, then make sure you also distribute these BCD files.

---

**Note**

---

If your Debug Configurations reference any default memory map BCD files supplied with RVDS, these do not have to be distributed.

---

4. Place the directory containing the Debug Configurations on a network location to which all users have access.
5. To access the Debug Configurations use one of the following methods:
  - Load the .brd file directly from the network location. In this case, you must set the RVDEBUG\_SHARE environment variable to the location of the file on the network.
  - Copy the home directory from the network location to your own machine (for example, C:\RVD\_home\DevTeam), and then start RealView Debugger with the following command (replace ... with the version and build number of your installation):
 

```
"C:\Program Files\ARM\RVD\Core\...\win_32-pentium\bin\rvdebug.exe"
--install="C:\Program Files\ARM\RVD\Core\...\win_32-pentium"
--home="C:\RVD_home\DevTeam"
```
  - Copy the files from the network directory to your own home directory.

---

**Note**

---

Be aware that this might overwrite your .brd file. Therefore, you might want to rename your existing .brd file. You can switch between different board files if required.

---

See also:

- *Considerations when distributing Debug Configurations* on page 3-59
- *The RealView Debugger search path* on page 1-17
- *Summary of supplied BCD files* on page 1-19
- *Loading a different board file* on page 3-15
- *Preparing Debug Configurations for distribution* on page 3-56.

### 3.18.1 Considerations when distributing Debug Configurations

If you want your users to directly access the Debug Configurations from a network location, rather than copying them to their local machine, then be aware of the following:

- Set the RVDEBUG\_SHARE environment variable to the location of the shared home directory (for example, H:\DevTeam). If you do not, references to target configuration files and custom BCD files must include the full path to the network location. Otherwise, RealView Debugger is unable to locate these files.
- Users must not specify the location of the shared network directory as a home directory when they start RealView Debugger. If they do, then RealView Debugger attempts to create the following user-specific files in the network directory:
  - armreg.sig
  - rvdebug.aws
  - rvdebug.ini.

Therefore, if a user changes their workspace settings, the changes are picked up by other users. Also, RealView Debugger attempts to create a new default rvdebug.brd file, if it does not exist.

#### See also

- *The RealView Debugger search path* on page 1-17.

### 3.19 Example of setting up an Integrator board and processor core module

This example demonstrates how to use the Connection Properties window to create a specific Integrator/AP and processor core module target configuration. It shows how to use a predefined Board/Chip Definition file, with extension .bcd, to set up your target.

After you set up your target, the example also demonstrates how you can connect to it using DSTREAM or RealView ICE with the Connect to Target window, and verify that RealView Debugger can connect to the target.

The example is split into the following sections, which must be executed in this sequence:

1. *Setting up the hardware and Debug Interface*
2. *Creating the new Debug Configuration*
3. *Configuring the new Debug Configuration on page 3-61*
4. *Linking board groups to the new Debug Configuration on page 3-62 (optional)*
5. *Connecting to the new target on page 3-63*
6. *Viewing the new target definition on page 3-64.*

#### 3.19.1 Setting up the hardware and Debug Interface

The first step is to set up the hardware and configure the DSTREAM or RealView ICE unit:

1. Make sure that your Integrator/AP board and processor core module are connected and switched on. This example uses the ARM940T processor, but you can use any processor core module supported by the Integrator/AP.
2. Make sure that you have the DSTREAM and RealView ICE host software installed, and that the DSTREAM or RealView ICE unit is connected and configured for use with RealView Debugger. If you have not configured the DSTREAM or RealView ICE unit, do so now.

##### See also

- *Creating the new Debug Configuration*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities.*

#### 3.19.2 Creating the new Debug Configuration

The next step is to create the new Debug Interface:

1. Start RealView Debugger.
2. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
3. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the required Debug Interface. For this example, expand RealView ICE.
4. Click **Add** for the RealView ICE Debug Interface. The RVConfig utility is displayed.
5. Configure your RealView ICE unit as required.
6. Save your changes, and close the RVConfig utility.
7. A new Debug Configuration is added. Figure 3-7 on page 3-61 shows an example after the RVI Debug Configuration is added.

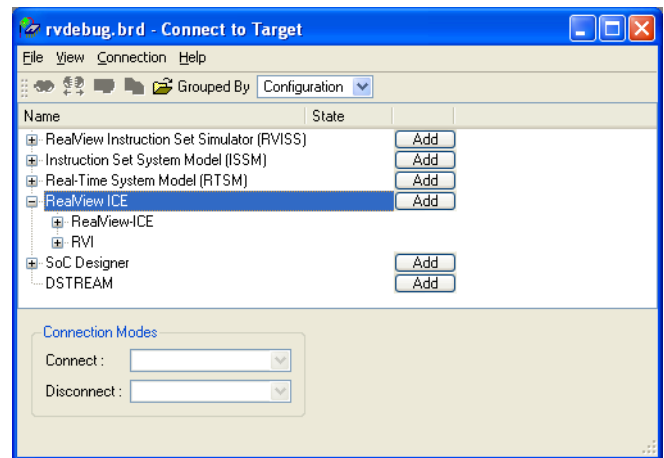


Figure 3-7 New Debug Configuration added

8. Right-click on the new Debug Configuration (RVI) to display the context menu.
9. Select **Rename Configuration** from the context menu.
10. Rename the Debug Configuration to **MP3Player**.

#### See also

- *Configuring the new Debug Configuration*
- *DSTREAM and RealView ICE Using the Debug Hardware Configuration Utilities.*

### 3.19.3 Configuring the new Debug Configuration

You must now configure the new Debug Configuration:

1. Right-click on the new MP3Player Debug Configuration to display the context menu.
2. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
3. Click the **Advanced** button to display the Connection Properties window.

The new settings group for the MP3Player Debug Configuration entry is selected, and the contents are displayed in the right pane. Figure 3-8 on page 3-62 shows an example.

The file name in your Configuration setting might be different to that shown.

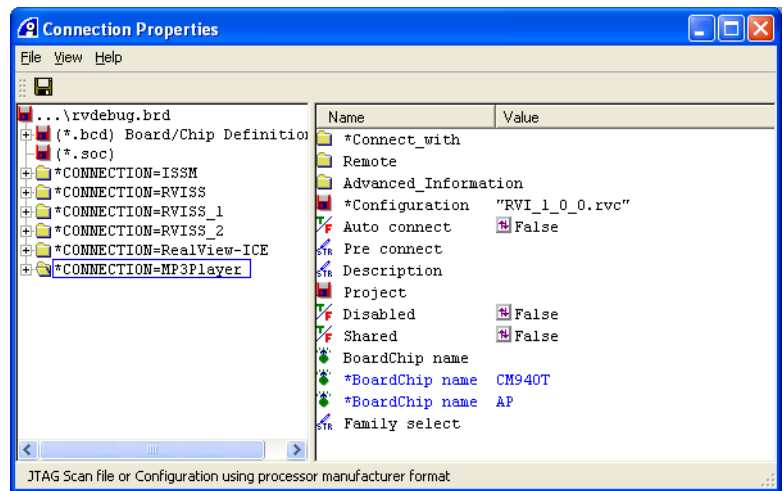


Figure 3-8 Displaying the new MP3Player connection properties

4. Right-click on Description in the right pane to display the context menu.
5. Select **Edit Value...** from the context menu.
6. Enter a short description for the new connection, for example Integrator/AP with ARM940 for MP3 product.

—— **Note** ——

Remember to press Enter to complete the entry.

7. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
8. Click the **OK** button to close the Connection Properties dialog box.

**See also**

- *Linking board groups to the new Debug Configuration.*

### 3.19.4 Linking board groups to the new Debug Configuration

The next step is to link board groups to the new connection. This is not essential but it gives extended target visibility and enables you to view register contents and manipulate memory.

—— **Note** ——

The connection created in this example is used in other examples in the rest of this chapter. If you do not link the board groups, the contents of the Registers view differ from those shown here.

To link board groups:

1. Right-click on the MP3Player Debug Configuration to display the context menu.
2. Select **Properties...** from the context menu. The Connection Properties dialog box is displayed.
3. Click the **BCD files** tab.
4. Select **AP** from the Available Definitions list to select the Integrator/AP description.

5. Click the **Add** button to add the definition to the Assigned Definitions list.
6. Select **CM940T** from the Available Definitions list to select the ARM940T processor core module description.
7. Click the **Add** button to add the definition to the Assigned Definitions list.  
The target configuration settings are copied from the source connection. However, depending on the target hardware, you might have to configure other settings, for example to enable semihosting. If required, you can do this now for the new connection.
8. Click the **OK** button to save your changes and close the Connection Properties dialog box.

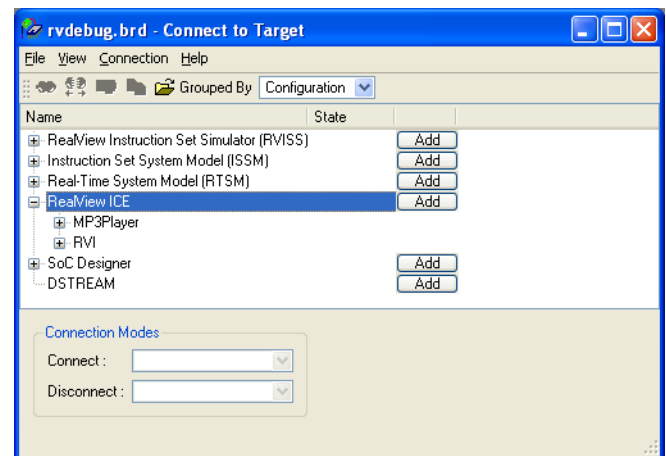
#### See also

- *Connecting to the new target.*

### 3.19.5 Connecting to the new target

To connect to your new target board and processor core module:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Select **Configuration** from the Grouped By list.
3. Expand the required Debug Interface. For this example, expand RealView ICE. The current RealView ICE Debug Configurations are displayed. Figure 3-9 shows an example:



**Figure 3-9 Connecting to the new target**

4. Connect to the target on the new MP3Player Debug Configuration in the usual way. RealView Debugger retrieves information specific to the target.

#### See also

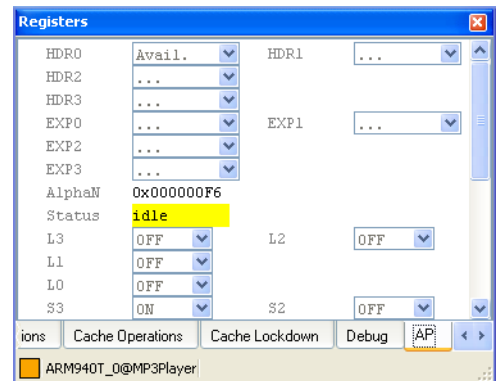
- *Viewing the new target definition on page 3-64.*



### 3.19.6 Viewing the new target definition

To view details about the new target hardware:

1. In the Code window, select **Registers** from the **View** menu to display the Registers view. Two new tabs are included at the bottom of the view, **AP** and **CM940T**.
2. Click the **AP** tab.  
RealView Debugger shows the abstraction of the hardware information specific to the Integrator/AP board. Figure 3-10 shows an example:



**Figure 3-10** AP tab in the Registers view

This tab view enables you to modify your Integrator/AP board features, such as the memory mapped peripherals.

3. To illustrate how RealView Debugger communicates directly with the Integrator/AP board, change the value of the L2 register in the Registers view to **ON**. The Red LED display on the Integrator/AP board is turned on.
4. Select the **CM940T** tab to see the abstraction of the hardware specific to the ARM940T processor core module. The **PRESENT** status of the Motherboard indicates that the processor core module is connected to the Integrator/AP board.
5. In the Output view at the bottom of the Code window, click the **Cmd** tab. The display includes the following line:

```
Advanced_info searched in: BOARD=AP, BOARD=CM940T.
```

This line indicates that RealView Debugger is using applying the settings in the AP.bcd and CM940T.bcd files for the connection.

#### ———— Note ————

This information is only displayed when you first connect to the target in the current debugging session.

As a result, the memory map now contains the definitions required to use the Flash memory on the Integrator.

#### See also

- *Flash programming* on page 3-50
- the following in the *RealView Debugger User Guide*:  
— Chapter 13 *Examining the Target Execution Environment*.

## 3.20 Troubleshooting Debug Configurations

The following sections describe how to solve some problems you might encounter with Debug Configurations:

- *Problems with target-specific settings*
- *Problems with missing or corrupt configuration files*
- *Debug Configuration reports a General Error status* on page 3-66.

### 3.20.1 Problems with target-specific settings

If you have configured settings in the Advanced\_Information block of a Debug Configuration, and those settings are not being applied to a target connection, then check the names of the Advanced\_Information block groups in that Debug Configuration. Groups with names Default or All are used for all target connections in a Debug Configuration. To make sure that settings are applied to a specific target, you must create a target-specific group.

An Advanced\_Information block group name must reflect the target to which you are connecting. For example, a block named ARM is used when you connect to any ARM architecture-based target.

#### See also

- *The Debug Configuration Advanced\_Information block* on page 3-8.

### 3.20.2 Problems with missing or corrupt configuration files

If your working versions of configuration files are accidentally deleted, or become corrupted, RealView Debugger is unable to determine your target configuration. In this case, making a connection to your chosen target is not possible.

You can do one the following:

- If you have made a backup of your configuration, restore it as described in *Saving and restoring connection properties* on page 1-17.
- If it is acceptable to lose all of your Debug Configurations, program preferences, workspaces and other information that is stored in your debugger home directory, you can delete your home directory:
  1. Exit RealView Debugger.
  2. Locate the home directory the debugger is using.
  3. Delete the home directory.
  4. Restart RealView Debugger. It creates a new debugger home directory for you as it starts up, containing a default set of configuration files.
- If there are Debug Configurations that you want to keep:
  1. Exit RealView Debugger.
  2. Move or rename your home directory.
  3. Restart RealView Debugger. A new home directory is created and populated with the default set of configuration files.
  4. Use the hints given in *Performing manual file or directory backups* on page 1-18 to copy backed up files to your new home directory.

**See also**

- *Saving and restoring connection properties* on page 1-17
- *The RealView Debugger home directory* on page 1-16
- *Performing manual file or directory backups* on page 1-18
- the following in the *RealView Debugger User Guide*:
  - *Troubleshooting target connection problems* on page 3-60.

**3.20.3 Debug Configuration reports a General Error status**

If a Debug Configuration has the state **General Error** in the **Connect to Target** window, then check that any BCD file assignments for that Debug Configuration are still valid. This situation occurs when a **BOARD**, **CHIP**, or **COMPONENT** definition is assigned to your Debug Configuration, but you have either:

- Deleted the BCD file containing that definition.
- Placed the BCD file in a location that is not on the RealView Debugger search path.

———— **Note** ————

It is recommended that you save BCD files in your home directory.

---

**Checking a BCD file assignment is still valid**

To check that a BCD file assignment is still valid:

1. Right-click on the Debug Configuration name to display the context menu.
2. Select **Properties...** from the context menu to display the Connection Properties dialog box.
3. Click the **Advanced** button from to display the Connection Properties window.
4. Right-click on each blue `*BoardChip_name` entry in the right pane to display the context menu.
5. Select **Jump to Definition** from the context menu. If a **Not Found** message is displayed, then:
  - a. Right-click on `*BoardChip_name` entry in the Debug Configuration to display the context menu.
  - b. Either:
    - Select **Delete** from the context menu to delete the entry.
    - Select an existing BCD file from the list.
6. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
7. Click the **OK** button to close the Connection Properties dialog box.

**See also**

- *The RealView Debugger home directory* on page 1-16
- *The RealView Debugger search path* on page 1-17
- the following in the *RealView Debugger User Guide*:
  - *Troubleshooting target connection problems* on page 3-60.

# Chapter 4

## Configuring Custom Memory Maps, Registers and Peripherals

This chapter describes in detail how to describe the memory map, memory mapped registers, and peripheral registers of your development platform to RealView® Debugger. It includes:

- *About configuring custom memory maps, registers, and peripherals* on page 4-2
- *Assigning a board, chip, or component group to a Debug Configuration* on page 4-7
- *Using the supplied BCD files* on page 4-12
- *Creating a BCD file to use as a template* on page 4-13
- *Basic procedure for creating BCD files* on page 4-15
- *Creating a new BCD file* on page 4-16
- *Creating and naming a board, chip, or component group* on page 4-18
- *Assigning board/chip definitions* on page 4-21
- *Setting top of memory* on page 4-24
- *Creating a memory map block* on page 4-29
- *Creating an enumeration for setting register values* on page 4-35
- *Creating a custom memory mapped register* on page 4-37
- *Creating a custom peripheral* on page 4-40
- *Creating the register tab for displaying custom registers and peripherals* on page 4-44
- *Setting up controlled memory blocks* on page 4-49
- *Creating memory map rules* on page 4-47
- *Creating a concatenated register* on page 4-54
- *Troubleshooting BCD files* on page 4-58.

## 4.1 About configuring custom memory maps, registers, and peripherals

You configure custom memory maps and registers in *Board/Chip Definition* (BCD) files (\*.bcd). A set of BCD files for many ARM® processors and development boards is provided with RealView Debugger.

See also:

- *BCD file configuration entries*
- *The board/chip definition Advanced\_Information block* on page 4-3
- *Board, chip, and component groups* on page 4-5
- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Using the examples* on page 4-6.

### 4.1.1 BCD file configuration entries

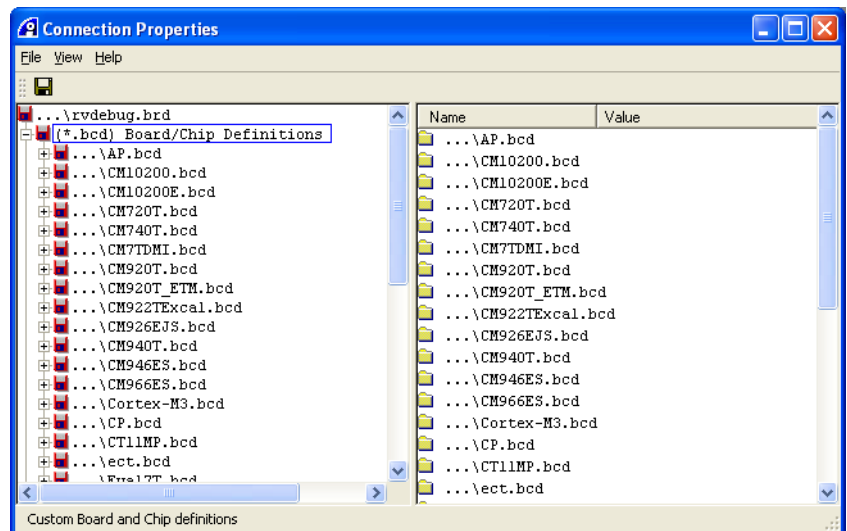
A BCD file can contain one or more configuration entries that enable you to describe the memory map, memory mapped registers, and peripheral registers of your development platform to RealView Debugger. This makes it possible for the debugger to:

- Present peripheral registers in a more human-readable format. By defining the addresses of I/O registers, and the bit fields within them, the debugger can display tabs in the Registers view where you can access these registers.
- Enable operations involving target memory to take account of the target memory map. By defining a memory map, the debugger can check that memory is used correctly, including refusing to load programs where there is no memory, and automatically invoking Flash memory programming routines.

#### Board/chip definitions

BCD file configuration information defines a hierarchy, starting from the general board-level and becoming more specific, through whole chips to component modules on a chip. This is achieved by creating one or more BOARD, CHIP, and COMPONENT groups in the BCD file. These groups are called *board/chip definitions*. However, RealView Debugger does not distinguish, functionally, between the different group names and you can use them as you require.

When RealView Debugger starts up, it searches for files with the extension .bcd and loads them into a group called (\*.bcd) Board/Chip Definitions. Figure 4-1 on page 4-3 shows an example:



**Figure 4-1 Viewing .bcd files in the Connection Properties window**

The name of each BOARD, CHIP, and COMPONENT group defined in the BCD files detected by RealView Debugger can be referenced from:

- any Debug Configuration
- another BCD file, or another group within the same BCD file.

In this way, the Debug Configuration defines a hierarchy of configuration details from the board level to the processor level. This makes the description of your development platform independent of the Debug Configuration used to access it, and makes it easier to reuse these descriptions in different debugging sessions.

For example, you can define a Debug Configuration that references:

- the BOARD=AP group in the file AP.bcd, to access the ARM Integrator™/AP board registers and memory map
- the BOARD=CM940T group in the file CM940T.bcd, to access the ARM940T™ processor core module registers and memory map.

#### See also

- *Summary of supplied BCD files* on page 1-19.

### 4.1.2 The board/chip definition Advanced\_Information block

A special group of settings is available in each board/chip definition in a BCD file, called the Advanced\_Information block. A BCD file contains one or more board/chip definitions that are specific to the target for which the file is created.

You cannot rename the top-level Default group of the Advanced\_Information block. For BCD files, it is recommended that you create your own named groups, and then delete the Default group.

## Naming an Advanced\_Information block group

RealView Debugger enables you to create multiple groups in the Advanced\_Information block of a board/chip definition. Each group has a unique name that relates to the target to which the settings are to apply. The name can be one of the following:

- a vendor, such as ARM
- a processor family, such as ARM9
- a partial processor name, such as ARM92
- a processor name, such as ARM940T
- a complete target connection name, such as ARM940T\_0.

## Matching of Advanced\_Information block group names and targets

When you connect to a target in a Debug Configuration that has a board/chip definition assigned, RealView Debugger attempts to match the target name with a group name in the Advanced\_Information block of that board/chip definition. The match is checked in the following sequence, starting with the left-most character:

1. Check the complete target connection name.  
The settings in a group called ARM926EJ-S\_0 are used only for an ARM926EJ-S™ processor that is at the first position on a DSTREAM or RealView ICE scan chain.
2. Check the target processor name.  
The settings in a group called ARM926EJ-S are used for an ARM926EJ-S processor at any position on a DSTREAM or RealView ICE scan chain. If there is more than one ARM926EJ-S processor, then the settings are used for all those processors.
3. Check the partial processor name.  
The settings in a group called ARM92 are used for all ARM92xx processors at any position on a DSTREAM or RealView ICE scan chain. For example, both ARM926EJ-S\_0 and ARM920T\_1 match.
4. Check the processor family name.  
The settings in a group called ARM9 are used for all the ARM9® family of processors at any position on a DSTREAM or RealView ICE scan chain. For example, both ARM926EJ-S\_0 and ARM966E-S\_1 match.
5. Check the vendor name.  
The settings in a group called ARM are used for any ARM processor at any position on a scan chain.
6. If no target matches are found in the board/chip definition assigned to the Debug Configuration, then attempt to match against groups in the main Debug Configuration settings.
7. If no target match is found in any Advanced\_Information block, then use the settings in the groups named Default or All.

### ————— **Note** —————

If you delete all groups within an Advanced\_Information block, then RealView Debugger creates a Default group in that block when the Connection Properties are refreshed.

When multiple blocks have a common string of characters, then the name with the longest match is used. For example, if there are two blocks named ARM9 and ARM940T, then the block ARM940T is used when connecting to an ARM940T processor.

**See also**

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Memory mapping Advanced\_Information settings reference* on page A-20.

**4.1.3 Board, chip, and component groups**

Configuring a board/chip definition requires that you create a BOARD, a CHIP, or a COMPONENT group. RealView Debugger uses these groups in the same way regardless of the type you create. However, it is recommended that you use them as follows so that it is clear what the group describes:

**BOARD** Use this to define target boards as a whole, such as the memory map and peripheral components.

For example, the CP.bcd file in your default settings directory describes the memory map, registers, and peripherals registers of the Integrator™/CP motherboard. It contains the BOARD=CP group, which defines the CP board/chip definition.

**CHIP** Use this to define significant devices on your development board.

For example, the supplied BCD file for the Evaluator-7T development board (. . \Eva17T.bcd) references the ARM processor-based KS32C50100 to define the processor and ASIC components of that device. It contains the CHIP=KS32C50100 group, which defines the KS32C50100 board/chip definition.

It is recommended that you use a CHIP entry if you want to use a device on more than one board, or if the device is in itself complex.

**COMPONENT** Other components that are not covered by a BOARD or CHIP group.

When you start RealView Debugger, the name of each group is added to:

- the related BoardChip\_name context menu
- the associated List Selection dialog box.

You use the Available Files list or context menu when assigning a board/chip definition to a Debug Configuration.

---

**Note**

You must only configure settings that relate to memory map or memory mapped registers in these groups. All other settings must be configured at the CONNECTION level.

---

**See also**

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Assigning a board, chip, or component group to a Debug Configuration* on page 4-7
- *Memory mapping Advanced\_Information settings reference* on page A-20
- Chapter 3 *Customizing a Debug Configuration*.

**4.1.4 Suggested naming convention for memory map related settings groups**

When creating registers, peripherals, and concatenated registers you can:

- specify the addresses relative to a memory map block
- set up enumerations to use for setting the values
- define rules for activating memory blocks depending on a register value



- define bit fields.

Table 4-1 lists suggested prefixes for the names of these entities. This helps you to distinguish the names when referencing them from other settings.

**Table 4-1 Suggested naming convention**

Setting group type	Setting group	Prefix
Memory blocks	Memory_block	M_
Enumerations for register values	Register_Enum	E_
Register bit fields	Bit_fields	B_
Concatenated registers	Concat_Register	C_
Rules for mapping memory blocks depending on a register value	Map_rule	R_

#### 4.1.5 Using the examples

The examples in this chapter modify the board file stored in your RealView Debugger home directory. By default, this is called `rvdebug.brd`. Debug Interface configuration files might also be stored in this directory, for example `.rvc` files or `.smc` files.

It is recommended that you back up this directory before starting the examples, so that you can restore your original configuration later. In these examples:

- It is assumed that you are starting with the default board file as installed with the base product. Depending on the type of installation you choose, and your other ARM products, the contents of this file might be different from the one shown here. However, the methods described can be applied to any board file.
- Board file entries are created and renamed. The names used are for illustration only and you can change them as you require. Duplicate names are not allowed.

#### ———— Note ————

Before you customize a Debug Configuration, you must disconnect from all the targets in that configuration.

#### See also

- *Saving and restoring connection properties* on page 1-17
- *Restoring the default connections and configurations* on page 3-55
- *Troubleshooting Debug Configurations* on page 3-65.

## 4.2 Assigning a board, chip, or component group to a Debug Configuration

The settings in a \*.bcd file are only used if:

- A board, chip, or component group in that .bcd file is referenced from a Debug Configuration.
- An Advanced\_Information block name is one of the following:
  - The whole or part of the name of the target being connected.  
For example, settings in blocks named either ARM9 or ARM940T are used if you connect to an ARM940T processor. However, when multiple block names have a common string (ARM9 in this example), then the block with the longest match is always used, ARM940T in this case.
  - Default
  - All.

Typically, groups named Default and All are not required in a BCD file.

- The identification string, id\_match, specified in the Advanced\_Information group, matches the hardware ID string returned by the target.  
If more than one assigned configuration group matches the target hardware, the group with the longest match is used.

This section describes how you assign board/chip definitions. These examples assume that you are assigning board/chip definitions to a DSTREAM or RealView ICE Debug Configuration.

If this is the first time you have used DSTREAM or RealView ICE, make sure that you configure the debug interface before trying to connect.

### ————— **Note** —————

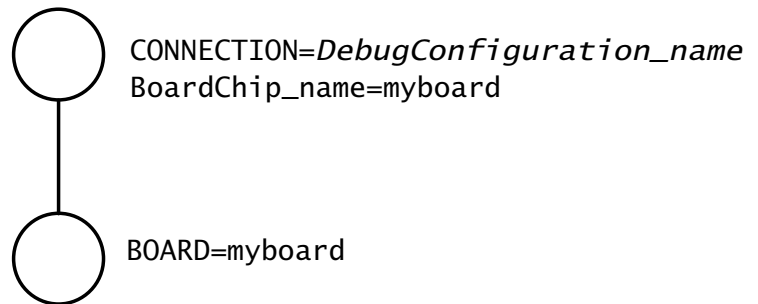
The examples in this section use existing board/chip definitions. However, you can create and use your own board/chip definitions.

See also:

- *Assigning one board group to a Debug Configuration* on page 4-8
- *Assigning several board groups to a Debug Configuration* on page 4-8
- *Assigning one or more board groups to another board group* on page 4-9
- *Assigning one or more board groups to a multiprocessor Debug Configuration* on page 4-11.

### 4.2.1 Assigning one board group to a Debug Configuration

To use the memory mapping features of RealView Debugger, you must define the memory map of your development platform in one or more board/chip definitions, which might be stored in difference BCD files. You must then reference the board/chip definitions from the Debug Configuration that defines the connections to the targets on your development platform. Figure 4-2 shows a basic configuration.



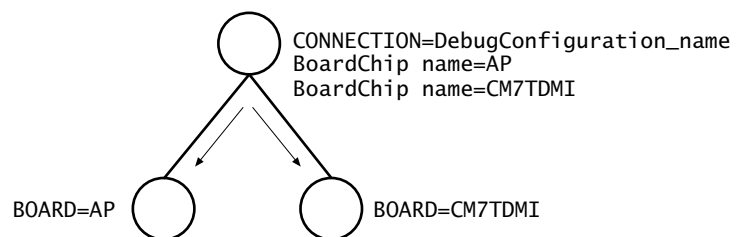
**Figure 4-2** Assigning one board to a connection

#### See also

- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14
- Chapter 2 *Customizing a Debug Interface configuration*
- Chapter 3 *Customizing a Debug Configuration*.

### 4.2.2 Assigning several board groups to a Debug Configuration

You might want to assign several groups to a single connection if the groups represent different, possibly optional, parts of the same development platform. For example, the Integrator/AP motherboard definition AP and an Integrator core module definition such as CM7TDMI. Figure 4-3 shows this kind of layout in tree form.



**Figure 4-3** Assigning two boards to a Debug Configuration

When you reference multiple boards, RealView Debugger merges the settings from each matching group. Therefore the complete configuration is the combined configurations of all of the matching groups. If the same setting is specified in more than one group, the specification in the group that is listed first in the (\*.bcd) Board/Chip Definitions group is used, for example AP in this example.

#### See also

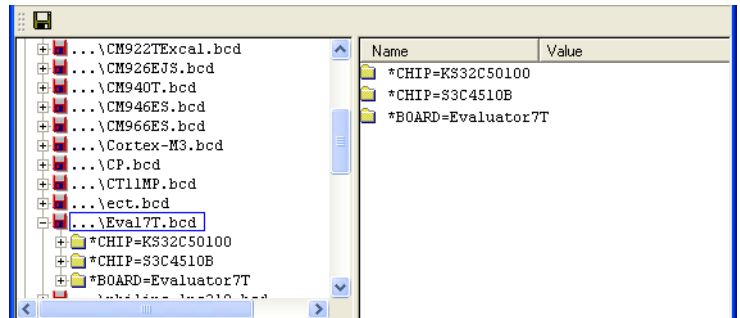
- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14

- Chapter 2 *Customizing a Debug Interface configuration*
- Chapter 3 *Customizing a Debug Configuration.*

### 4.2.3 Assigning one or more board groups to another board group

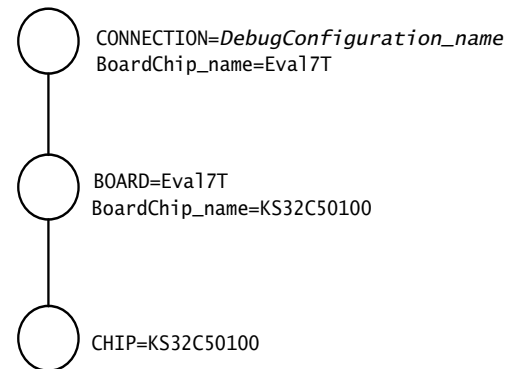
You might want to link several groups together so that you can share descriptions or simplify each part of a description. For example, the description of the ARM Evaluator-7T provided in Eval7T.bcd is split into a description of the board, \*BOARD=Evaluator7T, and a description of the processor on the board, KS32C50100 or S3C4510B depending on the version of the board.

Figure 4-4 shows an example:



**Figure 4-4 Board and chip groups for the Evaluator-7T**

Figure 4-5 shows the board and chip relationship in tree form.



**Figure 4-5 Tree view of the assigned groups in the Eval7T.bcd file**

Groups can contain BoardChip\_name references to other groups, so that you can build multi-layered descriptions. For example, if you are building a simple Ethernet router, you might use the network interface on the KS32C50100 with a second network interface provided by an AMD LANCE. Figure 4-6 on page 4-10 shows an example of this configuration.

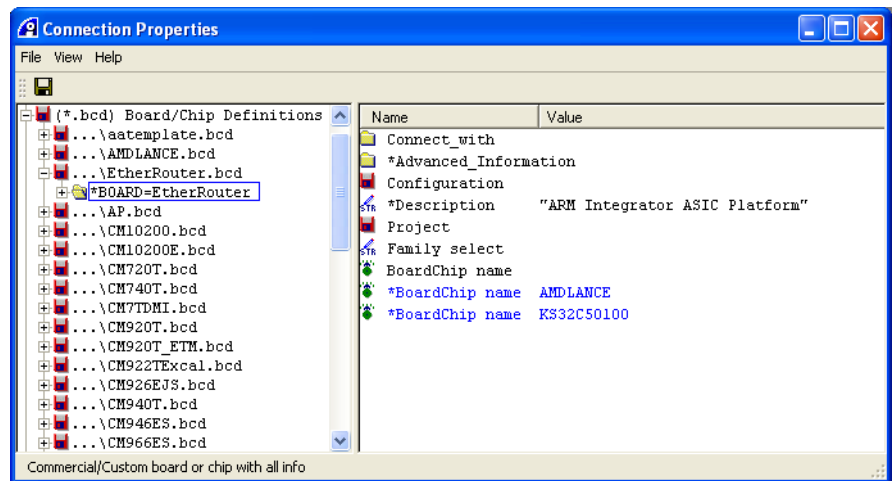


Figure 4-6 Board and chip groups for the EtherRouter board

Figure 4-7 shows the board and chip relationship in tree form.

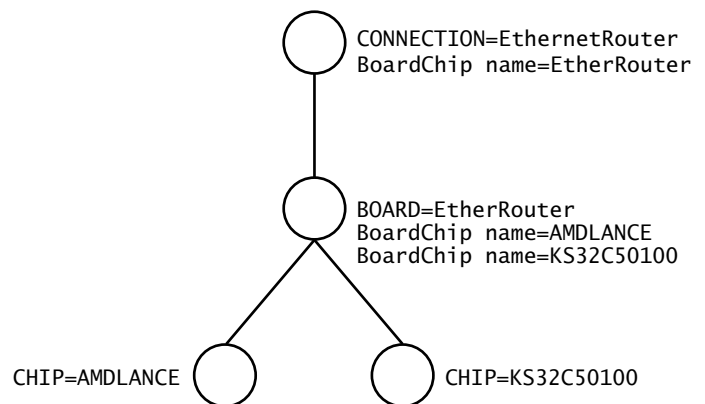


Figure 4-7 Tree view of the assigned groups for the EtherRouter board

#### ———— Note ————

You are not required to split your board up into distinct CHIP descriptions. You can create one BOARD description containing all of the required information. However, splitting your board up into distinct CHIP descriptions enables you to share descriptions or reuse a description for another development project.

#### See also

- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14
- *Basic procedure for creating BCD files* on page 4-15
- *Assigning board/chip definitions* on page 4-21
- Chapter 2 *Customizing a Debug Interface configuration*
- Chapter 3 *Customizing a Debug Configuration*.

#### 4.2.4 Assigning one or more board groups to a multiprocessor Debug Configuration

If you want to use RealView Debugger to debug a multiprocessor development platform, where some of the target configurations are different, you do so by defining multiple Advanced\_Information groups using names that match the target name.

For example, if you have a single Integrator CM920T, DSTREAM or RealView ICE names the connection ARM920T\_0. The \_0 in the name indicates that this target is on the first TAP position, that is position zero. If, in any BOARD, CHIP or COMPONENT, you create an Advanced\_Information group called ARM920T\_0, the entries in that group only apply to that target connection.

If you have two CM920T boards connected to an Integrator motherboard, DSTREAM or RealView ICE names them ARM920T\_0 and ARM920T\_1. If you create two Advanced\_Information groups called ARM920T\_0 and ARM920T\_1, you can configure each target connection independently. Figure 4-8 shows an example. If you also have a Default group, you can also have Advanced\_Information that applies to both processors linked to the connection.

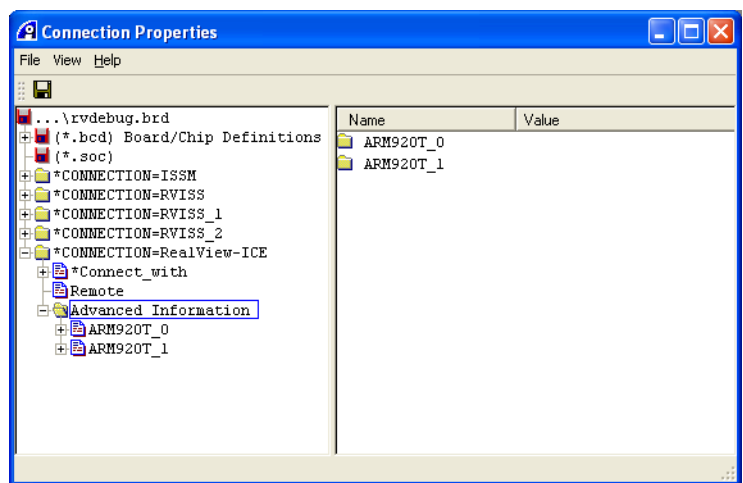


Figure 4-8 Customizing a two-processor Debug Configuration

#### See also

- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14
- *Basic procedure for creating BCD files* on page 4-15
- Chapter 2 *Customizing a Debug Interface configuration*
- Chapter 3 *Customizing a Debug Configuration*
- the following in the *RealView Debugger User Guide*:
  - *Connecting to multiple targets* on page 3-46.

### 4.3 Using the supplied BCD files

The BCD files (\*.bcd) provided with RealView Debugger are stored in your default settings directory:

C:\Documents and Settings\userID\Local Settings\Application  
Data\ARM\rvdebug\version\shadowbase\etc

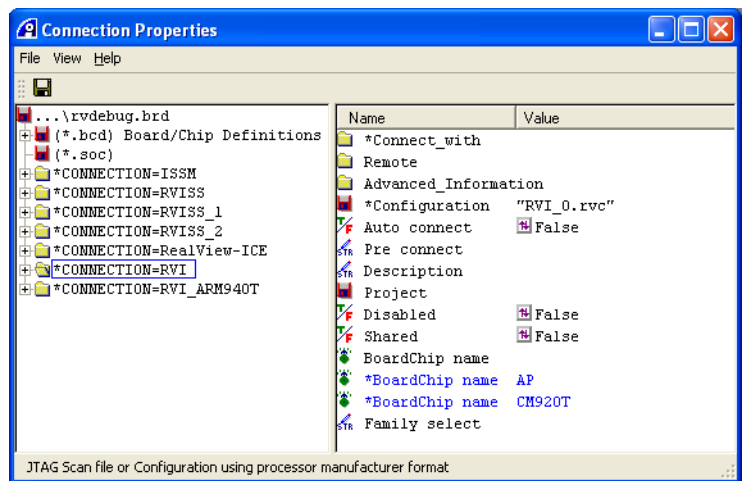
The files include details of the location and format of the custom registers, peripheral registers, and memory map available on the described target boards.

Use the supplied BCD files by referencing them from the Debug Configuration you use to communicate with your target. For example, if you are connecting to an ARM920T processor core module through DSTREAM or RealView ICE, modify the setting BoardChip\_name in the CONNECTION=RVI Debug Configuration group to reference the CM920T board/chip definition in the CM920T.bcd file.

#### ———— Note ————

If you upgrade to a later version of RealView Debugger you are provided with a new, and possibly different, version of these files. It is recommended, therefore, that you do not modify these files so that you can upgrade easily.

If you are using an Integrator/AP or Integrator/CP motherboard with a processor core module, you can combine the platform and processor core module descriptions by using multiple BoardChip\_name settings for the Debug Configuration. Figure 4-9 shows an example:



**Figure 4-9 Referencing two .bcd files in the Connection Properties window**

See also:

- *Summary of supplied BCD files* on page 1-19
- *Board, chip, and component groups* on page 4-5
- *Basic procedure for creating BCD files* on page 4-15
- *Assigning board/chip definitions* on page 4-21
- *Chapter 3 Customizing a Debug Configuration.*

## 4.4 Creating a BCD file to use as a template

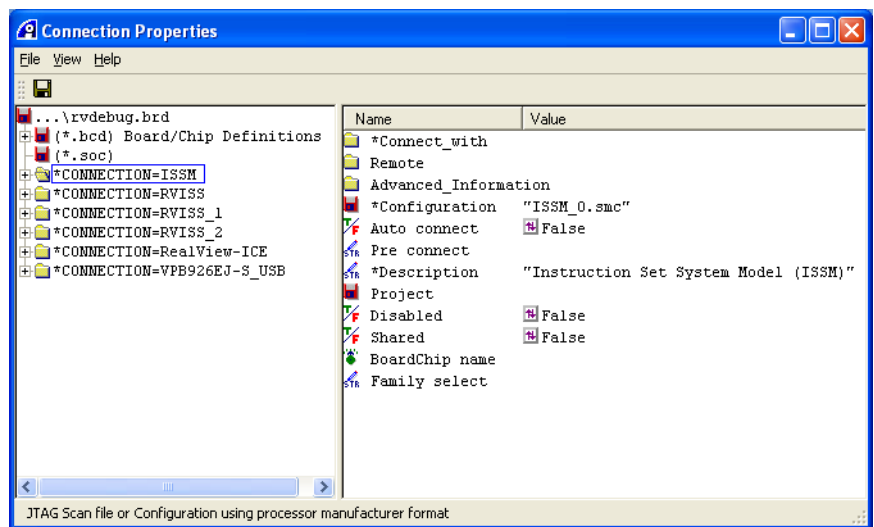
To create definitions for the memory map, memory mapped registers, and peripheral registers of your development platform, you are recommended to create a BCD file. You can then use this BCD file as a template for all future BCD files you create. You might want to create a different template for each vendor-specific target used on your development platform.

### ———— Note ————

The template BCD file created in this procedure is used as the starting point for many of the examples described in this chapter.

To create a BCD file to use as a template:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window. Figure 4-10 shows an example:



**Figure 4-10 Connection Properties window**

2. Expand the (\*.bcd) Board/Chip Definitions group in the left pane. A list of existing BCD files is displayed.
3. Save a copy of an existing BCD file:
  - a. Right-click on the AP.bcd file group to display the context menu.
  - b. Select **Save As...** from the context menu to display the Enter New Name dialog box.
  - c. Locate your RealView Debugger home directory.
  - d. Enter the name for your new BCD file, for example, **aatemplate.bcd**.
  - e. Click **Save** to save the BCD file and close the Enter New Name dialog box.
  - f. Select **Save Changes** from the **File** menu from the Connection Properties window menu to save your changes.
  - g. Select **Refresh** from the **File** menu to update the list of BCD files.
4. Expand the (\*.bcd) Board/Chip Definitions group again to show the list of BCD files.
5. Rename the BOARD= group as appropriate:
  - a. Expand the entry for the new BCD file, ...\aatemplate.bcd in this example.
  - b. Right-click on the BOARD= group in the left pane to display the context menu.



- c. Select **Rename** from the context menu to display the Group/Type Name Selector dialog box.
  - d. Enter an appropriate name, for example **ARM\_TEMPLATE**.
  - e. Click **OK** to close the dialog box.
6. Clear all settings that are currently configured:
  - a. Right-click on the BOARD=ARM\_TEMPLATE group in the left pane to display the context menu.
  - b. Select **Reset Contents** from the context menu. All settings in the BOARD=ARM\_TEMPLATE group are reset to the default settings.
7. Click the BOARD=ARM\_TEMPLATE group. The group settings are displayed in the right pane.
8. Change the Description in the right pane to an appropriate description:
  - a. Right-click on the Description to display the context menu.
  - b. Select **Edit Value...** from the context menu.
  - c. Enter **Template BCD File**. Make sure you press Enter to complete the entry.
9. Add a new Advanced\_Information group:
  - a. Expand the BOARD=ARM\_TEMPLATE group.
  - b. Expand the Advanced\_Information group.
  - c. Right-click on the Advanced\_Information group in the left pane to display the context menu.
  - d. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - e. Enter a name that is appropriate to the target for which this template is to be used. For this example, enter **ARM**.
  - f. Click **Create** to close the dialog box.
10. Delete the Default group:
  - a. Right-click on the Default group in the left pane to display the context menu.
  - b. Select **Delete** from the context menu. The Default group is deleted.
11. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

## 4.5 Basic procedure for creating BCD files

To create and use a new BCD file, follow this basic procedure:

1. Create one or more BCD files to store the configuration.  
See the BCD files provided with RealView Debugger for examples of how to set up a memory map, and memory mapped registers.
2. Create and name one or more BOARD, CHIP, or COMPONENT groups for the configuration. The type of groups you use depends on the type of hardware you are describing.

See also:

- *Managing configuration settings.*

### 4.5.1 Managing configuration settings

You configure your debug target by amending Debug Configuration entries using the Connection Properties window. This enables you to specify connection behavior, target visibility, image loading parameters, and disconnect options.

RealView Debugger provides great flexibility in how to configure these settings so that you can control your debug target and any custom hardware that you are using. This means that some settings can be defined in the top-level board file so that they apply:

- to Debug Configurations, for example CONNECTION=RealView-ICE
- on a per-board (or per-chip) basis using assigned board/chip definitions in one or more BCD files, for example CHIP=KS32C50100 in the file Eva17T.bcd.

---

#### Note

---

To avoid conflicts between settings when you reference multiple boards, follow the guidelines given in *Avoiding conflicts between linked board groups* on page 3-14.

---

#### See also

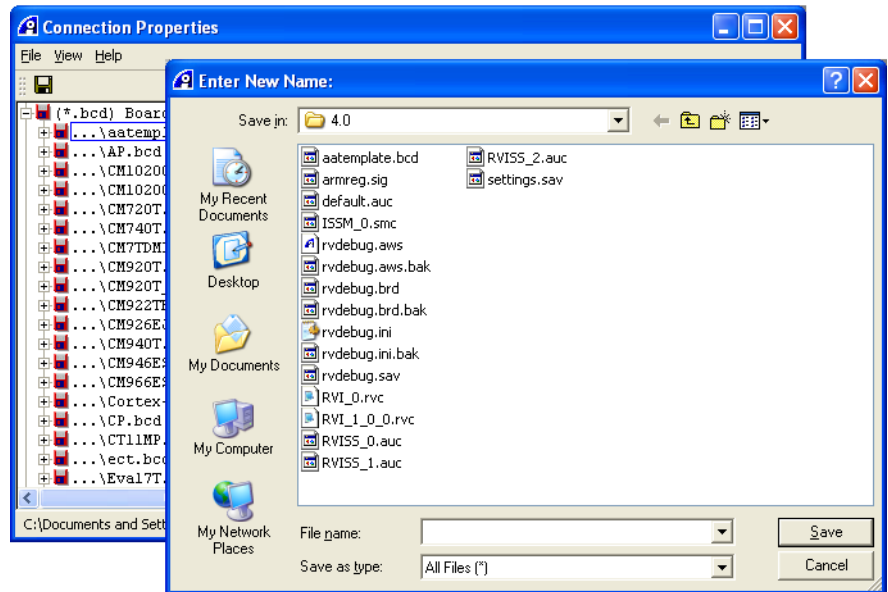
- *Board/Chip Definition files* on page 1-11
- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14
- *Board, chip, and component groups* on page 4-5
- *Creating a new BCD file* on page 4-16
- *Creating and naming a board, chip, or component group* on page 4-18
- *Assigning board/chip definitions* on page 4-21
- Chapter 3 *Customizing a Debug Configuration.*

## 4.6 Creating a new BCD file

To create a new BCD file, it is suggested that you create a blank BCD file to use as a template. However, you might want to view the settings in the BCD files provided with RealView Debugger, to see how the various settings are used.

To create a new BCD file using the `aatemplate.bcd`:

1. Create a template BCD file as described in *Creating a BCD file to use as a template* on page 4-13.
2. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
3. Expand the (\*.bcd) Board/Chip Definitions group in the left pane. A list of existing BCD files is displayed.
4. Right-click on the `... \aatemplate.bcd` entry.
5. Select **Save As...** from the context menu to display the Enter New Name dialog box. Figure 4-11 shows an example. The location displayed in this dialog box is the directory you last selected in the dialog box.



**Figure 4-11 Saving an existing BCD file with a new name**

By default, RealView Debugger searches for \*.bcd files in the current working directory, then in your home directory, and then in your default settings directory:

C:\Documents and Settings\userID\Local Settings\Application  
Data\ARM\rvdebug\version\shadowbase\etc

In this example, save the new file in your RealView Debugger home directory.

6. Enter the new filename. You must use the .bcd file extension when saving the file in your home directory.  
For this example, enter **AM.bcd**.
7. Click **Save**.

The New Name dialog box closes and the new name is displayed in the \*.bcd list. Although the new BCD entry replaces the BCD entry you used to make the copy, the original file still exists. To restore the complete list of BCD file:

- a. Select **Save Changes** from the **File** menu, to save the changes.
  - b. Select **Refresh** from the **File** menu. This restores the list of all current BCD files, including the new file you have created.
  - c. Expand the group (\*.bcd) Board/Chip Definitions to display the current list of BCD files. The list includes the new BCD file, that is AM.bcd.
8. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

---

**Note**

---

The general layout and controls of the RealView Debugger settings windows are described in the online help topic *Changing Settings*.

---

See also:

- *Board/Chip Definition files* on page 1-11
- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14
- *Board, chip, and component groups* on page 4-5
- *Creating a BCD file to use as a template* on page 4-13
- *Basic procedure for creating BCD files* on page 4-15
- *Creating and naming a board, chip, or component group* on page 4-18
- *Assigning board/chip definitions* on page 4-21
- Chapter 3 *Customizing a Debug Configuration*
- the following in the *RealView Debugger User Guide*:
  - *Redefining the RealView Debugger directories* on page 2-9.

## 4.7 Creating and naming a board, chip, or component group

To configure a new board/chip definition, you must create one or more BOARD, CHIP, and COMPONENT groups in a BCD file as required.

See also:

- *Adding a new board/chip definition*
- *Renaming a board/chip definition* on page 4-19

### 4.7.1 Adding a new board/chip definition

To create a new board/chip definition in a BCD file:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
2. Expand the (\*.bcd) Board/Chip Definitions group in the left pane. A list of existing BCD files is displayed.

#### ———— Note ————

The following steps assume that you have created the AM.bcd file as described in *Creating a new BCD file* on page 4-16.

3. Expand the required BCD file group. For example, expand the ... \AM.bcd group.
4. Right-click on the name of the BCD file to display the context menu. For example, right-click on the group ... \AM.bcd.
5. Select **Make New Group...** from the context menu to display the Group Type/Name selector dialog box.
6. Select the type of board/chip definition you want to create from BOARD, CHIP, or COMPONENT. For this example, select **CHIP**.
7. In the Group Name data field change the name from new to something suitable for your target, using only alphanumeric characters, underscore \_, and dash -. For this example, enter **S5471KT**.
8. Click **OK** to create the board/chip definition. The new S5471KT board/chip definition is created. Figure 4-12 shows an example:

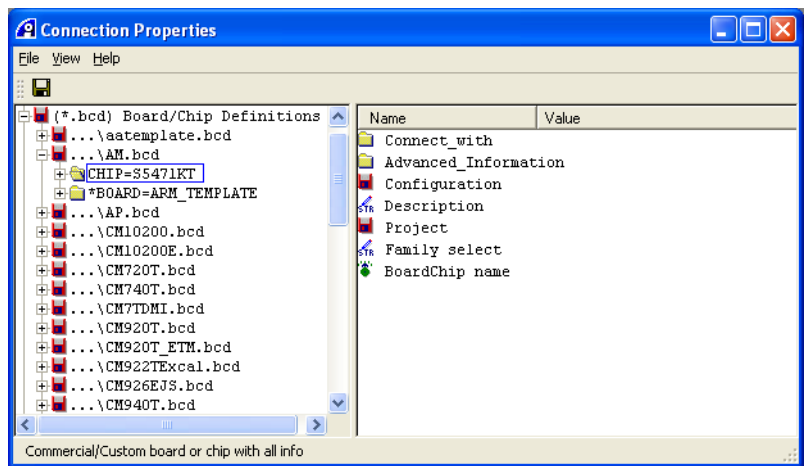


Figure 4-12 Viewing the new group in the BCD file

9. Delete the ARM\_TEMPLATE board/chip definition:
  - a. Right-click on the BOARD=ARM\_TEMPLATE group to display the context menu.
  - b. Select **Delete** from the context menu. The group is deleted.
10. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

---

**Note**

---

If multiple board/chip definitions are closely related, then you can create those definitions in a single BCD file. See the ... \CM966ES.bcd entry in the (\*.bcd) Board/Chip Definitions group for an example of this configuration.

---

**See also**

- *Board/Chip Definition files* on page 1-11
- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14
- *Board, chip, and component groups* on page 4-5
- *Creating a BCD file to use as a template* on page 4-13
- *Basic procedure for creating BCD files* on page 4-15
- *Creating a new BCD file* on page 4-16
- *Assigning board/chip definitions* on page 4-21
- Chapter 3 *Customizing a Debug Configuration*
- *The BOARD, CHIP, and COMPONENT groups* on page A-3.

#### 4.7.2 Renaming a board/chip definition

To rename an existing board/chip definition:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
2. Expand the (\*.bcd) Board/Chip Definitions group in the left pane. A list of existing BCD files is displayed.
3. Expand the entry of the BCD file containing the board/chip definition to be renamed. For example, expand the ... \AM.bcd group created in *Creating a new BCD file* on page 4-16.
4. Rename the ARM\_TEMPLATE board/chip definition:
  - a. Right-click on the BOARD=ARM\_TEMPLATE group to display the context menu.
  - b. Select **Rename** from the context menu to display the Group Type/Name selector dialog box.
  - c. In the Group Name field change the name to that required. You must use only alphanumeric characters, underscore (\_), and dash (-). For example, change the name to **ARM**.
  - d. Click **OK** to rename the board/chip definition.
5. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

**See also**

- *Board/Chip Definition files* on page 1-11
- *Summary of supplied BCD files* on page 1-19

- *Avoiding conflicts between linked board groups* on page 3-14
- *Board, chip, and component groups* on page 4-5
- *Creating a BCD file to use as a template* on page 4-13
- *Basic procedure for creating BCD files* on page 4-15
- *Creating a new BCD file* on page 4-16
- *Assigning board/chip definitions* on page 4-21
- *Chapter 3 Customizing a Debug Configuration*
- *The BOARD, CHIP, and COMPONENT groups* on page A-3.

## 4.8 Assigning board/chip definitions

To apply the settings in one or more board/chip definitions to a Debug Configuration, the BCD files containing those definitions must be in a location where RealView Debugger can find the files. Typically, if you create your own BCD files, then place them in your RealView Debugger home directory.

See also:

- *Assigning a board/chip definition to a Debug Configuration*
- *Assigning a board/chip definition to another board/chip definition* on page 4-22
- *Assigning multiple board/chip definitions to a Debug Configuration* on page 4-23
- the following in the *RealView Debugger User Guide*:
  - *Customizing a Debug Configuration* on page 3-20.

### 4.8.1 Assigning a board/chip definition to a Debug Configuration

To assign a board/chip definition to a Debug Configuration:

1. Expand the required Debug Interface in the Connect to Target window to display the related Debug Configurations. For example, expand on RealView ICE.
2. Right-click on the required Debug Configuration in the Connect to Target window to display the context menu. For example, right-click on RealView-ICE.
3. Select **Properties...** from the context menu to display the Connection Properties dialog box.
4. Click the **BCD files** tab to show the list of available board/chip definition.
5. Select your board/chip definition in the Available Definitions list.  
In this example, the list includes the board called CP and the chip called KS32C50100.  
For example, select **CP**.
6. Click the **Add** button. The CP definition is moved to the Assigned Definitions list.
7. Click the **OK** button to close the Connection Properties dialog box.

When you connect to a target using this Debug Configuration, the settings defined in the assigned board/chip definition are applied to that connection. Therefore, RealView Debugger is able to access the areas of your development platform as defined in that board/chip definition.

#### See also

- *Board/Chip Definition files* on page 1-11
- *The RealView Debugger search path* on page 1-17
- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14
- *Board, chip, and component groups* on page 4-5
- *Creating a BCD file to use as a template* on page 4-13
- *Basic procedure for creating BCD files* on page 4-15
- *Creating a new BCD file* on page 4-16
- Chapter 3 *Customizing a Debug Configuration*
- *The BOARD, CHIP, and COMPONENT groups* on page A-3.



## 4.8.2 Assigning a board/chip definition to another board/chip definition

A board/chip definition can reference one or more BOARD, CHIP, or COMPONENT groups from other board/chip definitions. In this way, you can build up an extended view of your development platform from other components.

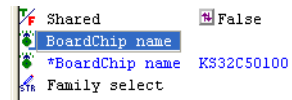
To assign a board/chip definition to another board/chip definition:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
2. Expand the (\*.bcd) Board/Chip Definitions group to show the current list of BCD files.
3. Select the required board/chip definition to which you want to assign another board/chip definition.
4. Right-click on the BoardChip\_name setting in the right pane to display the context menu.
5. Select the name of the required board/chip definition from the context menu.

### ———— Note ————

You might have to select **<More...>** from the context menu to find the name you require.

For example, select **KS32C50100**. A new setting is added to the right pane with an asterisk \* beside it. Figure 4-13 shows an example:



**Figure 4-13 Assigning the KS32C50100 group**

6. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

When you connect to a target using this Debug Configuration, the settings defined in the assigned board/chip definition are applied to that connection. Therefore, RealView Debugger is able to access the areas of your development platform as defined in that board/chip definition.

### ———— Note ————

Ignore any duplicate, overlapping memory, or register not found warning messages that are displayed when you connect.

## See also

- *Board/Chip Definition files* on page 1-11
- *The RealView Debugger search path* on page 1-17
- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14
- *Board, chip, and component groups* on page 4-5
- *Creating a BCD file to use as a template* on page 4-13
- *Basic procedure for creating BCD files* on page 4-15
- *Creating a new BCD file* on page 4-16
- *Chapter 3 Customizing a Debug Configuration*
- *The BOARD, CHIP, and COMPONENT groups* on page A-3.

### 4.8.3 Assigning multiple board/chip definitions to a Debug Configuration

To assign multiple board/chip definitions to another Debug Configuration:

1. Expand the required Debug Interface in the Connect to Target window to display the related Debug Configurations. For example, expand RealView ICE.
2. Right-click on the required Debug Configuration in the Connect to Target window to display the context menu. For example, right-click on RealView-ICE.
3. Select **Properties...** from the context menu to display the Connection Properties dialog box.
4. Click the **BCD files** tab to show the list of available board/chip definition.
5. Select the first board/chip definition in the Available Definitions list.  
For example, select **CP**.
6. Click the **Add** button. The CP definition is moved to the Assigned Definitions list.
7. Select the next board/chip definition in the Available Definitions list.  
For example, select **CM7TDMI**.
8. Click the **Add** button. The CM7TDMI definition is moved to the Assigned Definitions list.
9. Click the **OK** button to save your changes and close the Connection Properties dialog box.

When you connect to an ARM7TDMI target using this Debug Configuration, the settings defined in the assigned board/chip definitions are applied to that connection.

---

#### Note

---

Ignore any duplicate or overlapping memory warning messages that are displayed when you connect.

---

#### See also

- *Board/Chip Definition files* on page 1-11
- *The RealView Debugger search path* on page 1-17
- *Summary of supplied BCD files* on page 1-19
- *Avoiding conflicts between linked board groups* on page 3-14
- *Board, chip, and component groups* on page 4-5
- *Creating a BCD file to use as a template* on page 4-13
- *Basic procedure for creating BCD files* on page 4-15
- *Creating a new BCD file* on page 4-16
- *Troubleshooting BCD files* on page 4-58
- *Chapter 3 Customizing a Debug Configuration*
- *The BOARD, CHIP, and COMPONENT groups* on page A-3.

## 4.9 Setting top of memory

This example demonstrates how you can permanently set the top of memory for a target. It shows how to do this by updating the `Top_memory` setting in the Connection Properties of a BCD file linked to the Debug Configuration. After you have defined the setting, it is used whenever you connect to a target in that Debug Configuration.

Also, see the chapter that describes embedded software development in *ARM® Compiler toolchain Developing Software for ARM® Processors* for more details on using top of memory and stack heap values.

---

### Note

---

You cannot set top of memory for *RealView Instruction Set Simulator* (RVISS), ISSM, RTSM, and SoC Designer targets. There is no `@top_of_memory` symbol for these targets.

---

See also:

- *About top of memory, stack and heap*
- *Examining the top of memory value* on page 4-26
- *Setting top of memory* on page 4-26.

### 4.9.1 About top of memory, stack and heap

The default ARM C library implementation of the function `__user_initial_stackheap()` makes a semihosting call to read the debugger variable `top_of_memory`. The value returned is used to locate the stack base of your application. RealView Debugger uses target-dependent defaults for the stack and heap limits.

For more details about `__user_initial_stackheap()` function, see the following ARM Compiler toolchain documents:

- *Using the ARM C and C++ Libraries and Floating-Point Support*
- *ARM C and C++ Libraries and Floating-Point Support Reference*
- *Using the Linker*
- *Linker Reference*.

---

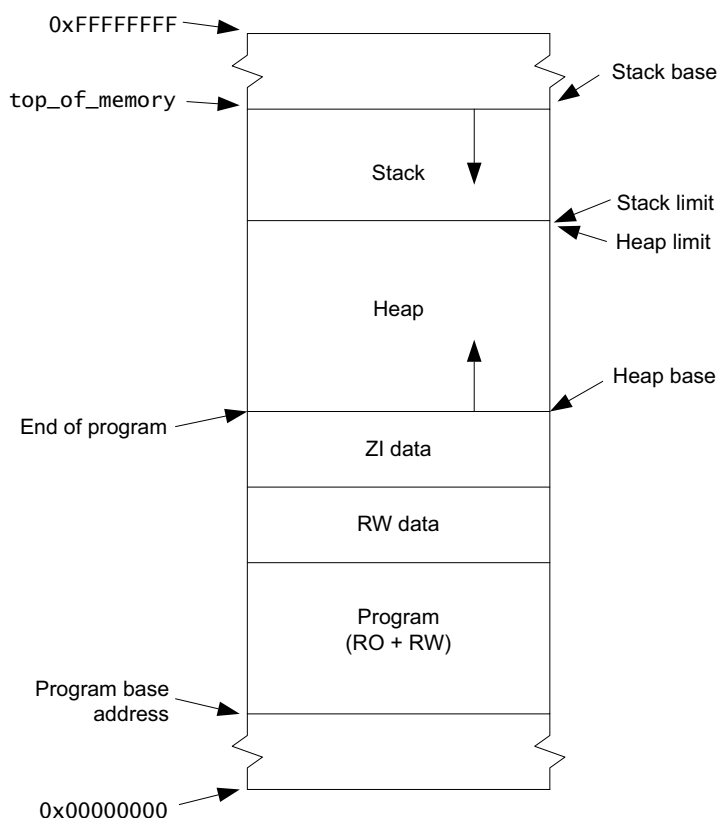
### Note

---

The default value of top of memory for ARM processors is `0x20000`.

---

Heap base is placed at the end of the image by default. The stack and heap are immediately below the `top_of_memory`. Figure 4-14 on page 4-25 shows the relationship between top of memory, stack, and heap.



**Figure 4-14 Relating top\_of\_memory to single section program layout**

**Note**

The top\_of\_memory value must be higher than the sum of the program base address, program code size, and program data size. If set incorrectly, the program might crash because of stack corruption or because the program overwrites its own code.

There is no requirement that the address specified by top\_of\_memory is at the true top of memory. A C or assembler program can use memory at addresses between top\_of\_memory and the true top of memory.

#### 4.9.2 Examining the top of memory value

To see the value of the top of memory, you can access the `top_of_memory` symbol. For example, enter the following CLI command:

```
> cexpression @top_of_memory Result is: 131072 0x00020000
```

#### 4.9.3 Setting top of memory

You can set the top of memory value in a board/chip definition used to configure the target. Although this configuration is for an Integrator/AP board with an ARM940T processor core module, the procedure for amending the settings is the same for any target.

To set the top of memory in a board/chip definition:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Display the setting for the RealView-ICE Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the required Debug Interface. For this example, expand RealView ICE.
  - c. Make sure that all targets are disconnected on the Debug Configuration (ARM940T\_0 in this example).
  - d. Right-click on the RealView-ICE Debug Configuration to display the context menu.
  - e. Select **Properties...** from the context menu to display the Connection Properties dialog box.
  - f. Click the **Advanced** button to display the Connection Properties window.
3. Assign the CP and CM940T board/chip definitions to the Debug Configuration:
  - a. Left-click on the BoardChip\_name setting in the right pane to display the context menu.
  - b. Select **CP** from the context menu.
  - c. Left-click on the original BoardChip\_name setting in the right pane (not the \*BoardChip\_name CP setting added in the previous step) to display the context menu.
  - d. Select **CM940T** from the context menu.

Figure 4-15 shows an example:

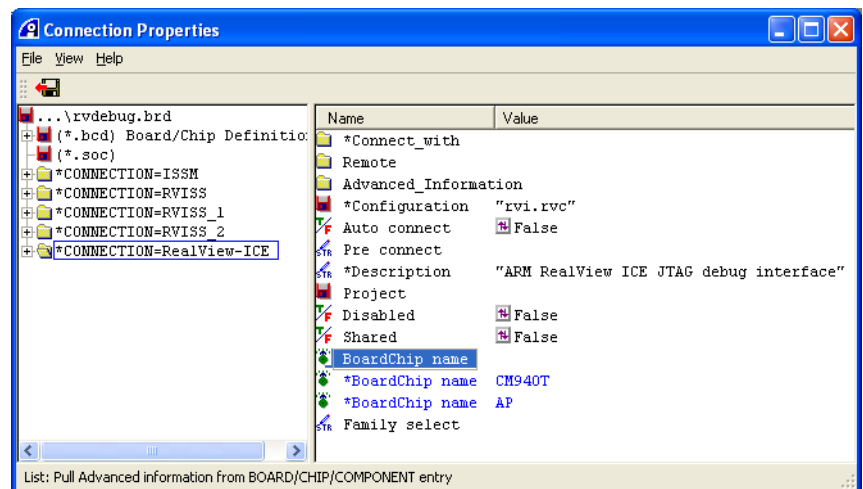


Figure 4-15 Connection Properties for the RealView-ICE Debug Configuration

4. Right-click on the \*BoardChip\_name CM940T setting to display the context menu.
5. Select **Jump to Definition** from the context menu. The BOARD=CM940T group is selected in the ... \CM940T.bcd entry. Figure 4-16 shows an example:

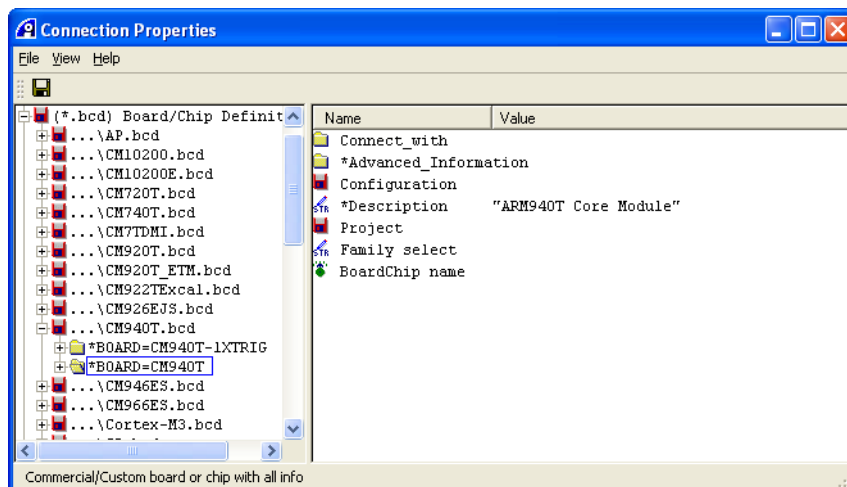


Figure 4-16 CM940T BOARD group selected

6. Expand the following entries in turn:
    - a. \*BOARD=CM940T
    - b. \*Advanced\_Information
    - c. \*ARM940T
  7. Select ARM\_config in the left pane.
  8. Set the value of Top\_memory in the right pane, as required. For example, set it to 0x40000 if your target has 256KB of RAM starting at location 0.
- **Note** ————
- Be sure to specify a value that is supported by your target.
9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
  10. Click the **OK** button to close the Connection Properties dialog box.

When you load a program compiled with the standard ARM C library to this target, the top\_of\_memory value you have set is used.

When you load a program, the debugger sanity-checks the Top\_memory setting in the assigned board/chip definition by checking that the locations below Top\_memory are writable. It issues a warning if they are not. However, your program might require much more RAM than is checked for by the debugger.

———— **Note** ————

Top\_memory is the board/chip definition setting name, but top\_of\_memory is the symbol name. You can examine the value of top\_of\_memory using the following CLI command:

```
cexpression @top_of_memory
```

**See also**

- *Board/Chip Definition files* on page 1-11
- *Avoiding conflicts between linked board groups* on page 3-14
- *ARM\_config group settings* on page A-11
- *Chapter 3 Customizing a Debug Configuration*
- *ARM® Compiler toolchain Developing Software for ARM® Processors*
- *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 Linker*
- *ARM® Compiler toolchain Linker Reference.*

## 4.10 Creating a memory map block

If you want to set up a memory map that is used automatically when you connect to a target, you must set up the memory map definition in the `Memory_block` group of a BCD file.

By default, a memory map block is uncontrolled. An uncontrolled memory map block is always displayed in the **Memory Map** tab of the Process Control view when you:

- assign the related BCD file to a Debug Configuration
- connect to a related target in that Debug Configuration.

---

### Note

---

See *Setting up controlled memory blocks* on page 4-49 if your target supports memory remapping.

---

See also:

- *Preparing the configuration*
- *Procedure for creating a memory map block* on page 4-30
- *Creating a memory map block that relates to custom registers and peripherals* on page 4-32
- *Viewing the memory map block* on page 4-33.

### 4.10.1 Preparing the configuration

Before you follow the procedure described in the next section:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the RVISS\_1 Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the required Debug Interface. For this example, expand RealView Instruction Set Simulator (RVISS).
  - c. Make sure that all targets are disconnected on the Debug Configuration (ARM926EJ-S in this example).
3. Right-click on the RVISS\_1 Debug Configuration to display the context menu.
4. Select **Properties...** from the context menu to display the Connection Properties dialog box.
5. Click the **Advanced** button to display the Connection Properties window. The RVISS\_1 Debug Configuration settings group is selected.
6. Remove any linked BCD groups to make sure that no pre-defined memory map related details are present.

For this example:

- a. Right-click on any BoardChip\_name setting to display the context menu.
- b. Select **Manage List...** from the context menu to display the Settings: List Manager dialog box.
- c. Select all the BoardChip\_name settings that you want to delete.
- d. Click **Remove**.



- e. Click **OK**. The Settings: List Manager dialog box closes, and the selected BoardChip\_name settings are deleted.
  - f. Select **Save Changes** from the **File** menu to save these changes.
7. Create a template BCD file if you do not already have one.
  8. Select **Refresh** from the **File** menu to refresh the list of BCD files.
  9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
  10. Click the **OK** button to close the Connection Properties dialog box.

#### See also

- *Board, chip, and component groups* on page 4-5
- *Setting up controlled memory blocks* on page 4-49
- *Troubleshooting BCD files* on page 4-58
- *The BOARD, CHIP, and COMPONENT groups* on page A-3.

### 4.10.2 Procedure for creating a memory map block

To create a memory map block for each area of memory on your development platform:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
  2. Expand the (\*.bcd) Board/Chip Definitions group.
- **Note** ————
- The following steps assume that you have created the template.bcd file as described in *Creating a BCD file to use as a template* on page 4-13.
- 
3. Make a copy of the template.bcd file:
    - a. Right-click on the ... \template.bcd group to display the context menu.
    - b. Select **Save As...** from the context menu to display the Enter New Name dialog box.
    - c. Change the location to your RealView Debugger home directory.
    - d. Enter the file name **MP3.bcd**.
    - e. Click the **Save** button to save the changes.
    - f. Select **Refresh** from the **File** menu to refresh the list of BCD files.
  4. Rename the BOARD group in the MP3.bcd file:
    - a. Expand the (\*.bcd) Board/Chip Definitions group.
    - b. Expand the ... \MP3.bcd group.
    - c. Right-click on BOARD=ARM\_TEMPLATE to display the context menu.
    - d. Select **Rename** from the context menu to display the Group/Type Name Selector dialog box.
    - e. Enter **MP3** in the Group Name field.
    - f. Click **OK** to save the change. The BOARD group is renamed to MP3.
  5. Expand the following groups in turn:
    - a. BOARD=MP3
    - b. Advanced\_Information

- c. ARM
  - d. Memory\_block.
6. Create a new memory map block under Memory\_block:
    - a. Right-click on Memory\_block, to display the context menu.
    - b. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
    - c. Enter a suitable name for the memory map block.  
For this example, enter **M\_SSRAM**.
    - d. Click **Create**.
  7. Edit the settings for the new M\_SSRAM group:
    - a. Click on the new M\_SSRAM group in the left pane to display the settings in the right pane.
    - b. Set the value of Start to the start address of the memory block, for example, **0x0**.
    - c. Set the value of Length in memory units for the memory block. For example, **0x20000** bytes for an ARM architecture-based processor.
    - d. For a target that supports TrustZone® technology, set Tz\_world to the required TrustZone world. The default is Secure.
    - e. Set the value of Description to **Static RAM**.

Figure 4-17 shows an example:

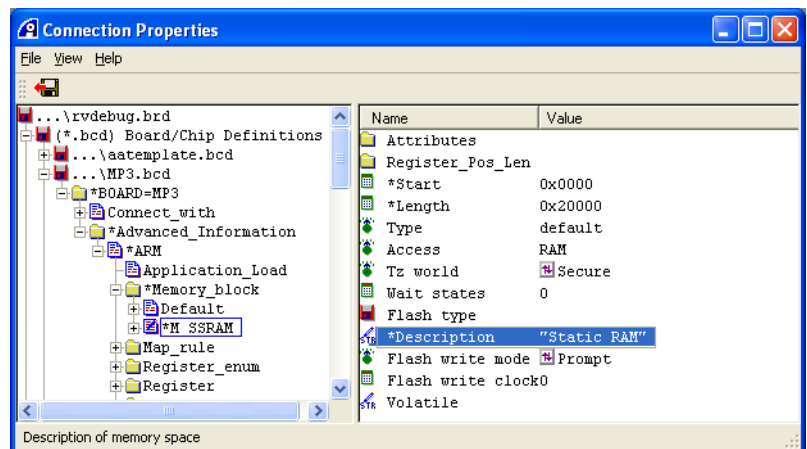


Figure 4-17 Viewing the contents of the new group

8. Assign the new BCD file to the RVISS\_1 Debug Configuration:
  - a. Select the CONNECTION=RVISS\_1 in the left pane.
  - b. Right-click on the BoardChip\_name setting in the right pane to display the context menu.
  - c. Select **MP3** from the context menu. A new \*BoardChip\_name MP3 setting is created.
9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

#### See also

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Setting up controlled memory blocks* on page 4-49
- *Troubleshooting BCD files* on page 4-58

- *Memory\_block* on page A-21.

#### 4.10.3 Creating a memory map block that relates to custom registers and peripherals

You can optionally create a memory map block that relates to custom registers you are going to define. This enables you to identify the locations of the custom registers relative to the base address of the memory map block. This memory map block is displayed in the **Memory Map** tab of the Process Control view.

---

**Note**

If you do not create a memory map block, then you must specify the absolute address for each custom register.

---

#### Procedure

To create a memory map block that is to be associated with your custom registers:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
2. Expand the (\*.bcd) Board/Chip Definitions group in the left pane.

---

**Note**

The following steps assume that you have created the MP3.bcd file as described in *Procedure for creating a memory map block* on page 4-30.

---

3. Expand the *Memory\_block* group.
4. If the *M\_SSRAM* memory block is defined:
  - a. Right-click on the *M\_SSRAM* group to display the context menu.
  - b. Select **Delete** from the context menu.
5. Rename the *Default* group under *Memory\_block* to **M\_REGS**.
6. Select the *M\_REGS* group in the left pane to display the group contents.
7. Set the values for the *M\_REGS* group:
  - a. Set the value of *Start* to the start address of the memory block, for example, **0x10000000**.
  - b. Set the value of *Length* in memory units for the memory block. For example, **0x800000** bytes for an ARM architecture-based processor.
  - c. For a target that supports TrustZone, set *Tz\_world* to the required TrustZone world. The default is Secure.
  - d. Set *Description* to **I/O Registers**.

Figure 4-18 on page 4-33 shows an example:

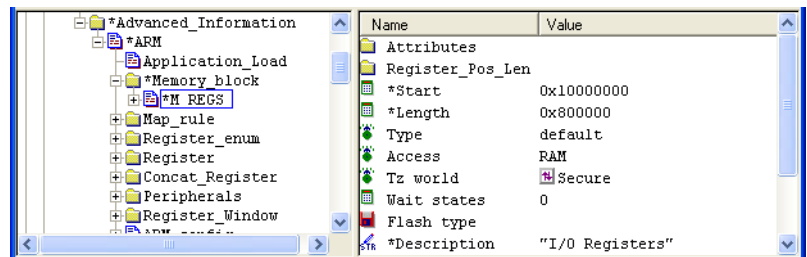


Figure 4-18 Configuring M\_REGS

8. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

#### See also

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Procedure for creating a memory map block* on page 4-30
- *Creating a custom memory mapped register* on page 4-37
- *Setting up controlled memory blocks* on page 4-49
- *Troubleshooting BCD files* on page 4-58
- *Memory\_block* on page A-21.

#### 4.10.4 Viewing the memory map block

To view the newly created memory map block:

1. Connect to the target in the RVISS\_1 Debug Configuration.
2. Select **Memory Map Tab** from the **View** menu to display the new memory map.
3. Expand the I/O Registers entry. Figure 4-19 shows an example:

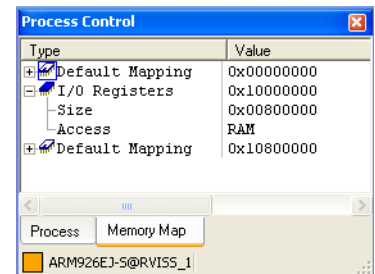


Figure 4-19 New memory map block in the Process Control view

4. Load an image, for example:  
`install_directory\RVDS\Examples\...\dhrystone\Debug\dhrystone.axf`
5. Click the **Memory Map** tab in Process Control view to display the new memory map.
6. Expand the memory map entries to see the details of the loaded image. Figure 4-20 on page 4-34 shows an example:

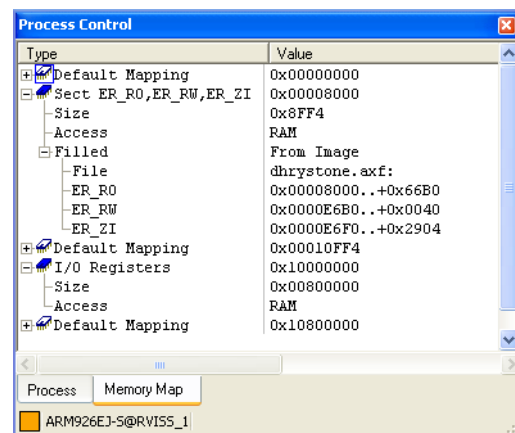


Figure 4-20 Memory map with image loaded

## 4.11 Creating an enumeration for setting register values

If any registers have enumerated values, then you can optionally set up enumerations, or names, for specific values that are used when the register value is displayed.

To set up enumerations to use for register values:

1. Locate the Connection Properties for the BCD file that you want to use to define your custom memory mapped registers:
  - a. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
  - b. Expand the (\*.bcd) Board/Chip Definitions group.
  - c. Expand the ...\*filename*.bcd group for the BCD file. For example, expand the ...\*MP3*.bcd group.
2. Expand the following groups for the BCD file in turn:
  - a. \*BOARD=MP3
  - b. Advanced\_Information
  - c. ARM (or the corresponding name specified in your BCD file)
  - d. Register\_enum.
3. Right-click on Register\_enum to display the context menu.
4. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
5. Enter **E\_name** for the name of the group. For example, enter **E\_ENABLE**.
6. Click **Create** to close the dialog box. A new E\_name group is created.
7. Select the new E\_name group in the left pane. The group contents are displayed in the right pane.
8. Set the value of Names in the right pane to the required enumeration names. The format of this settings is:

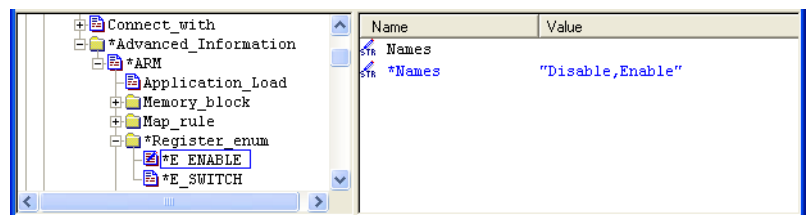
**enum1,enum2,...,enumN**

To number the enumerations 0, 1, ..., N. For example, **Disable,Enable** defines Disable as 0 and Enable as 1.

**enum1=val1,enum2=val2,...,enumN=valN**

To give your enumerations specific values. For example, if you want Enable to equal 7 and Disable to equal 0, then enter **Disable=0,Enable=7**. The assignment can be in any order, for example **Enable=7,Disable=0**.

For this example, specify **Disable,Enable**. Figure 4-21 shows an example:



**Figure 4-21 Creating enumerations**

9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

See also:

- *Avoiding conflicts between linked board groups* on page 3-14
- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Register\_enum group settings* on page A-26.

## 4.12 Creating a custom memory mapped register

A custom register allows RealView Debugger to access the memory mapped register on your development platform.

Before you create a custom memory mapped register, be aware of the following:

- If your register is to have enumerated values, you must first create the required register enumerations.
- If your register is to be associated with a memory map block you must first create the required memory map block.

See also:

- *Creating the custom register*
- *Defining a separate bit field for a custom register* on page 4-38
- *Creating a memory map block that relates to custom registers and peripherals* on page 4-32
- *Creating an enumeration for setting register values* on page 4-35.

### 4.12.1 Creating the custom register

To create a custom register:

1. Locate the Connection Properties for the board/chip definition that you want to use to define your custom memory mapped registers:
  - a. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
  - b. Expand the (\*.bcd) Board/Chip Definitions group.
  - c. Expand the group for the BCD file. For example, expand the ... \MP3.bcd group.
2. Expand the following groups for the BCD file in turn:
  - a. BOARD=MP3
  - b. Advanced\_Information
  - c. ARM (or the corresponding name specified in your board/chip definition)
  - d. Register.
3. Rename the Default group under Register to the required name. For example, enter **Newreg**.

———— **Note** ————

This specifies the register symbol name, which you can reference using @name. For example, cexpression @Newreg.

4. Select the new register group, for example Newreg.
5. If you have defined an M\_name memory block that is associated with this register, then:
  - a. Right-click on Base to display the context menu.  
The context menu contains the names of any memory blocks that you have created (M\_name).
  - b. Select the appropriate memory block name (**M\_name**) from the context menu.



If you have not defined an associated memory map block for the register, then leave Base set to Absolute.

6. If you have set Base to a memory block name (*M\_name*), then set the value of Start to the offset of the register from the base address of the *M\_name* memory block. For example, **0x20**.  
If Base is set to Absolute, then set the value of Start to the absolute address of the register. For example, enter **0x10000020**.
7. Optionally, enter a descriptive name that is to be displayed for the register:
  - a. Right-click on *Gui\_name* to display the context menu.
  - b. Select **Edit Value...** from the context menu.
  - c. Enter the required display name for the register. For example, enter **New register**.
  - d. Press Enter to complete the entry.
8. Select **Save Changes** from the **File** menu to save the these changes.
9. If you want to define separate bit fields for a register, then set up the required bit fields.

#### See also

- *Creating a memory map block that relates to custom registers and peripherals* on page 4-32
- *Creating an enumeration for setting register values* on page 4-35
- *Defining a separate bit field for a custom register*
- *Register group settings* on page A-27.

### 4.12.2 Defining a separate bit field for a custom register

To define a separate bit field for a custom register:

1. Expand the required register group. For this example, expand the Newreg group.
2. Expand the Bit\_fields group. You are now going to set up four bit fields.
3. Create a bit field as required. For this example:
  - a. Right-click on the Bit\_fields group to display the context menu.
  - b. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - c. Enter **B\_name** for the name of the bit field. For example, enter **B\_Remap**.

#### ———— Note ————

This specifies the bit field symbol name, which you can reference using @B\_name.  
For example, cexpression @B\_Remap.

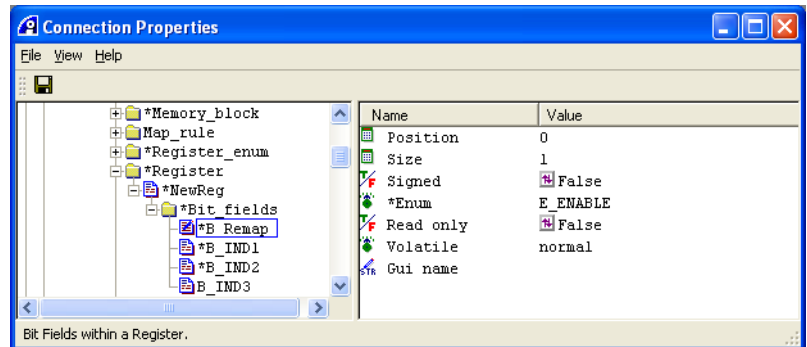
- d. Click **Create**. The new bit field is created (B\_Remap).
- e. Repeat these steps to create other bit fields as required. For example, create the bit fields B\_IND1, B\_IND2, and B\_IND3.

#### ———— Note ————

If you right-click on the last bit field you created, and select **Make New...** from the context menu, then RealView Debugger automatically assigns the next integer value to the name.

4. For each bit field you have created:
  - a. Select the bit field in the left pane (for example, B\_Remap).
  - b. Set the start Position of this bit field in the register (for example, enter 0).
  - c. Set the Size of this bit field in the register (for example, enter 1).
  - d. If you have defined any enumerations, then right-click on Enum and select the required enumeration (for example, E\_ENABLE).

Figure 4-22 shows an example:



**Figure 4-22 Creating bit field for registers**

5. Set the attributes for the remaining bit fields B\_IND1, B\_IND2, and B\_IND3 as shown in Table 4-2.

**Table 4-2 Additional bitfields**

Bit field name	Position setting	Size setting	Enum setting
B_IND1	1	1	E_SWITCH
B_IND2	2	4	not set
B_IND3	6	4	not set

6. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

### See also

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Register Bit\_fields group settings* on page A-28.

## 4.13 Creating a custom peripheral

A custom peripheral allows RealView Debugger to access the memory mapped peripherals on your development platform. A peripheral might be accessed using one or more registers.

Before you create a custom peripheral, be aware of the following:

- If any peripheral registers are to have enumerated values, you must create the required register enumerations.
- If your peripheral is to be associated with a memory map block you must first create the required memory map block.

See also:

- *Creating the custom peripheral*
- *Creating the registers for a custom peripheral* on page 4-41
- *Defining separate bit fields for a custom peripheral register* on page 4-42
- *Creating an enumeration for setting register values* on page 4-35
- *Creating a memory map block that relates to custom registers and peripherals* on page 4-32.

### 4.13.1 Creating the custom peripheral

To create a custom peripheral:

1. Locate the Connection Properties for the board/chip definition that you want to use to define your custom memory mapped registers:
  - a. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
  - b. Expand the (\*.bcd) Board/Chip Definitions group.
  - c. Expand the ...\*filename*.bcd group for the BCD file. For example, expand the ...\*MP3*.bcd group.
2. Expand the following groups for the board/chip definition in turn:
  - a. BOARD=*MP3*
  - b. Advanced\_Information
  - c. ARM (or the corresponding name specified in your board/chip definition)
  - d. Peripherals.
3. Rename the Default group under Peripherals to the name of your peripheral. For example, change the name to **SERBUS\_CTRL**.
4. Select the name of your peripheral group (SERBUS\_CTRL in this example).
5. Right-click on Base to display the context menu.
6. Select the required option from the context menu to specify the base address of the peripheral. The following options are available on the context menu:
  - The names of any memory map blocks that you have created. If the start address of your peripheral is to be relative to a memory map block, then you must first create that memory map block.

- **Absolute.** If you want to provide an absolute start address for the peripheral. This is the default.

---

**Note**

If you have not defined an associated memory map block for the peripheral, then leave Base set to Absolute. You must then specify the Start address in the next step as an absolute address. In this example, 0x10002000 is the absolute address of the SERBUS\_CTRL peripheral register.

---

7. Set the value of Start to **0x10002000**.
8. Enter a Description of your peripheral.  
For example, enter **I2C Controller**.
9. Select **Save Changes** from the **File** menu to save the these changes.
10. If your peripheral allows access to blocks of data, then specify the access method using the Access\_Method group settings.
11. Create any peripheral registers.

**See also**

- *Creating the registers for a custom peripheral*
- *Creating a memory map block that relates to custom registers and peripherals on page 4-32*
- *Peripherals group settings on page A-29.*

#### 4.13.2 Creating the registers for a custom peripheral

To create the registers for a custom peripheral (for example SERBUS\_CTRL):

1. Expand the peripheral group. For example, expand the SERBUS\_CTRL peripheral.
2. Expand the Register group.
3. Create a new peripheral register group (SERBUS\_CTRL\_SET in this example):
  - a. Right-click on the Default group to display the context menu.
  - b. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - c. Enter **SERBUS\_CTRL\_SET** for the name of the register.

---

**Note**

This specifies the register symbol name, which you can reference using @name. For example, cexpression @SERBUS\_CTRL\_SET.

---

  - d. Click **Create**. The new peripheral register group is created.
4. Set up the new peripheral register. For this example, do the following for the SERBUS\_CTRL\_SET register:
  - a. Select the SERBUS\_CTRL\_SET register.
  - b. Set Start to **0x10002000**.
  - c. Set Length to **4**.
  - d. Set Gui\_name to **Serial Bus Control**.

See *Register group settings* on page A-27 for more details of the settings in this group.

5. Repeat these steps for each peripheral register that you want to create.
6. Select **Save Changes** from the **File** menu to save these changes.
7. If you want to define separate bit fields for a register, then set up the required bit fields.

#### See also

- *Defining separate bit fields for a custom peripheral register*
- *Creating a memory map block that relates to custom registers and peripherals* on page 4-32
- *Peripherals group settings* on page A-29.

### 4.13.3 Defining separate bit fields for a custom peripheral register

To define separate bit fields for a custom peripheral register (SERBUS\_CTRL\_SET in this example):

1. Expand the required peripheral register group. For this example, expand the SERBUS\_CTRL\_SET group.
2. Expand the Bit\_fields group. You are now going to set up a bit field.
3. Create the required bit field (B\_SCLS for this example):
  - a. Right-click on the Default group to display the context menu.
  - b. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - c. Enter **B\_SCLS** for the name of the bit field.

**Note**

This specifies the bit field symbol name, which you can reference using @B\_name. For example, cexpression @B\_SCLS.

  - d. Click **Create** to create the B\_SCLS bit field.
  - e. Repeat these steps to create additional bit fields as required.
4. Select the new bit field (B\_SCLS) in the left pane and set up the required values. For example:
  - a. Set Position to **0** (this is the default).
  - b. Change Size to the required value. Leave the value set to 1, which is the default.
  - c. If you want to specify an enumeration for the bit field, you must have previously created the enumeration.  
For example, right-click on Enum, and select **E\_ENABLE** from the context menu.
  - d. If you want a specific name to be displayed for your bit field, modify the Gui\_name setting. Otherwise, the bit field name (B\_SCLS) is used. For example, change Gui\_name to **SCL Set**.

Figure 4-23 on page 4-43 shows an example:

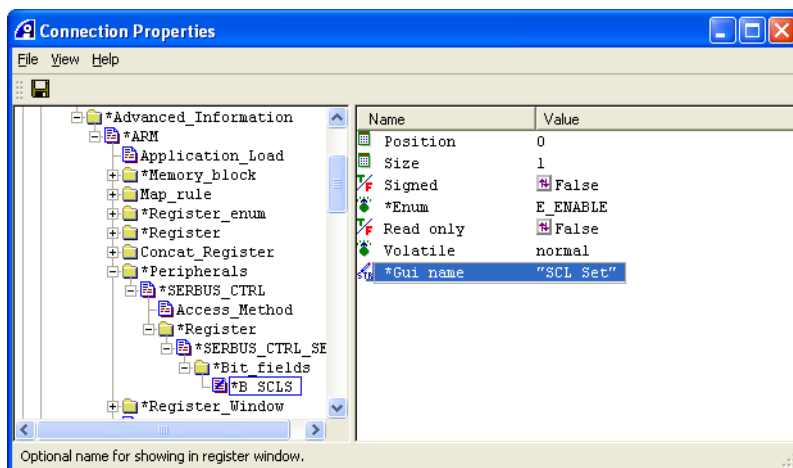


Figure 4-23 Creating bit fields for peripheral registers

5. Set up any additional bit fields you have created.
6. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

#### See also

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Creating a memory map block that relates to custom registers and peripherals* on page 4-32
- *Creating an enumeration for setting register values* on page 4-35
- *Register Bit\_fields group settings* on page A-28.

## 4.14 Creating the register tab for displaying custom registers and peripherals

Having created the custom registers and peripheral registers, you must have a way of displaying them in RealView Debugger, so that you can monitor or modify the values. You must define the visual appearance of the registers and peripheral registers, and specify the name of the tab that is to contain the registers. The tab is displayed in the Registers view.

To define the visual appearance of the custom registers and peripheral registers in the Registers view:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
2. Locate the Register\_Window group in the Advanced\_Information block group. Figure 4-24 shows an example:

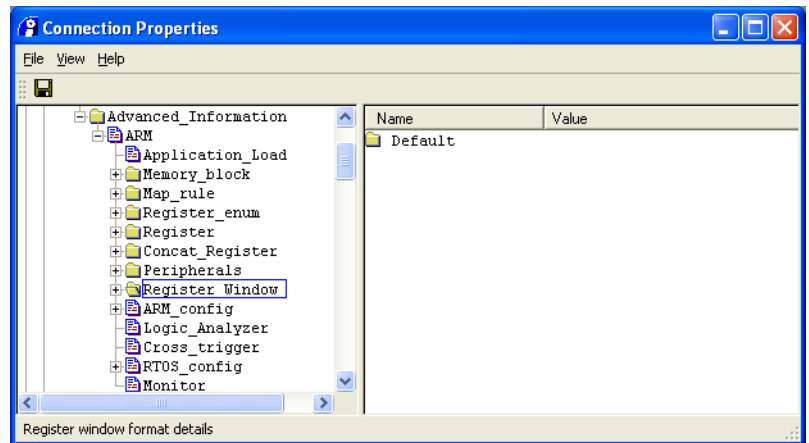


Figure 4-24 Register\_Window group

3. Expand the Register\_Window group in the left pane.
4. Rename the Default group under Register\_Window to **MP3\_REGS**.

———— **Note** ————

This specifies the name for the new register tab in the Registers view.

5. Select MP3\_REGS in the left pane.
6. Add the first Line setting:
  - a. Right-click on the Line setting to display the context menu.
  - b. Select **Edit Value** from the context menu.
  - c. Enter **\$+**.
  - d. Press Enter to complete the entry.

This makes the SERBUS\_CTRL\_SET peripheral register an expandable entry in the register tab.

———— **Note** ————

It is important in the following steps to right-click on the last \*Line you created when adding a new Line. If you right-click on the original Line setting, the lines are added in reverse order.

7. Create a line setting to display the SERBUS\_CTRL\_SET peripheral register:
  - a. Right-click on the \*Line "\$+" setting to display the context menu.

- b. Select **Make New...** from the context menu.
- c. Set the \*Line setting, to `=SERBUS_CTRL_SET`.
- d. Press Enter to complete the entry.

This displays the peripheral register name and the peripheral register on the same line in the register tab.

This displays the SERBUS\_CTRL\_SET peripheral register. The name displayed for the register is defined by the Gui\_name setting (for example, Serial Bus Control), and the name and peripheral register value are displayed on the same line in the register tab.

8. Create a line setting for any bit field names you have created. For example B\_SCLS, of the SERBUS\_CTRL\_SET peripheral register:
  - a. Right-click on the \*Line "`=SERBUS_CTRL_SET`" setting to display the context menu.
  - b. Select **Make New...** from the context menu.
  - c. Set the \*Line setting, to `B_SCLS`.
  - d. Press Enter to complete the entry.
9. Create a line to expand the SERBUS\_CTRL\_SET peripheral register:
  - a. Right-click on the \*Line "`B_SCLS`" setting to display the context menu.
  - b. Select **Make New...** from the context menu.
  - c. Set the \*Line setting, to `$+`.
  - d. Press Enter to complete the entry.
10. Create a line setting to display the Newreg register:
  - a. Right-click on the last \*Line "`$+`" setting to display the context menu.
  - b. Select **Make New...** from the context menu.
  - c. Set the \*Line setting, to `=Newreg`.
  - d. Press Enter to complete the entry.

The register name and the register are displayed on the same line in the register tab.

If you specified a descriptive name for the register in the Gui\_name setting (for example, New Register), then that name is displayed for the register. Otherwise, the register symbol name (Newreg) is displayed.

11. Create a line setting for the Newreg register to display the name INDICATORS for the register tab:
  - a. Right-click on the \*Line "`=Newreg`" setting to display the context menu.
  - b. Select **Make New...** from the context menu.
  - c. Set the \*Line setting, to `_INDICATORS`.  
Literals entered in \*Line (or Line) must be preceded by an underscore. The underscore is not displayed in the tab.
  - d. Press Enter to complete the entry.
12. Create a line setting for any bit field names you have created. For example B\_REMAP, B\_IND1, B\_IND2, and B\_IND3 of the Newreg register group:
  - a. Right-click on the \*Line "`_INDICATORS`" setting to display the context menu.
  - b. Select **Make New...** from the context menu.
  - c. Set the \*Line setting, to `B_Remap,B_IND1,B_IND2,B_IND3`.
  - d. Press Enter to complete the entry.

The Connection Properties window looks like Figure 4-25 on page 4-46.



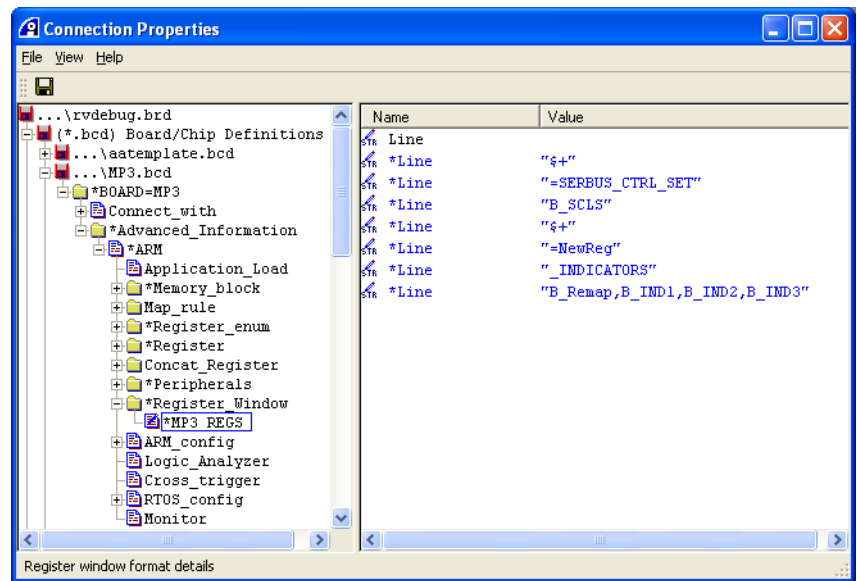


Figure 4-25 The MP3\_REGS group

All BCD file entries are now complete.

13. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

See also:

- *Viewing the custom registers and peripherals*
- *Register\_Window* on page A-30.

#### 4.14.1 Viewing the custom registers and peripherals

In the last stage, display the new registers and peripherals in the Registers view:

1. Connect to your target.
2. Select **Registers** from the **View** menu to display the Registers view.
3. Select the **MP3\_REGS** tab to view the custom registers. An example is shown in Figure 4-26.

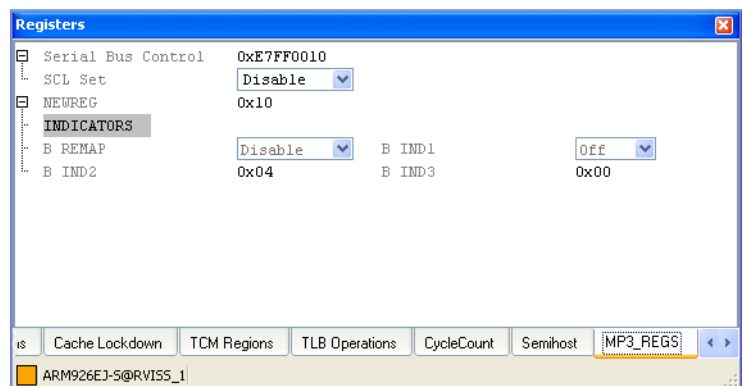


Figure 4-26 MP3\_REGS tab in the Registers view

## 4.15 Creating memory map rules

Memory map rules enable you to control how memory map blocks are displayed. Typically, map rules are required when multiple memory map blocks are defined that occupy the same area of memory, but only one of the blocks is to be active at a time. For example, when a target supports memory remapping. These memory map blocks are referred to as controlled memory map blocks.

The active memory map block is determined by examining the value of a custom register or peripheral register. There must be a map rule for each memory map block that is to be controlled by that register. Each map rule associates a unique register value with a specific memory map block.

Before you create map rules:

- Create the required memory map blocks.
- Either:
  - create a custom register
  - create a custom peripheral register.

To create the rules that control which memory block is used:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
2. Expand the (\*.bcd) Board/Chip Definitions group.
3. Expand ...\\MP3.bcd in the definitions list.
4. Expand the following groups in turn:
  - a. \*BOARD=MP3
  - b. \*Advanced\_Information
  - c. \*ARM
  - d. Map\_rule.
5. Select the Default group.
6. Rename the Default group of Map\_rule to **R\_FastRAM**.
7. Set the following values:
  - a. Set Register to **Newreg** (use the context menu).
  - b. Set Mask to **0x0001** (to check the value of the B\_REMAP bit field only).
  - c. Set Value to **0** (M\_FastRAM displayed when B\_REMAP bit field is set to Disable).
  - d. Set On\_equal to **M\_FastRAM**.

Figure 4-27 on page 4-48 shows an example:

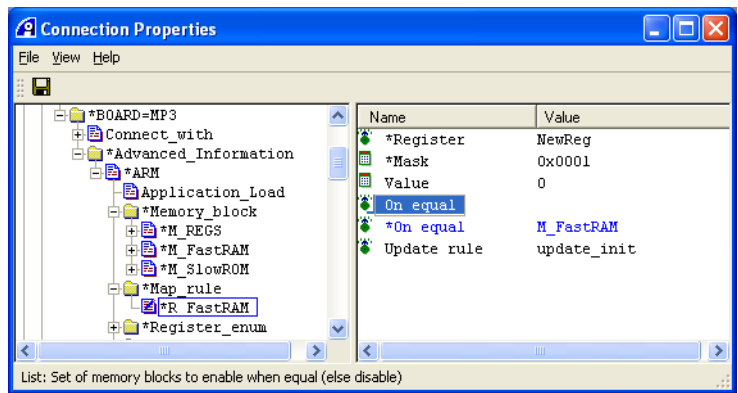


Figure 4-27 Settings for the R\_FastRAM map rule

8. Create the R\_SlowROM rule:
  - a. Right-click on R\_FastRAM to display the context menu.
  - b. Select **Make Copy...** from the context menu to display the Enter Name of New object dialog box. The name R\_FastRAM\_1 is inserted.
 

———— **Note** ————

If the name of a rule you are copying ends with a number, then RealView Debugger increments the number for the new rule.
  - c. Change the name to **R\_SlowROM**.
  - d. Click **Create**.
9. Select R\_SlowROM and set Value to **1**. That is, display the M\_SlowROM memory block when the B\_REMAP bit field is set to Enable.
10. Set the value of \*On equal to **M\_SlowROM**.

Figure 4-28 shows an example:

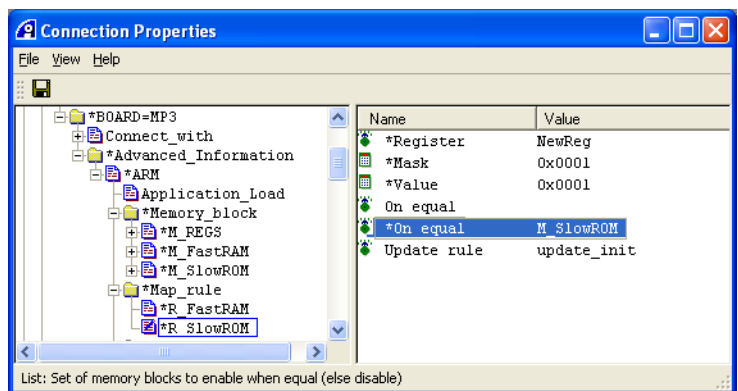


Figure 4-28 Settings for the R\_SlowROM map rule

All board file entries are now complete.

11. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

See also:

- *Creating a memory map block on page 4-29*
- *Creating a custom memory mapped register on page 4-37*
- *Creating a custom peripheral on page 4-40.*

## 4.16 Setting up controlled memory blocks

A *controlled memory map block* is one that is activated when a custom register or peripheral register has a specific value. That is, the memory map block is displayed in the **Memory Map** tab of the Process Control view. For example, an area of memory might be defined as RAM or ROM, depending on the value of a bit in a register. Therefore, you might have a memory map block that defines the area as RAM, and one that defines the same area as ROM. The value of the bit in the register determines whether to display the area as RAM or ROM.

To determine which memory map block is active you must specify a map rule for each controlled memory map block. Each map rule associates a unique register value with a specific memory map block.

---

### Note

---

This feature is supported only on ARM processors that support memory remapping, such as the ARM926EJ-S.

---

See also:

- *Before you set up controlled memory blocks*
- *Defining the controlled memory map blocks*
- *Defining the memory rules on page 4-51*
- *Displaying the controlled memory map blocks on page 4-52*

### 4.16.1 Before you set up controlled memory blocks

Before you start, create a custom register or peripheral register that you want to use to activate the controlled memory map blocks.

This example describes how to:

- Set up two memory map blocks that are activated at different times according to the value of a register. It uses the Newreg register created in *Creating a custom memory mapped register* on page 4-37. This is displayed in the **MP3\_REGS** tab in the Registers view.
- Set a memory rule to specify how the memory is used. When Remap is set to Enable, M\_SlowROM is activated. Otherwise, M\_FastRAM is used.

Perform the steps described in the following sections:

1. *Defining the controlled memory map blocks*
2. *Defining the memory rules on page 4-51*
3. *Displaying the controlled memory map blocks on page 4-52.*

### See also

- *Creating a memory map block on page 4-29*
- *Creating a custom memory mapped register on page 4-37*
- *Creating a custom peripheral on page 4-40.*

### 4.16.2 Defining the controlled memory map blocks

Define two controlled memory map blocks named M\_FastRAM and M\_SlowROM:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.

2. Locate the RVISS\_1 Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the required Debug Interface. For this example, expand RealView Instruction Set Simulator (RVISS).
  - c. Make sure that all targets are disconnected on the Debug Configuration.
3. Right-click on the RVISS\_1 Debug Configuration to display the context menu.
4. Select **Properties...** from the context menu to display the Connection Properties dialog box.
5. Click the **Advanced** button to display the Connection Properties window.
6. Locate the board/chip definition where you want to define the custom memory mapped registers:
  - a. Right-click on the \*BoardChip\_name MP3 setting in the right pane to display the context menu.
  - b. Select **Jump to Definition** from the context menu.  
The BOARD=MP3 group is selected in the BCD file.
7. Expand the following board/chip definition groups in turn:
  - a. \*BOARD=MP3
  - b. \*Advanced\_Information
  - c. \*ARM (or the corresponding name specified in your board/chip definition)
  - d. \*Memory\_Block.
8. Right-click on the \*M\_REGS group in the left pane to display the context menu.
9. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name M\_REGS\_1 is inserted.
10. Enter a new name for the group. For example, enter **M\_FastRAM**.
11. Click **Create**.
12. Select the M\_FastRAM group, in the left pane.
13. Set these values for the M\_FastRAM group:
  - a. Set the value of Start to the start address of the memory block, for example, **0x0** (this is the default).
  - b. Set the value of Length in memory units for the memory block. For example, **0x80000** bytes for an ARM architecture-based processor.
  - c. Set Description to **Fast Static RAM**.
  - d. Set Access to **RAM** (this is the default).
14. Right-click on M\_FastRAM to display the context menu.
15. Select **Make Copy...** from the context menu to display the Enter Name of New object dialog box. The name M\_FastRAM\_1 is inserted.
16. Enter a new name for this group. For example, enter **M\_SlowROM**.
17. Click **Create**.
18. Select the M\_SlowROM group in the left pane to display the settings values:
  - a. Set Access to **ROM**.

- b. Set \*Description to **Slow Boot ROM**.
19. Select **Save Changes** from the **File** menu to save the these changes.

#### See also

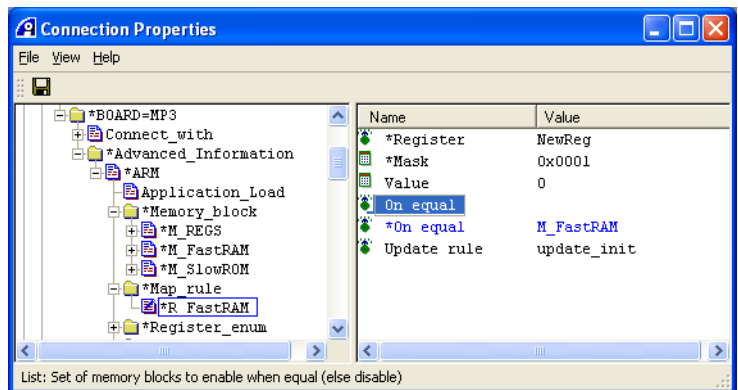
- *Suggested naming convention for memory map related settings groups* on page 4-5.

### 4.16.3 Defining the memory rules

Define the rules that control which memory map block is used:

1. Expand the Map\_rule group.  
The map rule defines which memory block to use. In this example, M\_SlowROM is activated if B\_Remap is set to Enabled (one). Otherwise M\_FastRAM is used.
2. Rename the Default group of Map\_rule to **R\_FastRAM**.
3. Set the following values:
  - a. Set Register to **Newreg** (use the context menu).
  - b. Set Mask to **0x0001** (to check the value of the B\_REMAP bit field only).
  - c. Set Value to **0** (M\_FastRAM displayed when B\_REMAP bit field is set to Disabled).
  - d. Set On\_equal to **M\_FastRAM**.

Figure 4-29 shows an example:



**Figure 4-29 Settings for the R\_FastRAM map rule**

4. Create the R\_SlowROM rule:
  - a. Right-click on R\_FastRAM to display the context menu.
  - b. Select **Make Copy...** from the context menu to display the Enter Name of New object dialog box. The name R\_FastRAM\_1 is inserted.  

———— **Note** ————

If the name of a rule you are copying ends with a number, then RealView Debugger increments the number for the new rule.
  - c. Change the name to **R\_SlowROM**.
  - d. Click **Create**.
5. Select R\_SlowROM and set Value to **1**. That is, display the M\_SlowROM memory block when the B\_Remap bit field is set to Enabled.
6. Set the value of \*On\_equal to **M\_SlowROM**. Figure 4-30 on page 4-52 shows an example:

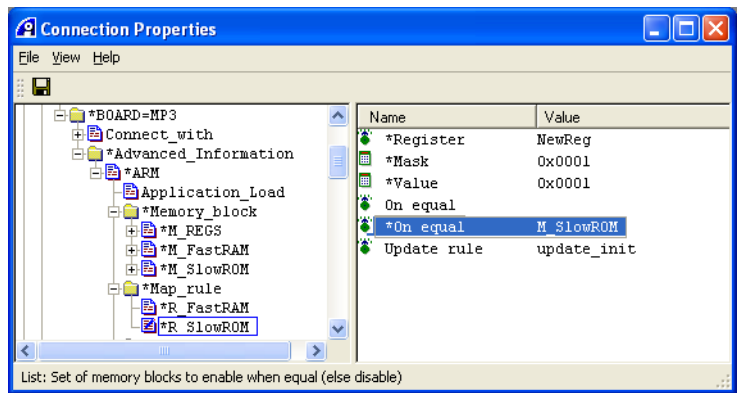


Figure 4-30 Settings for the R\_SlowROM map rule

All board file entries are now complete.

7. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
8. Click the **OK** button to close the Connection Properties dialog box.

#### 4.16.4 Displaying the controlled memory map blocks

To display the controlled memory map blocks:

1. Connect to your target.
2. Select **Registers** from the **View** menu.  
A floating Registers view is displayed.
3. Select the **MP3\_REGS** tab to view the custom registers. Figure 4-31 shows an example:

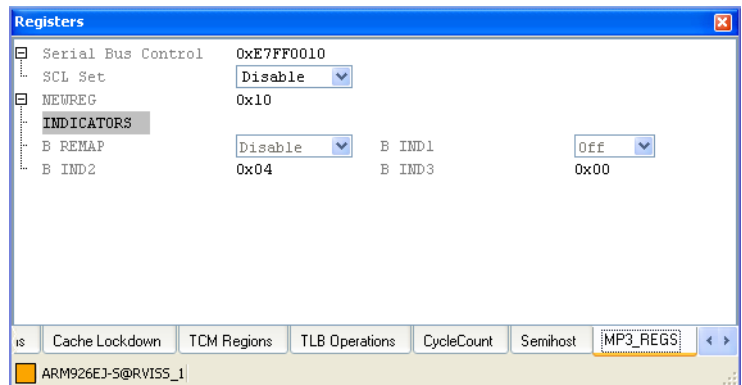


Figure 4-31 MP3\_REGS tab in the Registers view

4. Select **Memory Map Tab** from the **View** menu to display the Process Control view with the **Memory Map** tab selected. Figure 4-32 on page 4-53 shows an example:

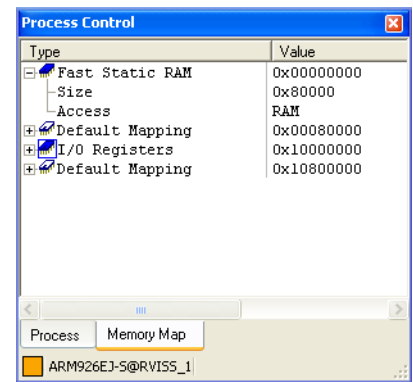


Figure 4-32 New memory block in the Memory Map tab

- In the Registers view, set the B\_REMAP value to Enable to activate the memory rule (R\_SlowROM).

The first entry in the **Memory Map** tab changes to ROM memory, indicated by a yellow icon. Figure 4-33 shows an example:

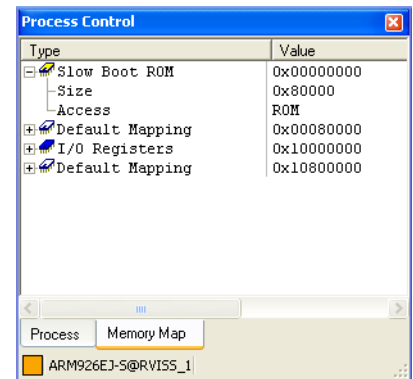


Figure 4-33 Activated ROM memory block in the Memory Map tab



## 4.17 Creating a concatenated register

The example used in this procedure creates a concatenated register `C_R8_R9_concat` for the RVISS Debug Configuration as follows:

- the high 16 bits of register R8 forms the low 16 bits of `R8_R9_concat`
- the low 16 bits of register R9 forms the high 16 bits of `R8_R9_concat`.

The example uses the blank BCD file created in *Creating a BCD file to use as a template* on page 4-13.

If you want to use an enumeration to set the values of your concatenated register, then you must create it before you create the register.

To create a concatenated register:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the RVISS\_1 Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the required Debug Interface. For this example, expand RealView Instruction Set Simulator (RVISS).
  - c. Expand the RVISS Debug Configuration.
  - d. Make sure that the ARM926EJ-S target is disconnected on the Debug Configuration.
3. Right-click on the RVISS\_1 Debug Configuration to display the context menu.
4. Select **Properties...** from the context menu to display the Connection Properties dialog box.
5. Click the **Advanced** button to display the Connection Properties window. The settings group for the Debug Configuration is selected (in this example, `CONNECTION=RVISS_1`).

———— **Note** ————

If you have followed the examples in previous sections, then you might already have the MP3 BoardChip definition assigned. If so, then skip the next step.

6. Assign the required BoardChip definition to the connection:
  - a. Right-click on the `BoardChip_name` setting in the right pane to display the context menu.
  - b. Select **MP3** from the context menu.
7. Locate the board/chip definition where you want to define the custom memory mapped registers:
  - a. Right-click on the `*BoardChip_name MP3` setting in the right pane to display the context menu.
  - b. Select **Jump to Definition** from the context menu.  
The `BOARD=MP3` group is selected in the BCD file.
8. Expand the following board/chip definition groups in turn:
  - a. `*BOARD=MP3`
  - b. `*Advanced_Information`
  - c. `*ARM` (or the corresponding name specified in your board/chip definition)
  - d. `Concat_Register`

9. Create the concatenated register as follows:
  - a. Rename the Default block to **C\_R8\_R9\_concat**.
 

———— **Note** ————

This specifies the concatenated register symbol name, which you can reference using *@name*. For example, cexpression *@C\_R8\_R9\_concat*.
  - b. Double-click on C\_R8\_R9\_concat in the right pane.  
The settings for the concatenated register are displayed.
  - c. Change the C\_R8\_R9\_concat settings to the values shown in Table 4-3. If you enter the Low\_mask and High\_mask values in decimal, then they are displayed as hexadecimal.

**Table 4-3 C\_R8\_R9\_concat settings**

Setting	Value
Low_name	<b>R8</b>
Low_shift	<b>16</b>
Low_mask	<b>0xFFFF</b>
High_name	<b>R9</b>
High_shift	<b>-16</b> (left shift)
High_mask	<b>0xFFFF0000</b>
Gui_name	<b>R8 and R9 combined</b>

———— **Note** ————

If you have followed the examples in previous sections, then you might already have the MP3\_REGS block in the Register\_Window group. If so, then use the MP3\_REGS block instead of Default for the next step.

10. Create the Registers view tab to view the concatenated register:
  - a. Double-click on the Register\_Window group in the left pane.
  - b. Rename the Default block to **Concat\_regs**.
  - c. Select Concat\_regs to display the related settings.
  - d. Right-click on Line to display the context menu.
  - e. Select **Edit Value...** from the context menu.
  - f. Enter **C\_R8\_R9\_concat**. Remember to press Enter to complete the entry.
11. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
12. Click the **OK** button to close the Connection Properties dialog box.

See also:

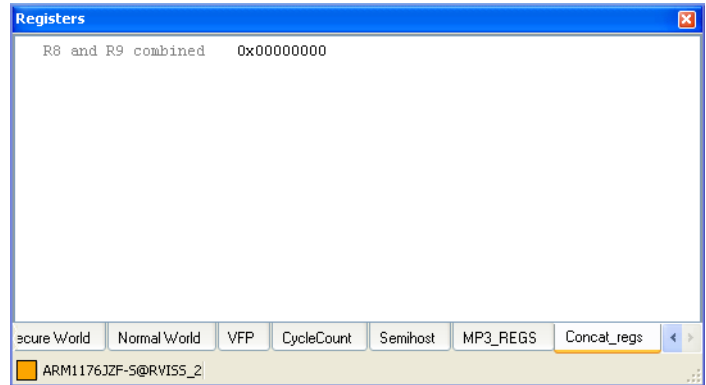
- *Viewing the concatenated register* on page 4-56
- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Creating a BCD file to use as a template* on page 4-13

- *Creating an enumeration for setting register values* on page 4-35
- *Concat\_Register* on page A-28.

#### 4.17.1 Viewing the concatenated register

To view the concatenated register created in *Creating a concatenated register* on page 4-54:

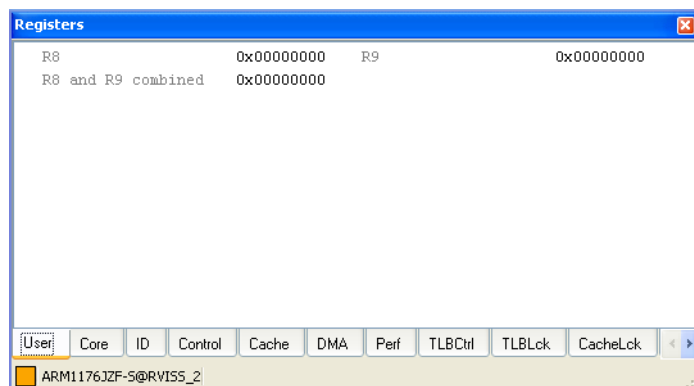
1. Connect to the target containing the concatenated register (ARM926EJ-S in this example).
2. Select **Registers** from the **View** menu to display the Registers view. Figure 4-34 shows an example:



**Figure 4-34 Concat\_reg tab in Registers view**

3. Copy the R8 and R9 registers to the User view:
  - a. Press the Ctrl key, then left click on the R8 and R9 registers to select them.
  - b. Right-click on one of the selected registers to display the context menu.
  - c. Select **Copy to User View** from the context menu.
4. Copy the concatenated register to the User view:
  - a. Click the **Concat\_regs** tab.
  - b. Right-click on the C\_R8\_R9\_concat concatenated register to display the context menu.
  - c. Select **Copy to User View** from the context menu.
5. Display the User view:
  - a. Right-click in the view to display the context menu.
  - b. Select **Show User View** from the context menu.

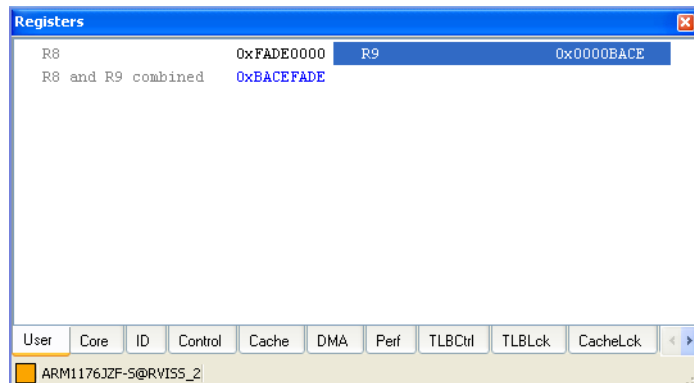
The **User** tab is displayed. Figure 4-35 on page 4-57 shows an example:



**Figure 4-35 User view showing concatenated register**

- Change the values of the R8 and R9 registers to see the value of the concatenated register change. For example, set R8 to 0xFADE0000 and R9 to 0x0000BACE.

Figure 4-36 shows an example:



**Figure 4-36 User view showing concatenated register values**

## 4.18 Troubleshooting BCD files

The following sections describe how to solve some problems you might encounter with BCD files:

- *Expected memory map is not displayed*
- *Expected register tab does not appear in the Registers view*
- *Duplicate definition warning messages on connecting*
- *Overlapping un-controlled memory region warning messages on connecting* on page 4-59
- *Map-Rule and Undefined register warnings* on page 4-61

### 4.18.1 Expected memory map is not displayed

Check the following:

- If you have assigned a board/chip definition to a Debug Configuration, and you do not see the expected memory map in the **Memory Map** tab of the Process Control view, then make sure you have assigned the correct board/chip definition to the Debug Configuration. You can assign more than one board/chip definition to a Debug Configuration if required. The settings in a board/chip definition are used only if the Advanced\_Information block group name matches the target to which you are connecting.
- If you have assigned the correct board/chip definition to a Debug Configuration, check the names of the Advanced\_Information block groups in the board/chip definition. Groups with names Default or All are used for all target connections in a Debug Configuration. An Advanced\_Information block group name must reflect the target to which you are connecting. For example, a group named ARM is used when you connect to any ARM architecture-based target.

#### See also

- *The board/chip definition Advanced\_Information block* on page 4-3
- *Memory mapping Advanced\_Information settings reference* on page A-20.

### 4.18.2 Expected register tab does not appear in the Registers view

Check that the name of the tab defined in the Register\_window group is not the same as a tab that already exists.

#### See also

- *Expected memory map is not displayed* for other related problems
- *The board/chip definition Advanced\_Information block* on page 4-3
- *Register\_Window* on page A-30.

### 4.18.3 Duplicate definition warning messages on connecting

If you have assigned multiple board/chip definitions to a Debug Configuration, then when you connect to a target you might see warning message of the form:

Warning: Duplicate entity definition 'entity\_symbol'

RealView Debugger uses the settings in the first board/chip definition that it finds. This is not a problem if both entities have exactly the same definition.

### Example

With the CM926EJ-S and CP board/chip definitions assigned to a Debug Configuration, warning messages similar to the following are displayed when you connect to an ARM926EJ-S target:

Warning: Duplicate register enumeration definition 'E\_ENABLE'

However, the definition of each entity is the same in all the supplied ARM related board/chip definitions, so this does not cause any problems.

### See also

- *Avoiding conflicts between linked board groups* on page 3-14.

#### 4.18.4 Overlapping un-controlled memory region warning messages on connecting

When you connect to a target you might see warning messages of the form:

Warning: Overlapping un-controlled memory region '*region\_1*' and '*region\_2*'

Also, this message is accompanied by one or more instances of the following message:

Warning: Memory map overlaps with existing mapping(s).

These messages are displayed for the following reasons:

- you have assigned multiple board/chip definitions to a Debug Configuration, and each board/chip definition defines memory areas that overlap
- you have defined multiple memory areas in the same board/chip definition that overlap.

Memory areas are uncontrolled when they are not associated with a memory map rule. This is the default configuration. If you have defined multiple memory areas that occupy the same area of memory, but only one is to be active at a time, then the memory areas must be controlled using map rules.

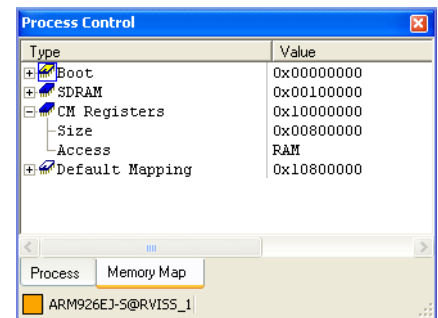
Uncontrolled memory map areas that overlap is not usually a problem if one board/chip definition defines a generic memory region, and another defines one or more specific memory regions. RealView Debugger creates a combined memory region by overlaying the generic region with the specific memory regions.

### Example

With the CM926EJ-S and CP board/chip definitions assigned to a Debug Configuration, warning messages similar to the following are displayed when you connect to an ARM926EJ-S target:

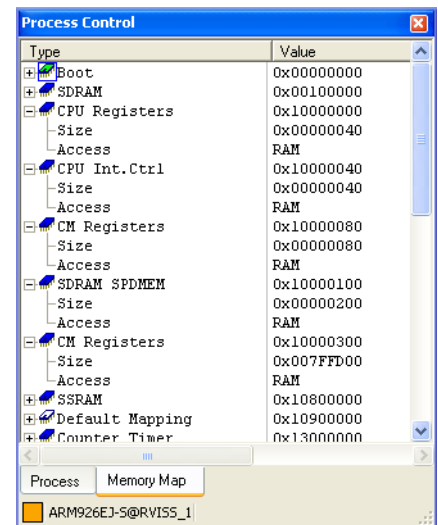
Warning: Overlapping un-controlled memory region 'M\_REGS (I/O Registers)' and 'M\_CM\_Regs (CM Registers)'

In this example, Figure 4-37 on page 4-60 shows the memory map for the ARM926EJ-S target.



**Figure 4-37** Memory map defined in the CM926EJ-S.bcd file

Figure 4-38 shows the combined memory from both the CM926EJ-S and the CP board/chip definition.



**Figure 4-38** Combined memory map for CM926EJ-S and CP board/chip definitions

In Figure 4-38 you see that the memory regions CPU Int.Ctrl and SDRAM SPDME from the CP board/chip definition overlay the CM Registers memory region from the CM926EJ-S board/chip definition.

### See also

- *Creating memory map rules* on page 4-47
- *Map\_rule group settings* on page A-25.

#### 4.18.5 Map-Rule and Undefined register warnings

If you have assigned multiple board/chip definitions to a Debug Configuration, then when you connect to a target you might see warning message of the form:

```
Warning: Map-Rule 'rule': register not found: 'register'
Warning: Undefined register 'register'
```

The following summary message is also displayed, to indicate the number of memory rules that are affected:

```
Warning: Failed on Advanced info: MemoryRules=n
```

These are usually displayed when a map rule references a register that is defined in another board/chip definition, but the referenced board/chip definition is not assigned to the Debug Configuration.

#### Example

With the CP board/chip definition assigned to a Debug Configuration, warning messages similar to the following are displayed when you connect to an ARM target (for example, an ARM926EJ-S target):

```
Warning: Map-Rule 'R_nREMAP': register not found: 'B_REMAP_A'
Warning: Undefined register 'B_REMAP_A'
Warning: Map-Rule 'R_REMAP': register not found: 'B_REMAP_A'
Warning: Undefined register 'B_REMAP_A'
Warning: Failed on Advanced info: MemoryRules=2
```

You must assign the board/chip definition for the corresponding target to the Debug Configuration (the CM926EJ-S definition in this example).

#### See also

- *Assigning one board group to a Debug Configuration* on page 4-8
- *Creating memory map rules* on page 4-47
- *Map\_rule group settings* on page A-25.



# Chapter 5

## Debug Configuration Tutorial

This chapter provides a tutorial on how to create a Debug Configuration and BCD files that define the memory map related elements for a basic development platform. You are recommended to follow this tutorial before creating your own BCD files. It includes:

- *About the Debug Configuration tutorial on page 5-2*
- *Before starting the tutorial on page 5-5*
- *Creating a new Debug Configuration on page 5-6*
- *Configuring the new Debug Configuration on page 5-7*
- *Creating the EtherRouter.bcd file on page 5-8*
- *Creating the AMDLANCE.bcd file on page 5-10*
- *Creating the EtherRouter BOARD group on page 5-11*
- *Creating the AMDLANCE CHIP group on page 5-12*
- *Assigning board/chip definitions on page 5-13*
- *Creating the memory map on page 5-15*
- *Creating the enumerations for the register values on page 5-19*
- *Creating a custom register on page 5-21*
- *Creating the register tab for displaying custom registers on page 5-24*
- *Setting up controlled memory map blocks on page 5-27*
- *Displaying the controlled memory map blocks on page 5-31*
- *Creating memory map rules on page 5-29*
- *Creating a concatenated register on page 5-33.*

## 5.1 About the Debug Configuration tutorial

This tutorial shows how to set up the Debug Configuration for a simple Ethernet router. The Debug Configuration references a BCD file that:

- defines the memory map, registers, and peripherals
- references chip definitions in other BCD files.

Although the tutorial uses the RealView Instruction Set Simulator (RVISS) Debug Interface, the procedures described can be performed for any Debug Interface.

The tutorial in this chapter modifies the board file stored in your RealView Debugger home directory. By default, this is called `rvdebug.brd`. Debug Interface configuration files might also be stored in this directory, for example `.rvc` files or `.smc` files.

It is recommended that you back up this directory before starting the examples, so that you can restore your original configuration later.

### ———— Note ————

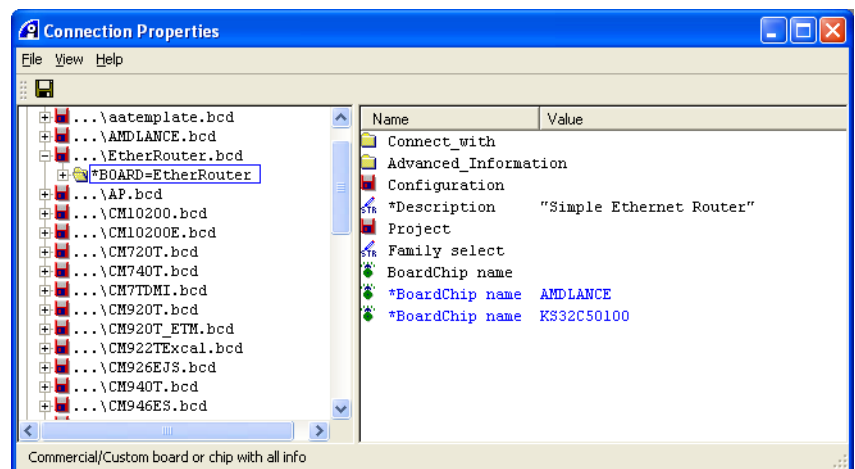
The example used in this tutorial is to demonstrate the features available for creating custom memory maps, registers, and peripherals. It is not intended to be a real example.

See also:

- *About the development platform used in this tutorial*
- *Saving and restoring connection properties* on page 1-17
- *Restoring the default connections and configurations* on page 3-55
- *Troubleshooting Debug Configurations* on page 3-65.

### 5.1.1 About the development platform used in this tutorial

The development platform used in this tutorial is a simple Ethernet router. The router uses a KS32C50100 for one network interface and an AMD LANCE for the second network interface. This configuration is shown in Figure 5-1.



**Figure 5-1 Linked groups for the EtherRouter board**

This is shown in tree from in Figure 5-2 on page 5-3.

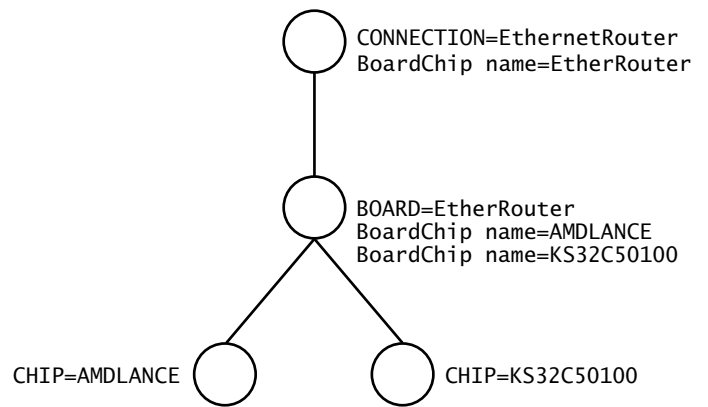


Figure 5-2 Tree view of the linked groups

**Note**

You are not required to split your board up into distinct CHIP descriptions. You can create one BOARD description containing all of the required information. However, splitting your board up into distinct CHIP descriptions enables you to reuse them for another development project.

**About the configuration used in this tutorial**

The example configuration used in this tutorial demonstrates:

- How to create the following entities:
  - a *RealView ARMulator*<sup>®</sup> ISS (RVISS) Debug Configuration called EthernetRouter
  - an AMDLANCE.bcd file, containing AMDLANCE chip definition
  - an ethernet.bcd file, containing references to the AMDLANCE and KS32C50100 chip definitions.
- How to define two memory regions:
  - M\_IO\_REGS that occupies the address range 0x20000000 to 0x207FFFFF
  - M\_I2C that occupies the address range 0x10002000 to 0x10002FFF.
- How to define enumerations that define meaningful names to appropriate register values:
  - E\_ENABLE, which defines enumerations for Off and On
  - E\_REMAP, which defines enumerations for RAM and ROM
  - E\_SWITCH, which defines enumerations for Disable and Enable.
- How to define a custom register that has the following attributes:
  - The name MemoryStatus.
  - An offset of 0x20 from the base of the memory region.
  - Four bit fields that are used as indicators of the state of the register and are named INDICATORS. The bit fields are labeled B\_REMAP, B\_LED\_SWITCH, B\_IND1, and B\_IND2.
- How to define a serial bus control peripheral SERBUS\_CTRL that is accessed using two registers:
  - SERBUS\_CTRL\_CLR containing the bits B\_SCLC and B\_SDAC
  - SERBUS\_CTRL\_SET containing the bits B\_SCLS and B\_SDAS.

- How to define two memory blocks M\_FastRAM and M\_SlowROM that are activated at different times according to the value of a register:
  - The MemoryStatus register determines which memory block to activate.
  - Memory rules R\_FastRAM and R\_SlowROM to specify how the memory is used. When B\_REMAP is set to ROM, M\_SlowROM is activated. Otherwise, M\_FastRAM is used.
- How to define a concatenated register C\_R8\_R9\_concat that is created from the core registers R8 and R9.
- How to define a tab in the Registers view, called EtherRouter, to display the custom registers and peripherals.

**See also**

- *Before starting the tutorial* on page 5-5.

## 5.2 Before starting the tutorial

Before you start the tutorial:

1. Disconnect all target connections, if any are established.
2. Create a template BCD file called `aatemplate.bcd`.

See also:

- *Creating a new Debug Configuration* on page 5-6
- *Creating a BCD file to use as a template* on page 4-13.

### 5.3 Creating a new Debug Configuration

Create a new Debug Configuration when you do not want to change the default Debug Configurations.

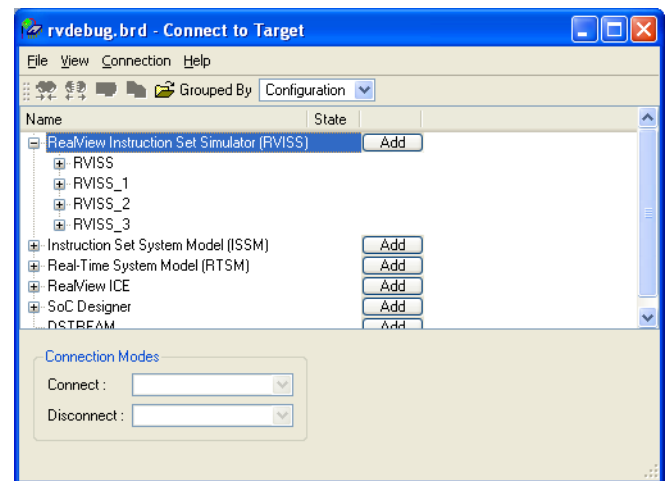
To create a new Debug Configuration:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Locate the required Debug Configuration:
  - a. Select **Configuration** from the Grouped By list.
  - b. Expand the RealView Instruction Set Simulator (RVISS) Debug Interface.
3. Click **Add** for the RealView Instruction Set Simulator (RVISS) Debug Interface. The ARMulator Configuration dialog box is displayed.

**———— Note ————**

If you are using a different Debug Interface, then a configuration dialog box appropriate to that interface is displayed.

4. Select **ARM926EJ-S** for the processor.
5. Click **OK**. The ARMulator Configuration dialog box closes, and a new Debug Configuration is added, called RVISS\_3 as shown in Figure 5-3.



**Figure 5-3 RVISS\_3 Debug Configuration in the Connect to Target window**

6. Right-click on the new Debug Configuration (RVISS\_3) to display the context menu.
7. Select **Rename Configuration** from the context menu.
8. Change the name of the Debug Configuration to **EthernetRouter**.

See also:

- *Configuring the new Debug Configuration* on page 5-7
- *Chapter 2 Customizing a Debug Interface configuration*
- the following in the *RealView Debugger User Guide*:
  - *Changing the name of a Debug Configuration* on page 3-17.

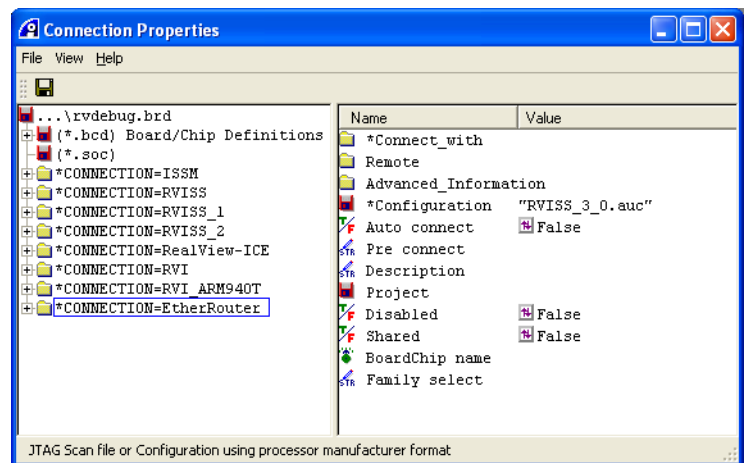
## 5.4 Configuring the new Debug Configuration

Configure the new Debug Configuration to reflect the development platform you are debugging.

You must now configure the new Debug Configuration:

1. Right-click on the EthernetRouter Debug Configuration to display the context menu.
2. Select **Properties...** from the context menu to display the Connection Properties dialog box.
3. Click the **Advanced** button to display the Connection Properties window. The new configuration group is selected. Figure 5-4 shows an example.

The file name in your Configuration setting might be different to that shown.



**Figure 5-4 Connection Properties for the EtherRouter Debug Configuration**

4. Right-click on Description in the right pane to display the context menu.
5. Select **Edit Value...** from the context menu.
6. Enter the description **Simple Ethernet Router**.
7. Select **Save Changes** from the **File** menu to save your changes.

See also:

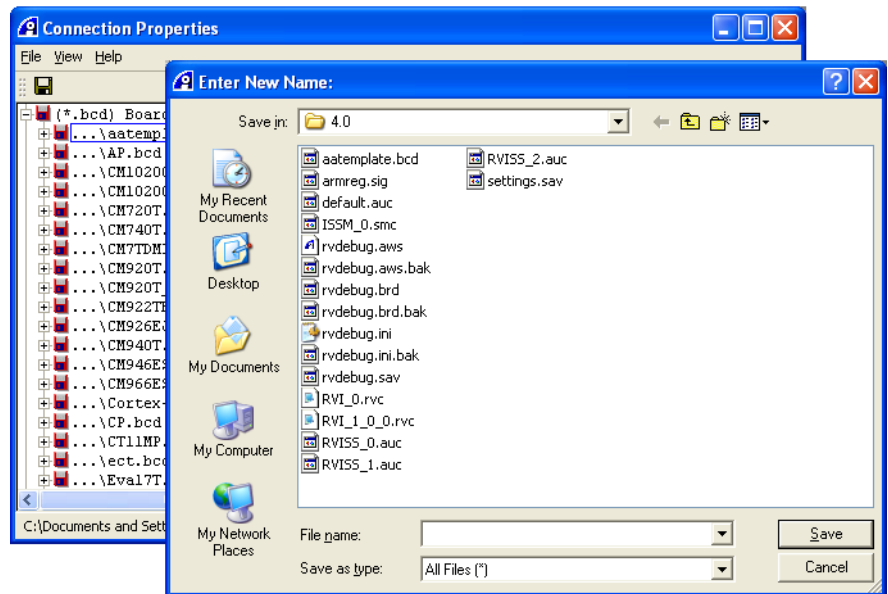
- *Creating the EtherRouter.bcd file* on page 5-8.

## 5.5 Creating the EtherRouter.bcd file

Create the EtherRouter.bcd file to define the memory mappings.

To create the EtherRouter.bcd file:

1. Expand the group (\*.bcd) Board/Chip Definitions to display the current list of target descriptions in the left pane.
2. Right-click on the ...\aatemplate.bcd entry.
3. Select **Save As...** from the context menu to display the Enter New Name dialog box. Figure 5-5 shows an example. The location displayed in this dialog box is the directory you last selected in the dialog box.



**Figure 5-5 Saving an existing BCD file with a new name**

By default, RealView Debugger searches for \*.bcd files in the current working directory, then in your home directory, and then in your default settings directory:

```
C:\Documents and Settings\userID\Local Settings\Application
Data\ARM\rvdebug\version\shadowbase\etc
```

In this example, save the new file in your RealView Debugger home directory.

4. Enter the new filename. You must use the .bcd file extension when saving the file in your home directory.

For this example, enter **EtherRouter.bcd**.

5. Click **Save**.

The New Name dialog box closes and the new name is displayed in the \*.bcd list. Although the new BCD entry replaces the BCD entry you used to make the copy, the original file still exists.

6. Refresh the list of BCD files:
  - a. Select **Save Changes** from the **File** menu to save the changes.
  - b. Select **Refresh** from the **File** menu. This refreshes the list of all current BCD files.
  - c. Expand the group (\*.bcd) Board/Chip Definitions to display the current list of BCD files. The list includes the new BCD file, that is EtherRouter.bcd.



See also:

- *Creating the AMDLANCE.bcd file* on page 5-10
- the following in the *RealView Debugger User Guide*:
  - *Redefining the RealView Debugger directories* on page 2-9.

## 5.6 Creating the AMDLANCE.bcd file

To create the AMDLANCE.bcd file:

1. Expand the group (\*.bcd) Board/Chip Definitions to display the current list of target descriptions in the left pane.
2. Right-click on the ...\aatemplate.bcd entry.
3. Select **Save As...** from the context menu to display the Enter New Name dialog.
4. Enter the filename **AMDLANCE.bcd**.
5. Click **Save**.
6. Refresh the list of BCD files:
  - a. Select **Save Changes** from the **File** menu to save the changes.
  - b. Select **Refresh** from the **File** menu. This refreshes the list of all current BCD files.
  - c. Expand the group (\*.bcd) Board/Chip Definitions to display the current list of BCD files. The list includes the new AMDLANCE.bcd file.

See also:

- *Creating the EtherRouter BOARD group* on page 5-11.

## 5.7 Creating the EtherRouter BOARD group

To create the EtherRouter group in the EtherRouter.bcd file:

1. Expand the ... \EtherRouter.bcd entry.
2. Right-click on the \*BOARD=ARM\_TEMPLATE group to display the context menu.
3. Select **Rename** from the context menu to display the Group Type/Name selector dialog box.
4. In the Group Name field change the name to **EtherRouter**.

———— **Note** ————

Only alphanumeric characters, underscore \_, and dash - are allowed.

5. Click **OK** to rename the group.
6. Save your changes.

See also:

- *Creating the AMDLANCE CHIP group* on page 5-12.

## 5.8 Creating the AMDLANCE CHIP group

To create the AMDLANCE group in the AMDLANCE.bcd file:

1. Expand the ... \AMDLANCE.bcd entry.
2. Right-click on ... \AMDLANCE.bcd to display the context menu.
3. Select **Make New Group...** from the context menu to display the Group Type/Name selector dialog box.
4. Select **CHIP** for the type of group.
5. In the Group Name field change the name to **AMDLANCE**.
6. Click **OK**. The CHIP=AMDLANCE group is added.
7. Delete the BOARD=ARM\_TEMPLATE group:
  - a. Right-click on the BOARD=ARM\_TEMPLATE group to display the context menu.
  - b. Select **Delete** from the context menu. The group is deleted.
8. Save your changes.

See also:

- *Assigning board/chip definitions* on page 5-13.

## 5.9 Assigning board/chip definitions

To apply the settings in one or more BCD groups to a Debug Configuration, the BCD files containing those groups must be in a location where RealView Debugger can find the files. Typically, if you create your own BCD files, then place them in your RealView Debugger home directory.

See also:

- *Assigning the EtherRouter board/chip definition to a Debug Configuration*
- *Assigning the AMDLANCE and KS32C50100 chips to the EtherRouter board on page 5-14*
- *Creating the memory map on page 5-15.*

### 5.9.1 Assigning the EtherRouter board/chip definition to a Debug Configuration

To assign the EtherRouter board/chip definition to a Debug Configuration:

1. In the Connection Properties window, select CONNECTION=EthernetRouter in the left pane to display the settings for the EthernetRouter Debug Configuration, shown in Figure 5-6.

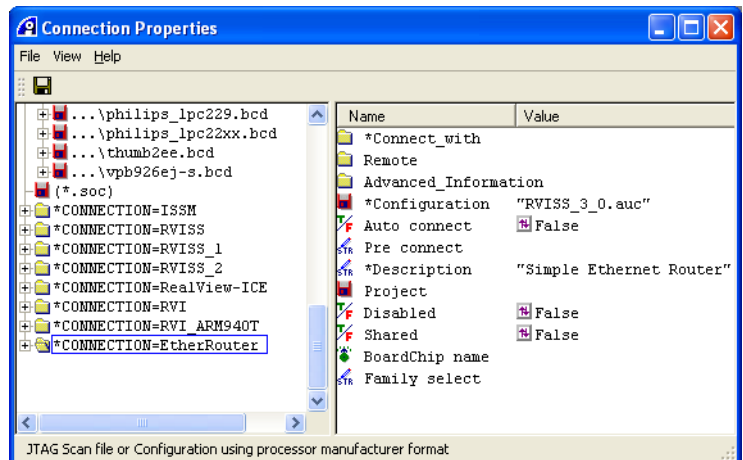


Figure 5-6 Connection properties for the RVI Debug Configuration

2. Right-click on the BoardChip\_name setting, in the right pane, to display the context menu.
3. Select **EtherRouter** from the context menu. A new setting is displayed in the right pane with an asterisk \* beside it, shown in Figure 5-7.

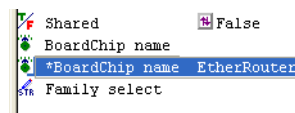


Figure 5-7 Assigning the EtherRouter BCD group

4. Save your changes.

#### See also

- *The RealView Debugger search path on page 1-17*
- *Assigning board/chip definitions.*

### 5.9.2 Assigning the AMDLANCE and KS32C50100 chips to the EtherRouter board

To assign the KS32C50100 and AMDLANCE chips to the EtherRouter board:

1. Right-click on the \*BoardChip\_name EtherRouter setting in the right pane to display the context menu.
2. Select **Jump to Definition** from the context menu. This expands the group (\*.bcd) Board/Chip Definitions to show the current list of BCD files. The ...\\EtherRouter.bcd group is also expanded, and the BOARD=EtherRouter group is selected.
3. Left-click on the BoardChip\_name setting in the right pane to display the context menu.
4. Select **KS32C50100** from the context menu. A new setting is added to the right pane with an asterisk \* beside it. That is, \*BoardChip\_name KS32C50100.
5. Right-click on the BoardChip\_name that does not have an asterisk again to display the context menu.
6. Select **AMDLANCE** from the context menu. A new setting is added to the right pane with an asterisk \* beside it, shown in Figure 5-8.

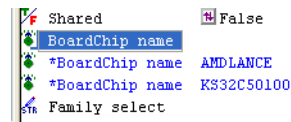


Figure 5-8 Setting up the EtherRouter.bcd

7. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
8. Click the **OK** button to close the Connection Properties dialog box.

#### See also

- *The RealView Debugger search path* on page 1-17
- *Assigning board/chip definitions* on page 5-13.

## 5.10 Creating the memory map

If you want to set up a memory map that is used automatically when you connect to a target, you must set up the memory map definition in the Memory\_block group of a BCD file.

See also:

- *Creating the M\_IO\_REGS memory map block*
- *Creating the M\_I2C memory map block* on page 5-16
- *Viewing the memory map* on page 5-17
- *Creating the enumerations for the register values* on page 5-19.

### 5.10.1 Creating the M\_IO\_REGS memory map block

To create the M\_IO\_REGS memory map block:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties dialog box.
2. Expand the (\*.bcd) Board/Chip Definitions group.
3. Expand the ... \EtherRouter.bcd group.
4. Expand the following groups in turn:
  - a. \*BOARD=EtherRouter
  - b. Advanced\_Information
  - c. ARM
  - d. Memory\_block.
5. Create a new memory map block under Memory\_block:
  - a. Right-click on Memory\_block to display the context menu.
  - b. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - c. Enter **M\_IO\_REGS** for the name for the memory map block.
  - d. Click **Create**.
6. Delete the Default memory block:
  - a. Right-click on Default to display the context menu.
  - b. Select **Delete** from the context menu.
7. Edit the settings for the new M\_IO\_REGS memory block:
  - a. Select the M\_IO\_REGS memory block.
  - b. Set the value of Start to the start address of the memory block, for example, **0x20000000**.
  - c. Set the value of Length in memory units for the memory block. For example, **0x800000** bytes for an ARM architecture-based processor.
  - d. Set Description to **I/O Registers**, shown in Figure 5-9 on page 5-16.

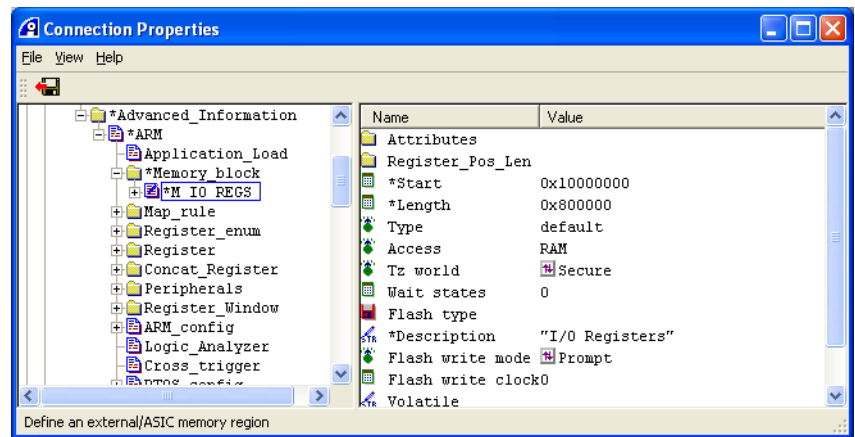


Figure 5-9 Configuring the M\_IO\_REGS custom register

8. Save your changes.

#### See also

- *Creating the memory map on page 5-15.*

### 5.10.2 Creating the M\_I2C memory map block

To create the M\_I2C memory map block:

1. Create the M\_I2C memory map block:
  - a. Right-click on \*Memory\_block to display the context menu.
  - b. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - c. Enter **M\_I2C** for the name for the memory map block.
  - d. Click **Create**.
2. Edit the settings for the new M\_I2C memory block:
  - a. Select the M\_I2C memory block.
  - b. Set the value of Start to the start address of the memory block, for example, **0x10002000**.
  - c. Set the value of Length in memory units for the memory block. For example, **0x1000** bytes for an ARM architecture-based processor.
  - d. Set Description to **I2C Control Registers**, shown in Figure 5-10 on page 5-17.



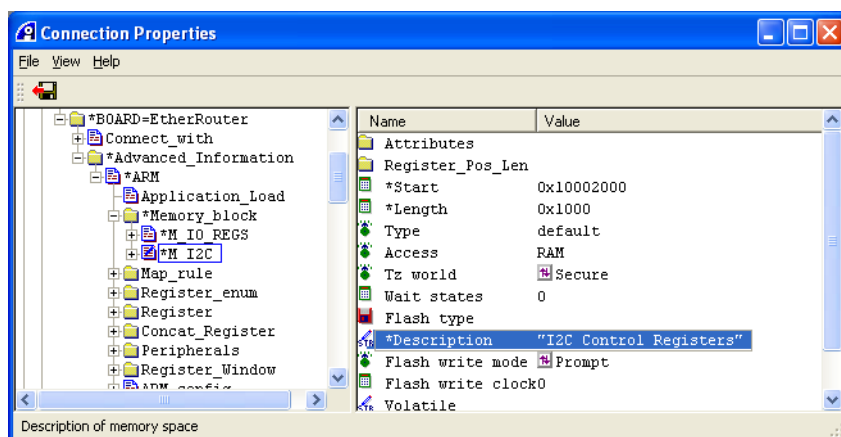


Figure 5-10 Configuring the M\_I2C custom register

3. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

### See also

- *Creating the memory map* on page 5-15.

### 5.10.3 Viewing the memory map

To view the newly created memory map:

1. Connect to the ARM926EJ-S target in the RVISS EtherRouter Debug Configuration.
2. Select **Memory Map Tab** from the **View** menu to display the new memory map. Figure 5-11 shows an example:

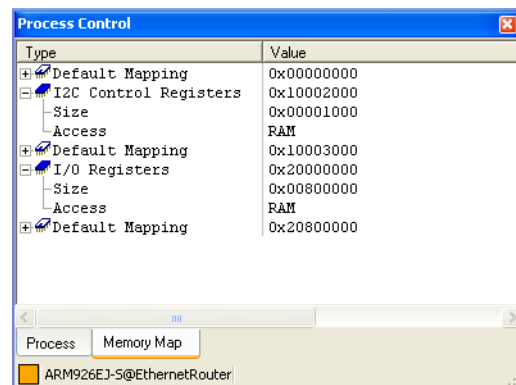


Figure 5-11 New memory map in the Process Control view

3. Load an image, for example:  
`install_directory\RVD\Examples\...\dhrystone\Debug\dhrystone.axf`
4. Click the **Memory Map** tab in Process Control view to display the new memory map.
5. Expand the Sect ER\_R0,ER\_RW,ER\_ZI block as shown in Figure 5-12 on page 5-18. Details of the loaded image are visible.

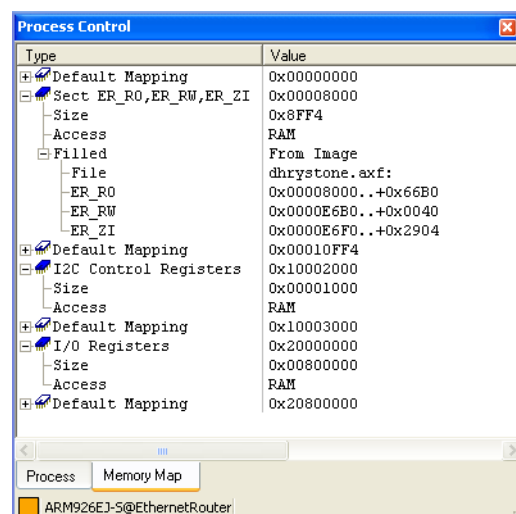


Figure 5-12 Memory map with dhrystone image loaded

6. Disconnect from the ARM926EJ-S target.

#### See also

- *Creating the memory map* on page 5-15.

## 5.11 Creating the enumerations for the register values

To create the enumerations for the appropriate register values:

1. Right-click on the EthernetRouter Debug Configuration to display the context menu.
2. Select **Properties...** from the context menu to display the Connection Properties dialog box.
3. Click the **Advanced** button to display the Connection Properties window.
4. Right-click on the \*BoardChip\_name EtherRouter setting in the right pane to display the context menu.
5. Select **Jump to Definition** from the context menu.  
The \*BOARD=EtherRouter group in the ... \EtherRouter.bcd entry is selected.
6. Expand the following groups in turn:
  - a. \*BOARD=EtherRouter
  - b. \*Advanced\_Information
  - c. \*ARM.
  - d. Register\_enum.
 A Default group is available.
7. Create the E\_REMAP enumeration:
  - a. Right-click on Register\_enum to display the context menu.
  - b. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - c. Enter **E\_REMAP** for the name of the group.
  - d. Click **Create** to close the dialog box. A new E\_REMAP group is created.
  - e. Select the E\_REMAP group in the left pane. The group contents are displayed in the right pane.
  - f. Set the value of Names in the right pane to **RAM,ROM**. Press Enter to complete the entry. A new \*Names "RAM,ROM" setting is created.
8. Create the E\_SWITCH enumeration:
  - a. Right-click on \*Register\_enum to display the context menu.
  - b. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - c. Enter **E\_SWITCH** for the name of the group.
  - d. Click **Create** to close the dialog box. A new E\_SWITCH group is created.
  - e. Select the E\_SWITCH group in the left pane. The group contents are displayed in the right pane.
  - f. Set the value of Names in the right pane to **Off,On**. Press Enter to complete the entry. A new \*Names "Off,On" setting is created.
9. Create the E\_ENABLE enumeration:
  - a. Right-click on the Default group to display the context menu.
  - b. Select **Rename** from the context menu to display the Enter Value dialog box.
  - c. Enter **E\_ENABLE** for the name of the group.
  - d. Click **Rename**. The name changes from Default to E\_ENABLE.

- e. Select the E\_ENABLE group in the left pane to display the group contents.
- f. Set the value of the Names setting in the right pane to **Disable,Enable**. Press Enter to complete the entry. A new \*Names "Disable,Enable" setting is created, shown in Figure 5-13.

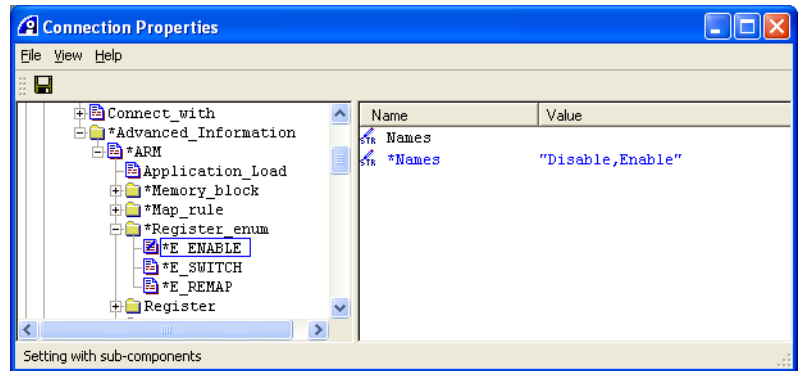


Figure 5-13 Creating enumerations

10. Select **Save Changes** from the **File** menu to save these changes.

See also:

- *Creating a custom register* on page 5-21.

## 5.12 Creating a custom register

When creating a custom register you can also define the bit fields for that register.

See also:

- *Creating the custom register*
- *Defining the bit fields for the custom register*
- *Creating the register tab for displaying custom registers on page 5-24.*

### 5.12.1 Creating the custom register

To create the custom register:

1. Expand the Register group. A Default group is available.
2. Rename the Default group to **MemoryStatus**:
  - Right-click on the Default group to display the context menu.
  - Select **Rename** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - Enter **MemoryStatus** for the name of this register.
  - Click **Rename**. The name changes from Default to MemoryStatus.
3. Select the MemoryStatus register group in the left pane.
4. Right-click on Base in the right pane to display the context menu.
5. Select **M\_IO\_REGS** from the context menu.
6. Set the value of Start to **0x20**. This is the offset of the register from the base address of the M\_IO\_REGS memory block.
7. Right-click on Gui\_name to display the context menu.
8. Select **Edit Value...** from the context menu.
9. Enter **Memory Status Register**.
10. Save your changes.

**See also**

- *Creating a custom register.*

### 5.12.2 Defining the bit fields for the custom register

To define the bit fields for the Newreg custom register:

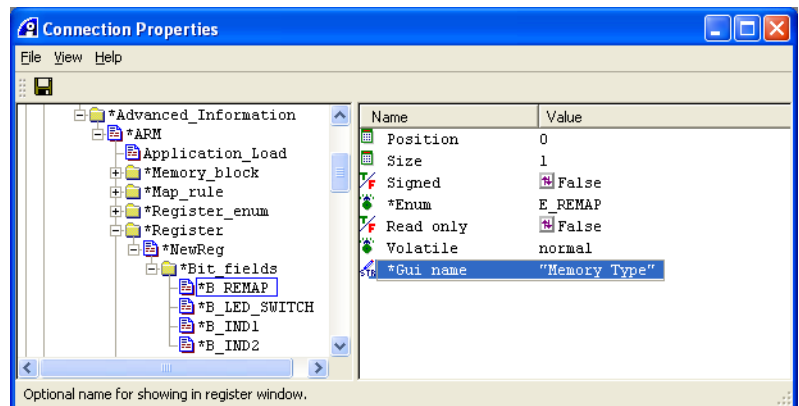
1. Expand the MemoryStatus register. A Bit\_fields group is available.
2. Expand the Bit\_fields group. You are now going to set up four bit fields.
3. Create the first bit field:
  - a. Right-click on the Default group to display the context menu.
  - b. Select **Rename** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - c. Enter **B\_REMAP** for the name of the bit field.
  - d. Click **Rename**. The name changes from Default to B\_REMAP.

4. Create the additional bit fields:
  - a. Right-click on B\_REMAP to display the context menu
  - b. Select **Make Copy...** from the context menu to display the Enter Name of New object dialog box. The name B\_REMAP\_1 is inserted.
  - c. Change the name to B\_LED\_SWITCH.
  - d. Click **Create** to create the B\_LED\_SWITCH bit field.
  - e. Repeat these steps to create the B\_IND1 and B\_IND2 bit fields.

**———— Note ————**

If you right-click on the last bit field you create, then RealView Debugger automatically assigns the next integer value to the name.

5. Select B\_REMAP in the left pane and set the following values, shown in Figure 5-14:
  - a. Make sure Size is set to **1** (this is the default value).
  - b. Set Enum to **E\_REMAP**.
  - c. Set GUI\_name to **Memory Type**.
  - d. Leave all other settings to the default values.



**Figure 5-14 Creating bit field for a custom register**

6. Select the B\_LED\_SWITCH group and set the following values:
  - a. Set Position to **1**.
  - b. Make sure Size is set to **1** (this is the default value).
  - c. Set Enum to **E\_SWITCH**.
  - d. Set GUI\_name to **LED Switch**.
  - e. Leave all other settings to the default values.
7. Select the B\_IND1 groups and set the following values:
  - a. Set Position to **2**.
  - b. Set Size to **4**.
  - c. Set GUI\_name to **IND1**.
  - d. Leave all other settings to the default values.
8. Select the B\_IND2 group and set the following values:
  - a. Set Position to **6**.
  - b. Set Size to **4**.
  - c. Set GUI\_name to **IND2**.
  - d. Leave all other settings to the default values.

9. Save your changes.

**See also**

- *Creating a custom register* on page 5-21.

## 5.13 Creating the register tab for displaying custom registers

Having created the custom registers, you must have a way of displaying them in RealView Debugger, so that you can monitor or modify the values. You must define the visual appearance of the registers, and specify the name of the tab that is to contain the registers. The tab is displayed in the Registers view.

See also:

- *Defining the visual appearance of the custom registers*
- *Viewing the custom registers* on page 5-25
- *Setting up controlled memory map blocks* on page 5-27.

### 5.13.1 Defining the visual appearance of the custom registers

To define the visual appearance of the custom registers in the Registers view:

1. Expand the Register\_Window group in the left pane.
2. Rename the Default group under Register\_Window to **EtherRouter**. This specifies the name for the new register tab in the Registers view.
3. Select EtherRouter in the left pane.
4. Set the Line setting to \$+.

This makes the MemoryStatus register group an expandable entry in the register tab.

———— **Note** ————

It is important in the following steps to right-click on the last \*Line you created when adding a new Line. If you right-click on the original Line setting, the lines are added in reverse order.

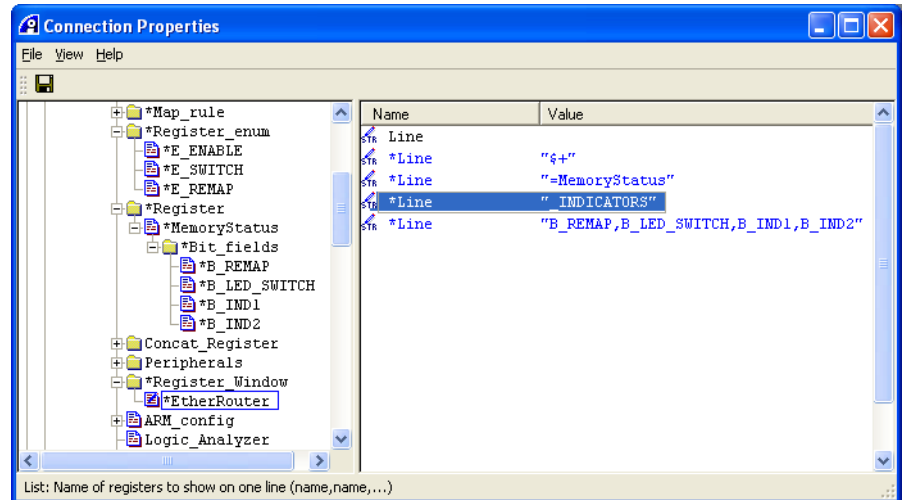
5. Create a line setting to display the MemoryStatus register:
  - a. Right-click on the \*Line "\$+" setting to display the context menu.
  - b. Select **Make New...** from the context menu.
  - c. Set the \*Line setting to **=MemoryStatus**.
  - d. Press Enter to complete the entry.

This displays the MemoryStatus register. The name displayed for the register is defined by the Gui\_name setting (that is Memory Remap Status), and the name and register value are displayed on the same line in the register tab.
6. Create a line setting for the Newreg register to display the name INDICATORS in the register tab:
  - a. Right-click on the \*Line "=MemoryStatus" setting to display the context menu.
  - b. Select **Make New...** from the context menu.
  - c. Set the \*Line setting to **\_INDICATORS**.  
Literals entered in \*Line (or Line) must be preceded by an underscore. The underscore is not displayed in the tab.
  - d. Press Enter to complete the entry.
7. Create a line for bit fields B\_REMAP, B\_LED\_SWITCH, B\_IND1, and B\_IND2 of the MemoryStatus register:
  - a. Right-click on the \*Line "\_INDICATORS" setting to display the context menu.
  - b. Select **Make New...** from the context menu.



- c. Set the \*Line setting to **B\_REMAP,B\_LED\_SWITCH,B\_IND1,B\_IND2**.
- d. Press Enter to complete the entry.

The Connection Properties window looks like Figure 5-15.



**Figure 5-15 The EtherRouter Register\_window group**

All board file entries are now complete.

8. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
9. Click the **OK** button to close the Connection Properties dialog box.

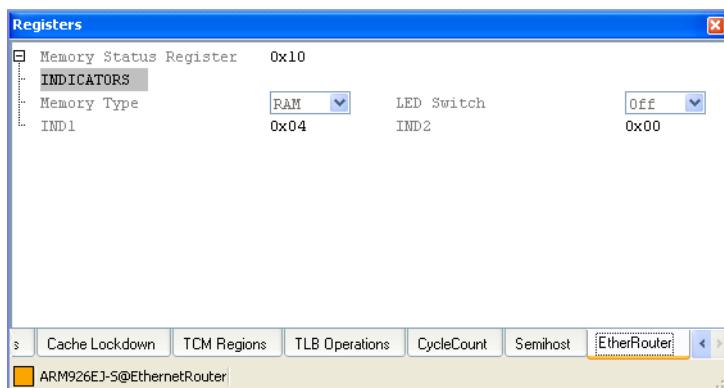
#### See also

- *Creating the register tab for displaying custom registers* on page 5-24
- *Viewing the custom registers*
- *Setting up controlled memory map blocks* on page 5-27
- *Register\_Window* on page A-30

### 5.13.2 Viewing the custom registers

In the last stage, display the new register in the Registers view:

1. Connect to the ARM926EJ-S target.
2. Select **Registers** from the **View** menu to display the Registers view.
3. Select the **EtherRouter** tab to view the custom registers. Figure 5-16 on page 5-26 shows an example:



**Figure 5-16 EtherRouter tab in the Registers view**

4. Disconnect from the ARM926EJ-S RVISS target.

### See also

- *Creating the register tab for displaying custom registers* on page 5-24
- *Defining the visual appearance of the custom registers* on page 5-24.

## 5.14 Setting up controlled memory map blocks

Define the two controlled memory map blocks named M\_FastRAM and M\_SlowROM:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties dialog box.
2. Expand the (\*.bcd) Board/Chip Definitions group.
3. Expand the ... \EtherRouter.bcd group.
4. Expand the following groups in turn:
  - a. \*BOARD=EtherRouter
  - b. \*Advanced\_Information
  - c. \*ARM
  - d. \*Memory\_Block.
5. Create the M\_FastRAM memory block:
  - a. Right-click on the \*Memory\_block group in the left pane to display the context menu.
  - b. Select **Make New...** from the context menu to display the Enter Name of New object dialog box. The name default is inserted.
  - c. Enter the name **M\_FastRAM**.
  - d. Click **Create**.
6. Change the settings for the M\_FastRAM memory block, shown in Figure 5-17:
  - a. Select the M\_FastRAM group in the left pane.
  - b. Set the value of Start to the start address of the memory block, for example, **0x0** (this is the default).
  - c. Set the value of Length in memory units for the memory block. For example, **0x80000** bytes for an ARM architecture-based processor.
  - d. Set Description to **Fast Static RAM**.
  - e. Set Access to **RAM** (this is the default).

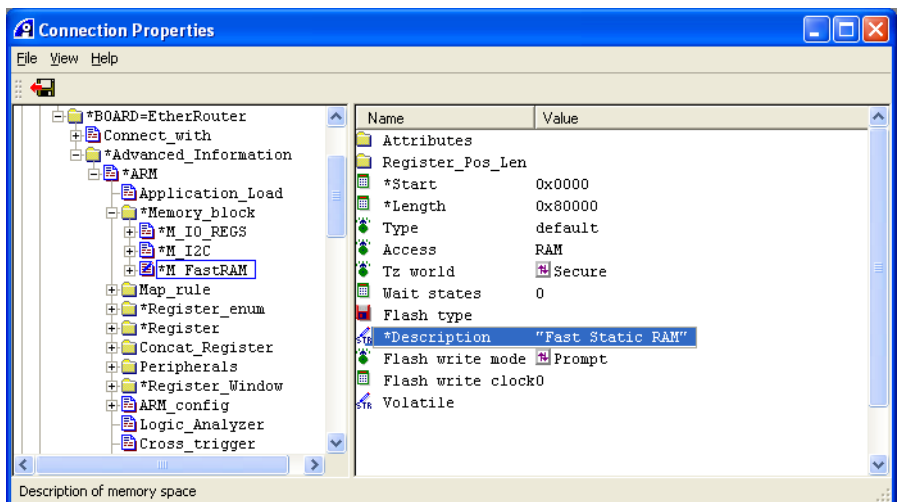


Figure 5-17 M\_FastRAM memory block

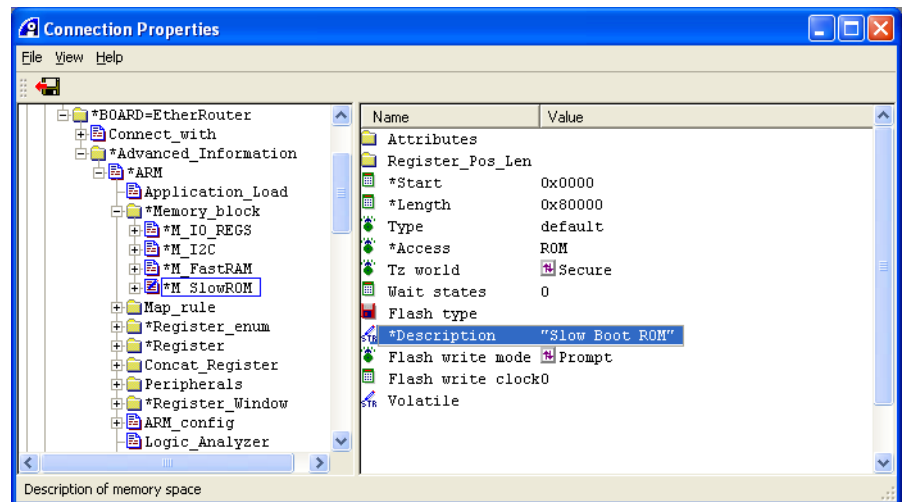
7. Create the M\_SlowROM memory block:
  - a. Right-click on M\_FastRAM to display the context menu.

- b. Select **Make Copy...** from the context menu to display the Enter Name of New object dialog box. The name M\_FastRAM\_1 is inserted.

———— **Note** ————

If the name of a memory block you are copying ends with a number, then RealView Debugger increments the number for the new block name.

- c. Change the name to **M\_SlowROM**.
  - d. Click **Create**.
8. Change the settings for the M\_SlowROM memory block, shown in Figure 5-18:
    - a. Select the M\_SlowROM group in the left pane.
    - b. Set Access to **ROM**.
    - c. Set \*Description to **Slow Boot ROM**.



**Figure 5-18 M\_SlowROM memory block**

9. Save your changes.

See also:

- *Creating memory map rules* on page 5-29.

## 5.15 Creating memory map rules

Create the rules that determine which controlled memory map block is displayed:

1. Expand the Map\_rule group.
2. Select the Default group.
3. Rename the Default group to **R\_FastRAM**.
4. Set the following values, shown in Figure 5-19:
  - a. Set Register to **MemoryStatus** (use the context menu).
  - b. Set Mask to **0x0001** (to check the value of the B\_REMAP bit field only).
  - c. Set Value to **0** (M\_FastRAM displayed when B\_REMAP bit field is set to Disable).
  - d. Set On\_equal to **M\_FastRAM**.

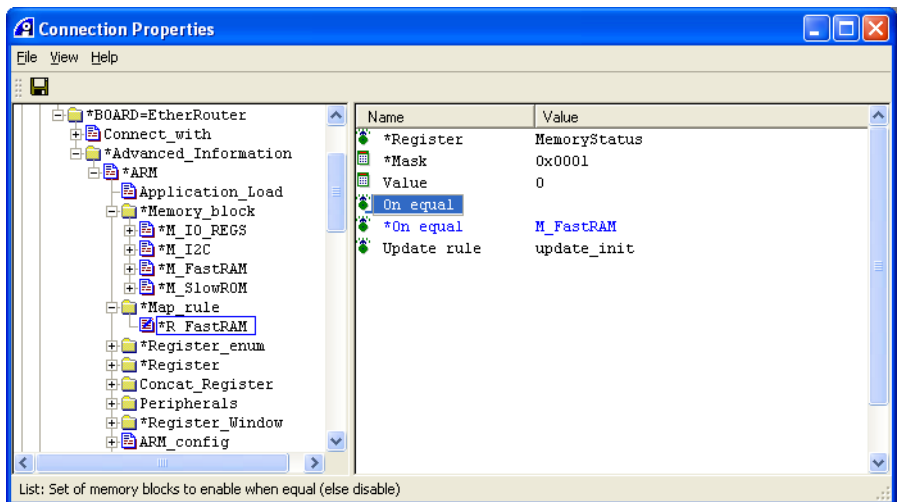
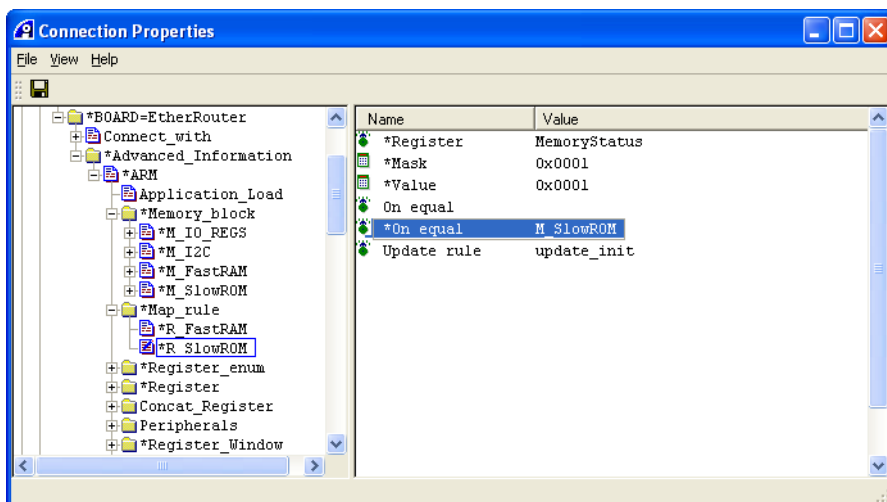


Figure 5-19 Settings for the R\_Fast\_RAM map rule

5. Create the R\_SlowROM rule:
  - a. Right-click on R\_FastRAM to display the context menu.
  - b. Select **Make Copy...** from the context menu to display the Enter Name of New object dialog box. The name R\_FastRAM\_1 is inserted.
 

**Note**

If the name of a rule you are copying ends with a number, then RealView Debugger increments the number for the new rule.
  - c. Change the name to **R\_SlowROM**.
  - d. Click **Create**.
6. Select R\_SlowROM and set Value to **1**. That is, display the M\_SlowROM memory block when the B\_REMAP bit field is set to Enable.
7. Set the value of \*On\_equal to **M\_SlowROM**, shown in Figure 5-20 on page 5-30.



**Figure 5-20 Settings for the R\_SlowROM map rule**

All BCD file entries are now complete.

8. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

See also:

- *Displaying the controlled memory map blocks on page 5-31.*

## 5.16 Displaying the controlled memory map blocks

To display the controlled memory map blocks:

1. Connect to the ARM926EJ-S target in the EtherRouter Debug Configuration.
2. Select **Registers** from the **View** menu to display the Registers view.
3. Select the **EtherRouter** tab to view the custom registers. Figure 5-21 shows an example. You might have to click on the right scroll button to locate the tab.



Figure 5-21 EtherRouter tab in the Registers view

4. Select **Memory Map Tab** from the **View** menu to display the Process Control view with the **Memory Map** tab selected. Figure 5-22 shows an example:

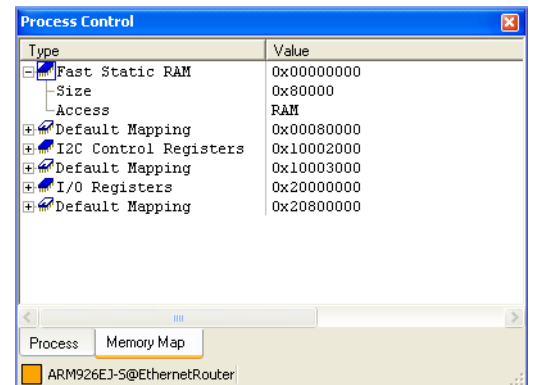
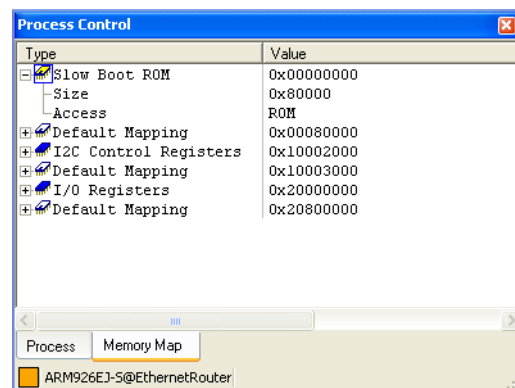


Figure 5-22 RAM memory block in the Memory Map tab

5. In the Registers view, change the value of Memory Type to **ROM**. This activates the memory rule (R\_SlowROM) and so changes the memory that is mapped at address 0x00000000.

The memory block in the **Memory Map** tab at address 0x00000000 changes to a ROM entry, indicated by a yellow icon, shown in Figure 5-23 on page 5-32.



**Figure 5-23 Activated ROM memory block in the Memory Map tab**

See also:

- *Creating a concatenated register* on page 5-33.



## 5.17 Creating a concatenated register

In this part of the tutorial, you create a concatenated register C\_R8\_R9\_concat for the EthernetRouter Debug Configuration. The concatenated register has the following attributes:

- the high byte of the core register R8 forms the low byte of C\_R8\_R9\_concat
- the low byte of the core register R9 forms the high byte of C\_R8\_R9\_concat.

See also:

- *Creating the C\_R8\_R9\_concat concatenated register*
- *Viewing the concatenated register* on page 5-34.

### 5.17.1 Creating the C\_R8\_R9\_concat concatenated register

To create the C\_R8\_R9\_concat concatenated register:

1. Right-click on the EthernetRouter Debug Configuration in the Connect to Target window to display the context menu.
2. Select **Properties...** from the context menu to display the Connection Properties dialog box.
3. Click the **Advanced** button to display the Connection Properties window. The settings group for the Debug Configuration is selected (in this example, CONNECTION=EthernetRouter).
4. Locate the BCD file where you want to define the custom memory mapped registers:
  - a. Right-click on the \*BoardChip\_name EtherRouter setting in the right pane to display the context menu.
  - b. Select **Jump to Definition** from the context menu.  
The BOARD=EtherRouter group is selected in the BCD file.
5. Expand the following groups in turn:
  - a. BOARD=EtherRouter
  - b. \*Advanced\_Information
  - c. \*ARM
  - d. Concat\_Register.
6. Create the concatenated register as follows:
  - a. Rename the Default block to **C\_R8\_R9\_concat**.
  - b. Select C\_R8\_R9\_concat in the left pane.  
The settings for the concatenated register are displayed.
  - c. Change the C\_R8\_R9\_concat settings to the values shown in Table 5-1.

**Table 5-1 R8\_R9\_concat settings**

Setting	Value
Low_name	<b>R8</b>
Low_shift	<b>16 (0x0010)</b>
Low_mask	<b>0xFFFF</b>

Table 5-1 R8\_R9\_concat settings (continued)

Setting	Value
High_name	<b>R9</b>
High_shift	<b>-16</b> (left shift)
High_mask	<b>0xFFFF0000</b>

7. Add the concatenated register to the **EtherRouter** tab:
  - a. Expand the Register\_Window group in the left pane.
  - b. Select EtherRouter to display the related settings.
  - c. Right-click on the first Line setting to display the context menu.
  - d. Select **Edit Value...** from the context menu.
  - e. Enter **C\_R8\_R9\_concat**. Remember to press Enter to complete the entry.
8. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.
9. Click the **OK** button to close the Connection Properties dialog box.

#### See also

- *Creating a concatenated register* on page 5-33
- *Viewing the concatenated register*
- *Concat\_Register* on page A-28.

### 5.17.2 Viewing the concatenated register

To view the concatenated register created in *Creating a concatenated register* on page 5-33:

1. Connect to the ARM926EJ-S target in the EtherRouter Debug Configuration.
2. Select **Registers** from the **View** menu to display the Registers view. Figure 5-24 shows an example:

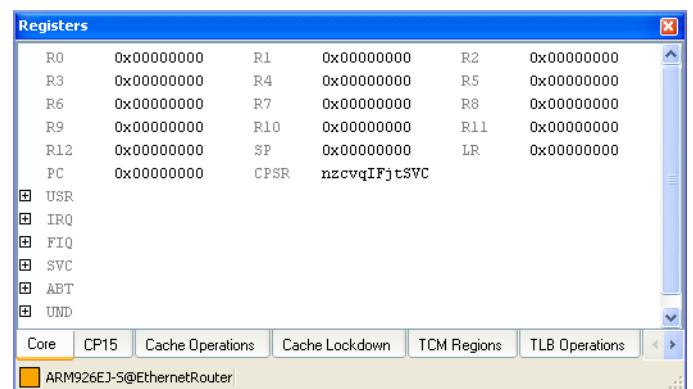
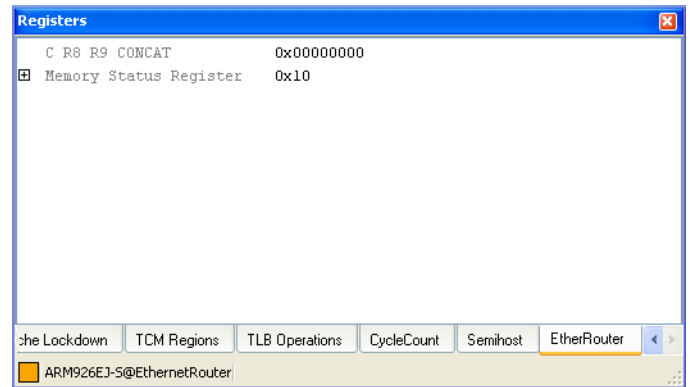


Figure 5-24 Core tab shown in the Registers view

3. Copy the R8 and R9 registers to the User view:
  - a. Select both the R8 and R9 registers.
  - b. Right-click one of the selected registers to display the context menu.
  - c. Select **Copy to User View** from the context menu.

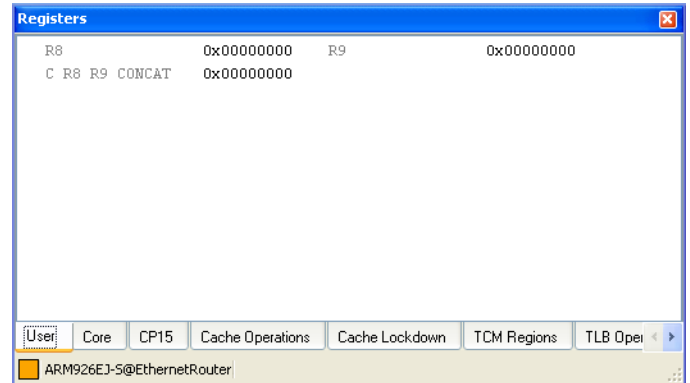
4. Click the **EtherRouter** tab to display the registers for the Ethernet router. Figure 5-25 shows an example:



**Figure 5-25 Concatenated register in EtherRouter tab**

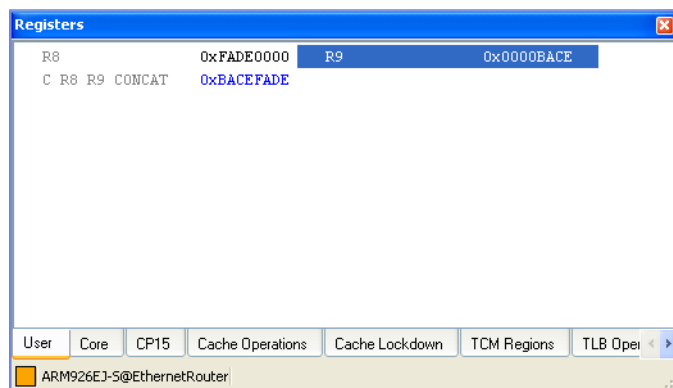
5. Copy the concatenated register to the User view:
  - a. Click the **EtherRouter** tab.
  - b. Right-click on the concatenated register to display the context menu.
  - c. Select **Copy to User View** from the context menu.
6. Display the User view:
  - a. Right-click in the view to display the context menu.
  - b. Select **Show User View** from the context menu to display the **User** tab.

Figure 5-26 shows an example:



**Figure 5-26 User view showing the concatenated registers**

7. Change the values of the R8 and R9 registers to see the value of the concatenated register change. For example, set R8 to 0xFADE0000 and R9 to 0x0000BACE. Figure 5-27 on page 5-36 shows an example:



**Figure 5-27** User view showing concatenated register values

**See also**

- *Creating a concatenated register* on page 5-33
- *Creating the C\_R8\_R9\_concat concatenated register* on page 5-33.

# Chapter 6

## Programming Flash with RealView Debugger

This chapter describes how to use RealView® Debugger to program Flash memory on your development platform. It contains:

- *Introduction to Flash programming with RealView Debugger* on page 6-2
- *RealView Debugger files used for Flash programming* on page 6-7
- *pakflash utility command syntax* on page 6-10
- *Programming Flash on the ARM development boards* on page 6-11
- *Programming Flash for a custom development platform* on page 6-16
- *Gathering information about your development platform* on page 6-19
- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20
- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23
- *Creating the Flash-level and board-level AME files* on page 6-29
- *Generating the FME file* on page 6-34
- *Checking the FME file with the dispflash utility* on page 6-38
- *Creating a BCD file* on page 6-39
- *Programming an image into Flash* on page 6-44
- *Troubleshooting* on page 6-46.

## 6.1 Introduction to Flash programming with RealView Debugger

RealView Debugger enables you to program Flash memory (that is, download programs or patch code and data). Unlike standard RAM-based memory, you cannot program Flash memory directly with RealView Debugger. This is because Flash has a block structure and requires various control signals to be generated to access the device.

How RealView Debugger writes to Flash memory depends on:

- the Flash type
- the specific device used
- how this device is integrated into your design.

Programming Flash with RealView Debugger requires the following files:

- Flash MEmory file
- Board/Chip Definition file.

RealView Debugger provides files for various ARM and third-party development platforms. If the files provided do not fit your requirements, then you must create your own files. How you create these files depends on whether or not your development platform uses a Flash type that is already supported by RealView Debugger.

See also:

- *Summary of files used to program Flash on supported development platforms*
- *Summary of files used to program custom Flash types on custom platforms* on page 6-6.

### 6.1.1 Summary of files used to program Flash on supported development platforms

Table 6-1 shows the files used to program Flash for the development platforms that are provided with RealView Debugger.

**Table 6-1 Files used to program Flash on supported development platforms**

Platform	Flash type supported	Files used to create the FME file	Files used to program Flash <sup>a</sup>
ARM® Evaluator-7T	AMD	board_amd_eval7T.ame b_evaluator7t.s flash_amd.ame f_amd_sst_arm.s flash_EVALUATOR_7T.mk	flash_amd_eval7t.fme Eval7T.bcd
	SST	board_sst_eval7T.ame b_evaluator7t.s flash_sst.ame f_amd_sst_arm.s flash_EVALUATOR_7T.mk	flash_sst_eval7t.fme Eval7T.bcd
EB ARM11 MPCore	Intel	b_EB_CT11MPCore_intel.s board_intel_EB_CT11MPCore.ame EB_CT11MPCore_intel.mk f_intel_arm.s flash_intel.ame	EB_CT11MPCORE_28F256L30B90.fme EB_CT11MPCore.bcd
EB ARM1136	Intel	b_EB_CT1136_intel.s board_intel_EB_CT1136.ame EB_CT1136_intel.mk f_intel_arm.s flash_intel.ame	EB_CT1136_28F256L30B90.fme EB_1136.bcd

Table 6-1 Files used to program Flash on supported development platforms (continued)

Platform	Flash type supported	Files used to create the FME file	Files used to program Flash <sup>a</sup>
ARM Integrator™/AP	Intel	b_IntegratorAP.s board_intel_arm.ame f_intel_arm.s flash_IntegratorAP.mk flash_intel.ame	flash_IntegratorAP.fme AP.bcd
ARM Integrator/CP	Intel	b_IntegratorCP.s board_intel_arm.ame f_intel_arm.s flash_IntegratorCP.mk flash_intel.ame	flash_IntegratorCP.fme CP.bcd
Atmel AEB1	Intel	b_at91eb01.s board_aeb1.ame f_intel_arm.s flash_aeb1.mk flash_intel.ame	flash_aeb1.fme
Atmel AT91EB01, AT91EB01_UB, and AT91EB63	Atmel	b_at91eb01.s board_atmel_arm.ame f_atmel_arm.s flash_at91eb01.mk flash_atmel.ame	flash_AT91EB01_UB.fme
Icyteature ICYIMX3	S29GL512N	board_ICYIMX35.ame board_ICYIMX35.s f_S29GL512N_arm.s flash_S29GL512N.ame ICYMX35_S29GL512N.mk	ICYMX35_S29GL512N.fme ICYMX35.bcd
iMX27_LiteKit	Intel	b_iMX27_LiteKit.s board_iMX27_LiteKit.ame f_intel_arm.s flash_intel.ame iMX27_LiteKit.mk	iMX27_LiteKit.fme iMX27_LiteKit.bcd
iMX31_lite	Intel	b_iMx31_lite.s board_intel_arm.ame f_intel_arm.s flash_intel.ame iMx31_lite.mk	iMx31_lite.fme iMX31_LiteKit.bcd
OSK5912	Micron	b_omap_5912.s board_micron_arm.ame f_micron_arm.s flash_micron.ame omap_5912.mk	omap_5912.fme OMAP5912.bcd
PB ARM11 MPCore	Intel	b_PBARM11MPCore_intel.s board_intel_PBARM11MPCore.ame f_intel_arm.s flash_intel.ame PBARM11MPCore_intel.mk	b_PBARM11MPCore_intel.fme PBARM11MPCore.bcd
PB ARM1176	Intel	b_PBARM1176_intel.s board_intel_PBARM1176.ame f_intel_arm.s flash_intel.ame PBARM1176_intel.mk	PBARM1176_intel.fme PB1176JZF-S.bcd

Table 6-1 Files used to program Flash on supported development platforms (continued)

Platform	Flash type supported	Files used to create the FME file	Files used to program Flash <sup>a</sup>
PB Cortex-A8	Intel	b_PBCortex-A8_intel.s board_intel_PBCortex-A8.ame f_intel_arm.s flash_intel.ame PBCortex-A8_intel.mk	b_PBCortexA8_intel.fme PB-A8.bcd
PBX Cortex-A9	Intel	b_PBXcortex-A9_intel.s board_intel_PBXcortex-A9.ame f_intel_arm.s flash_intel.ame PBXCortex-A9_intel.mk	b_PBXcortexA9_intel.fme PBX-A9.bcd
Philips LPC210x	LPC210x	b_lpc210x.s board_lpc210x_arm.ame f_lpc210x.s flash_lpc210x.ame makefile	flash_lpc210x.fme philips_lpc221.bcd philips_lpc2104_5_6.bcd philips_lpc21x4_21x9.bcd
Philips LPC213x	LPC213x	b_lpc213x.s board_lpc2131_arm.ame board_lpc2132_arm.ame board_lpc2134_arm.ame board_lpc2136_arm.ame board_lpc2138_arm.ame f_lpc213x.s flash_lpc213x.ame makefile	flash_lpc2131.fme flash_lpc2132.fme flash_lpc2134.fme flash_lpc2136.fme flash_lpc2138.fme philips_lpc210.bcd philips_lpc214.bcd
Philips LPC2294	LPC2294	b_lpc2294.s board_lpc2294_arm.ame f_lpc2294.s flash_lpc2294.ame makefile	flash_lpc2294.fme philips_lpc229.bcd philips_lpc21x4_21x9.bcd
PHYTEC phyCORE-iMX27	Intel	b_iMx27_phytec.s board_intel_arm.ame f_intel_arm.s flash_intel.ame iMx27_phytec.mk	phyCORE_iMX27.fme phyCORE_iMX27.bcd
PHYTEC phyCORE-iMX31	Intel	b_iMx31_phytec.s board_intel_arm.ame f_intel_arm.s flash_intel.ame iMx31_phytec.mk	iMx31_phytec.fme phyCORE_iMX31.bcd
PHYTEC phyCORE-iMX35	Intel	board_iMx35_phytec.ame board_iMx35_phytec.s f_intel_arm.s flash_intel.ame phytec_iMX35_PC28F256.mk	phytec_iMX35_PC28F256.fme phyCORE_iMX35.bcd
SAMSUNG S3C2450	AMD	b_2450_AMD.s board_AMD_2450.ame f_AM29LV800B_arm.s flash_AM29LV800B.ame S3C2450_AM29LV800B.mk	S3C2450_AM29LV800B.fme SMDK2450_NOR_BOOT.bcd
SAMSUNG S5PC100	AMD	board_AMD_SMDKC100.ame f_AM29LV800B_arm.s flash_AM29LV800B.ame SMDKC100_AM29LV800B.mk SMDKC100_AMD.s	SMDKC100_AM29LV800B.fme S5PC100_Essentials.bcd



**Table 6-1 Files used to program Flash on supported development platforms (continued)**

Platform	Flash type supported	Files used to create the FME file	Files used to program Flash <sup>a</sup>
STMicroelectronics ST30	ST30	b_st30.s board_st30_arm.ame f_st30_arm.s flash_st30.ame st30.mk	flash_st30.fme
STMicroelectronics	STARM71xF	b_starm71xf.s board_starm71xf_arm.ame f_starm71xf_arm.s flash_starm71xf.ame starm71xf.mk	flash_starm71xf.fme
Versatile AB926EJ-S	AMD	b_VPB926EJ-S_amd.s board_amd_VPB926EJ-S.ame f_intel_arm.s flash_amd.ame VAB926EJ-S_amd.mk	VPB926EJS.fme VAB926EJ-S.bcd
Versatile PB926EJ-S 64KB	Intel	b_VPB926EJ-S_64KB_intel.s board_intel_VPB926EJ-S_64KB.ame f_intel_arm.s flash_intel.ame VPB926EJ-S_64KB_intel.mk	VPB926EJ-S_64KB_28F256L30B90.fme vpb926ej-s_64KB.bcd
Versatile PB926EJ-S 256KB	Intel	b_VPB926EJ-S_256KB_intel.s board_intel_VPB926EJ-S_256KB.ame f_intel_arm.s flash_intel.ame VPB926EJ-S_256KB_intel.mk	VPB926EJ-S_256KB_28F256L30B90.fme vpb926ej-s_256KB.bcd

a. The specified .bcd files are in the C:\Documents and Settings\userID\Local Settings\Application Data\ARM\rvdebug\version\shadowbase\etc directory.

The files provided are:

- assembly files (.s) containing the Flash algorithms and board-level routines required to program the Flash
- ASCII Method files (.ame) that describe the Flash memory for the target
- Flash Method files (.fme) generated with the pakflash utility from assembly code and ASCII Method files
- A Board/Chip Definition file (.bcd) that describes the memory map for your custom platform, and specifies the Flash Method file that RealView Debugger is to use to program the Flash device.

Makefiles to create the FME file from these sources are also provided in the *install\_directory\RVD\Flash\...\platform* directory, for example in:

*install\_directory\RVD\Flash\...\platform\IntegratorCP\Flash\_IntegratorCP.mk.*

#### See also

- *Introduction to Flash programming with RealView Debugger* on page 6-2
- *Summary of files used to program custom Flash types on custom platforms* on page 6-6.

### 6.1.2 Summary of files used to program custom Flash types on custom platforms

If your development platform and Flash type are not one of those provided with RealView Debugger, then you must create the following files to program Flash:

`b_flashwrapper.s`

This is an assembly wrapper that acts as an interface between the RealView Debugger API and your Flash programming code, for example `b_IntegratorCP.s`.

`flash.h`

This is a C header file containing prototypes for the functions called by RealView Debugger.

`flash_algorithm.c` **or** `f_flash_algorithm.s`

This contains your Flash programming routines, which are called by the related routines in `b_flashwrapper.s`.

`flash-level.ame`

This ASCII METHOD file describes the Flash memory for the target.

`board-level.ame`

This ASCII METHOD file describes the board for the target.

`board.bcd`

A Board/Chip Definition file that describes the memory map for your custom development platform, and specifies the Flash METHOD file that RealView Debugger is to use to program the Flash device.

The C, assembler, and ASCII METHOD files are required to generate the Flash METHOD file.

#### See also

- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23
- *Creating the Flash-level and board-level AME files* on page 6-29
- *Creating a BCD file* on page 6-39.

## 6.2 RealView Debugger files used for Flash programming

RealView Debugger uses information provided in various target-specific files when programming Flash. These files are described in the following sections:

- *Flash-level code*
- *Board-level code*
- *ASCII MMethod file* on page 6-8
- *Flash MMethod file* on page 6-8
- *Board Chip Definition file* on page 6-9.

RealView Debugger provides source code, AME, FME, and make files for supported targets in `install_directory\RVD\Flash\...\platform`.

If your target is not one of those provided, you must create your own versions of these files. The procedure to do this is described in *Programming Flash for a custom development platform* on page 6-16.

The files for the ARM Evaluator-7T platform are in:

`install_directory\RVD\Flash\...\platform\eval7t`.

### 6.2.1 Flash-level code

Flash-level code is usually written as assembly code to provides the algorithms that are required to access Flash. The Flash-level assembly files provided with RealView Debugger are named `f_flashtype.s`.

---

#### Note

These algorithms are designed to run under RealView Debugger control, and are not intended to form the basis of standalone Flash programming code. For more details on developing standalone Flash programming code, see the *ARM Application Note 111 Flash Programming*. You can obtain this Application Note from the ARM web site <http://www.arm.com>.

---

#### Evaluator-7T example

The Flash-level file provided for the ARM Evaluator-7T is named `f_amd_sst_arm.s`.

#### See also

- *RealView Debugger files used for Flash programming.*

### 6.2.2 Board-level code

Board-level code is usually written as assembly code, and must include the Flash-level assembly code using the `INCLUDE` directive. It must perform any board-specific operations that are required to access Flash, such as unlock operations.

The board-level assembly files provided with RealView Debugger are named `b_board.s`.

#### Evaluator-7T example

The board-level file provided for the ARM Evaluator-7T is named `b_evaluator7t.s`.

#### See also

- *RealView Debugger files used for Flash programming*

- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20.

### 6.2.3 ASCII Method file

*ASCII Method* (AME) files describe the Flash memory for a target, and have the extension `.ame`.

A target must have:

- A Flash-level AME file. The files provided with RealView Debugger are named `flash_flashtype.ame`.
- A board-level AME file. The files provided with RealView Debugger are named `board_flashtype.ame`.

#### Evaluator-7T example

The AME files provided for the Evaluator-7T are named:

```
board_amd_eval7t.ame
board_sst_eval7t.ame
flash_sst_eval7t.ame
flash_amd_eval7t.ame
```

---

#### Note

Four files are provided because there are two different versions of this board, each with a different flash device.

---

#### See also

- *RealView Debugger files used for Flash programming* on page 6-7
- *Flash-level AME file format* on page 6-29
- *Board-level AME file format* on page 6-31.

### 6.2.4 Flash Method file

A *Flash Method* (FME) file is produced from a standard axf image using a special RealView Debugger utility called `pakflash`. This image is created from the assembly code files and the AME files.

It combines textual descriptive information (in AME files) about the Flash devices on your target with an appropriate code algorithm for reading, writing and erasing. This code runs on the target (under RealView Debugger control) when you select options in the RealView Debugger Flash Memory Control dialog.

FME files have the extension `.fme`. The files provided with RealView Debugger are named `flash_boardname.fme`.

#### Evaluator-7T example

The FME files provided for the ARM Evaluator-7T are named:

```
flash_amd_eval7t.fme
flash_sst_eval7t.fme
```

#### See also

- *RealView Debugger files used for Flash programming* on page 6-7.
- *Flash-level code* on page 6-7

- *ASCII MMethod file* on page 6-8.

### 6.2.5 Board Chip Definition file

*Board Chip Definition* (BCD) files add extended target visibility to RealView Debugger. As a minimum the BCD file must:

- specify where the Flash is located in your memory map
- reference an appropriate FME file.

BCD files have the extension `.bcd`. The files provided with RealView Debugger are named *boardname.bcd*, and are located in your default settings directory:

C:\Documents and Settings\userID\Local Settings\Application  
Data\ARM\rvdebug\version\shadowbase\etc

#### Evaluator-7T example

The BCD file provided for the ARM Evaluator-7T is named `Eva17T.bcd`.

#### See also

- *RealView Debugger files used for Flash programming* on page 6-7
- *Flash MMethod file* on page 6-8
- *Creating a BCD file* on page 6-39.

## 6.3 pakflash utility command syntax

The pakflash utility packs an Flash M<sup>E</sup>thod file for RealView Debugger to use when programming Flash.

The command syntax is:

```
pakflash [ -f name ] axf-file.axf -o FME-filename.fme [ -p ] [ -s addr ] [ -r ] [ -d ]
[AME-file.ame]
```

where:

**-f *name***      The board name assigned to the BOARD= setting in the board-level ASCII M<sup>E</sup>thod file.

If you do not specify this option, *name* is assumed to be an entry [FLASH=DEFAULT] in the ASCII M<sup>E</sup>thod file.

***axf-file***      The name of the ELF/DWARF (.axf) image containing the Flash programming routines.

***FME-filename*** The generated Flash M<sup>E</sup>thod file name. The default is *axf-file.fme*.

**-p**              Indicates PC-relative routines.

**-s *addr***      The start address. This also implies -p.

This option overrides reloc.start\_addr= setting in the board-level AME file, and the --ro\_base linker option.

**-r**              The routine is already in ROM, so get the information only.

**-d**              Indicates debug. Therefore, dump information on the header that is created.

***AME-file***      The ASCII M<sup>E</sup>thod file.

If you do not specify an ASCII M<sup>E</sup>thod file, the method file name is assumed to be the same as the .axf file name, but with the .ame extension.

See also:

- *Creating the Flash-level and board-level AME files* on page 6-29
- *Board-level AME file format* on page 6-31
- *Specifying the linker options* on page 6-36.

## 6.4 Programming Flash on the ARM development boards

RealView Debugger provides Flash programming files for the following ARM development boards:

- ARM AEB01
- ARM Evaluator-7T
- ARM Integrator/AP
- ARM Integrator/CP
- ARM VPB926EJ-S (Versatile Platform for ARM926EJ-S).

All files required to program Flash on the ARM development boards are provided with RealView Debugger (see *Summary of files used to program Flash on supported development platforms* on page 6-2).

---

### Note

---

If you have another target board with a standard AMD, ATMEL, Intel, or SST Flash device you must create a board-specific assembler file and link that file to create an FME file before you can program the Flash memory. If you are using another type of Flash memory, you must also create the Flash programming routines. See *Programming Flash for a custom development platform* on page 6-16 for more details.

---

The following sections describe how to program Flash on the ARM development boards, using the Integrator/AP board as an example:

- *About the ARM Integrator/AP board*
- *Assigning a BCD file to a Debug Configuration*
- *Reviewing the information contained in the Integrator/AP BCD file* on page 6-12
- *Displaying the memory map in the Process Control view* on page 6-13
- *Programming an image into Flash on the Integrator/AP* on page 6-14

### 6.4.1 About the ARM Integrator/AP board

The ARM Integrator/AP board includes 32MB of Flash memory for user applications. This is located at address 0x24000000.

---

### Note

---

If you program the Flash on an Integrator using this release of RealView Debugger, you bypass the AFS Flash library system information blocks. These blocks are used by the AFS Flash Library and are stored at the end of each image written to Flash. If you rely on these blocks to keep track of what is in the Flash memory of your target, keep a record of the state and recreate it after trying the example.

---

### See also

- *Programming Flash on the ARM development boards.*

### 6.4.2 Assigning a BCD file to a Debug Configuration

To assign the BCD file for the Integrator/AP board to a Debug Configuration:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Select **Configuration** from the Grouped By list.

3. Expand the Debug Interface containing the Debug Configuration to which you want to assign the BCD file. For example, expand RealView ICE.
  4. Make sure that all targets are disconnected in the Debug Configuration.
  5. Right-click on the Debug Configuration to display the context menu. For example, right-click on RVI.
  6. Select **Properties...** from the context menu to display the Connection Properties window. The settings group for the chosen Debug Configuration is selected (CONNECTION=RVI in this example).
  7. In the right pane, right-click on BoardChip\_name to display the context menu.
  8. Select the required Board/Chip definition from the context menu.  
For example, select **AP** to assign the Board/Chip definition for the Integrator/AP board. A new entry is displayed in the right pane with an asterisk \* beside it (that is \*BoardChip\_name AP).
- **Note** ————
- If the required Board/Chip definition is not shown in the list, select **<More...>** from the context menu. A list dialog box is displayed, which lists all available Board/Chip definitions.
- 
9. Select **Save and Close** from the **File** menu to save your changes and close the Connection Properties window.

#### See also

- *Programming Flash on the ARM development boards* on page 6-11.

### 6.4.3 Reviewing the information contained in the Integrator/AP BCD file

The BCD file for the Integrator/AP platform is already set up for you. You might want to review the settings in the BCD to see how they are used. To do this:

1. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
2. Expand the (\*.bcd) Board/Chip Definitions entry.
3. Right-click on the ...\AP.brd entry to display the context menu.
4. Select **Expand whole Tree** from the context menu.  
The Memory\_block entry has many memory areas defined, included one for Flash. Each entry corresponds to an entry in the memory map.
5. Select the \*M\_FLASH entry.  
The Start, Length, and Flash\_type entries are those used by RealView Debugger when you program Flash memory.
6. Select **Close Window** from the **File** menu to close the Connection Properties window without making any changes.

#### See also

- *Programming Flash on the ARM development boards* on page 6-11
- *Displaying the memory map in the Process Control view* on page 6-13.



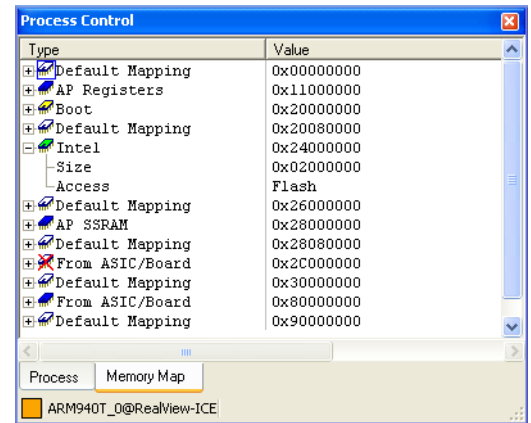
#### 6.4.4 Displaying the memory map in the Process Control view

When you connect to the target to which you have assigned a BCD file, you can display the memory map in the **Memory Map** tab of the Process Control view.

To display the memory map:

1. Connect to a target in the Debug Configuration to which you have assigned a BCD file.
2. Click the **Cmd** tab in the Output view.  
The information message ...Local Advanced\_info, BOARD=AP in this view shows that RealView Debugger is using the Integrator/AP BCD file (AP.bcd). As a result, the memory map contains the definitions required to use the Flash memory on the Integrator/AP board.
3. Select **Memory Map Tab** from the **View** menu to view the new memory map before loading an image.

Figure 6-1 shows the Flash memory as defined on the Integrator/AP board, and that it is using an Intel flash device.



**Figure 6-1 New memory map in the Process Control view**

#### Note

The Flash\_type setting in the connection properties for the Flash memory area definition determines whether or not the Flash Memory Control dialog box is displayed.

#### See also

- *Programming Flash on the ARM development boards* on page 6-11
- *Describing the memory map* on page 6-40
- the following in the *RealView Debugger User Guide*:  
— Chapter 9 *Mapping Target Memory*.

### 6.4.5 Programming an image into Flash on the Integrator/AP

To program an image into Flash, you request RealView Debugger to write to the Flash memory region that you have defined by using the Integrator/AP BCD file. The Integrator Flash starts at memory address 0x24000000, so to write an image to Flash:

1. If required, create an image file compiled to run with code at 0x24000000 and that has data in RAM.

For example, rebuild the dhrystone image using the following linker options:

```
--ro_base 0x24000000 --rw_base 0x8000
```

2. Select **Load Image...** from the **Target** menu to display the Select Local File to Load: dialog box.
3. Locate the required image file, and click **Open**. The Flash Memory Control dialog box is displayed, shown in Figure 6-2. RealView Debugger queues the image in preparation for writing to Flash.

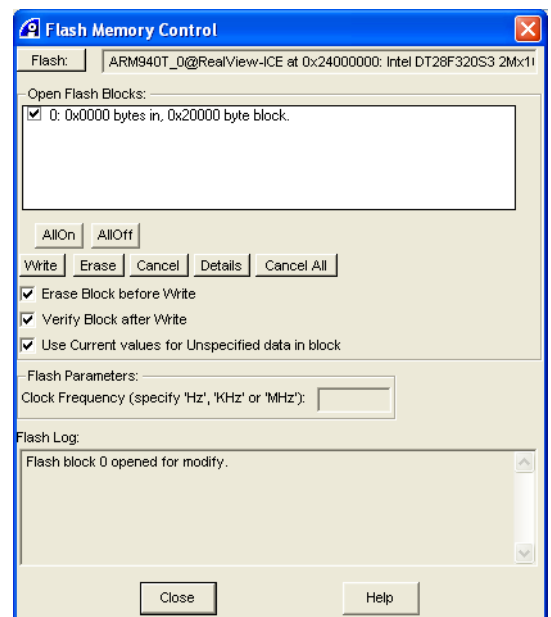


Figure 6-2 The Flash Memory Control dialog box

4. If you are updating part of the Flash block and you want to retain the current values in the rest of the block, select the **Use Current values for Unspecified data in block** check box. This check box is selected by default.

———— **Note** ————

When selected, RealView Debugger reads and then writes the entire block. This might take some time to complete.

5. The Clock Frequency field is enabled if it is required by your Flash device. This has the format  $f$ [Hz|kHz|MHz].

Enter the clock frequency  $f$  as a positive floating point number, for example, 14.175MHz. The unit specifier is optional, and defaults to Hz.

———— **Note** ————

This field is enabled only if the board-level AME file that you used to create the FME file contains the line `needs_clock=true`.

6. Click **Write** to commit the image into Flash.

You must wait for the Flash routine to load and run, that is wait until the last block has been written before continuing (shown in Figure 6-3).

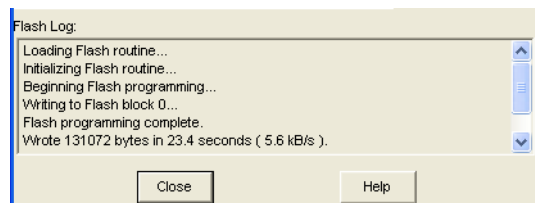


Figure 6-3 Flash programming complete

7. Click **Close** to close the Flash Memory Control dialog box.

The memory map in the **Memory Map** tab is updated to show the details of the image, shown in Figure 6-4.

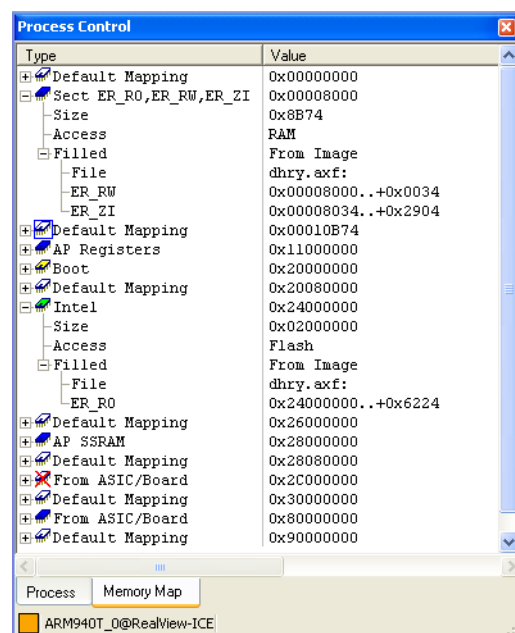


Figure 6-4 Flash image details in memory map

#### Note

Be aware that if you reload an image that resides in Flash, then RealView Debugger re-displays the Flash Memory Control dialog box. However, RealView Debugger resets the PC to the image entry point, and you do not have to rewrite the image to Flash. Click **Close** to close the Flash Memory Control dialog box. Alternatively, if you reset the PC to the image entry point, this is avoided.

#### See also

- *Programming Flash on the ARM development boards* on page 6-11
- *Board-level AME file format* on page 6-31
- the following in the *RealView Debugger User Guide*:
  - Chapter 4 *Loading Images and binaries*.

## 6.5 Programming Flash for a custom development platform

If you are not using an ARM development board, then suitable BCD and FME files are not provided with RealView Debugger. Therefore, you must create your own BCD and FME files.

See also:

- *Third-party support for ARM processor-based development platforms*
- *Programming a Flash type supported by RealView Debugger*
- *Programming a custom Flash type* on page 6-17.

### 6.5.1 Third-party support for ARM processor-based development platforms

If you are using a standard development platform from a third party supplier, your supplier might have suitable BCD and FME files for your development board. Therefore, contact your supplier before you create your own files.

**See also**

- *Programming Flash on the ARM development boards* on page 6-11.

### 6.5.2 Programming a Flash type supported by RealView Debugger

If your hardware uses a Flash type that is also supported by an ARM development board, then you can use the appropriate Flash-level assembly code that is provided with RealView Debugger. These files are:

```
f_amd_sst_arm.s
f_atmel_arm.s
f_intel_arm.s
f_toshiba_arm.s
f_st30_arm.s
```

You must provide all the other files required to program the Flash.

#### Preparing to follow the procedure for a supported Flash type

If you want to use the Evaluator-7T example to follow the procedure in *Procedure for programming a supported Flash type*, do the following:

1. Create a new directory called `eval7t_support`.
2. Copy the following files from the `install_directory\RVD\Flash\...\platform\eval7t` directory into the `eval7t_support` directory:
  - `f_amd_sst_arm.s`
  - `flash_sst.ame`
  - `board_sst_eval7t.ame`
3. Where the procedure directs you to create other required files, create the files in the `eval7t_support` directory.

#### Procedure for programming a supported Flash type

The procedure for programming a Flash type supported by RealView Debugger for your own development platform involves the following steps:

1. Gather information about your target.
2. Create the Flash-level and board-level AME files.

3. Generate the FME file. To do this, create a make file to:
  - build an ELF/DWARF (.axf) image from the Flash algorithm code
  - run the pakflash utility to generate the FME file from the axf image and the AME files.
4. Check the FME file with the dispflash utility.
5. Create a BCD file.
6. Program your Flash device.

#### See also

- *Summary of files used to program Flash on supported development platforms* on page 6-2
- *Programming Flash on the ARM development boards* on page 6-11
- *Gathering information about your development platform* on page 6-19
- *Creating the Flash-level and board-level AME files* on page 6-29
- *Generating the FME file* on page 6-34
- *Checking the FME file with the dispflash utility* on page 6-38
- *Creating a BCD file* on page 6-39
- *Programming an image into Flash* on page 6-44.

### 6.5.3 Programming a custom Flash type

If your hardware does not use a Flash type that is supported by an ARM development board, then you must provide your own Flash-level code, in addition to the other files required to program your Flash type.

#### Preparing to follow the procedure for a custom Flash type

You can use the Evaluator-7T example when following the procedure in *Procedure for programming a custom Flash type*. To do this:

1. Create a new directory for your custom Flash type. However, for this example, create a directory called eval7t\_custom.
2. You must create your own flash AME files, and put them in your custom directory. However, for this example, copy the following files from the `install_directory\RVD\Flash\...\platform\eval7t` directory into the eval7t\_custom directory:
  - flash\_sst.ame
  - board\_sst\_eval7t.ame.
3. Where the procedure directs you to create the required files, create the files in the eval7t\_custom directory.

#### Procedure for programming a custom Flash type

The procedure for programming a custom Flash type for your own development platform involves the following steps:

1. Gather information about your development platform and its components.
2. Create the Flash algorithms for the required Flash type.
3. Create the Flash-level and board-level AME files.

4. Generate the FME file. To do this, create a make file to:
  - build a ELF/DWARF (.axf) image from the Flash algorithm code
  - run the pakflash utility to generate the FME file from the axf image and the AME files.
5. Check the FME file with the dispflash utility.
6. Create a BCD file.
7. Program your Flash device.

**See also**

- *Summary of files used to program custom Flash types on custom platforms* on page 6-6
- *Programming Flash on the ARM development boards* on page 6-11
- *Gathering information about your development platform* on page 6-19
- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23
- *Creating the Flash-level and board-level AME files* on page 6-29
- *Generating the FME file* on page 6-34
- *Checking the FME file with the dispflash utility* on page 6-38
- *Creating a BCD file* on page 6-39
- *Programming an image into Flash* on page 6-44.

## 6.6 Gathering information about your development platform

To provide Flash support for your development platform, you must have the following information:

- a copy of the datasheet for the Flash device used on your development platform
- a memory map of your development board that shows where the Flash and RAM are located
- details of any board-specific code that might be relevant.

See also:

- *Evaluator-7T example.*

### 6.6.1 Evaluator-7T example

The Evaluator-7T board has an SST39VF400A Flash device fitted. The device has 512KB of memory, and a sector size of 4KB.

The memory map for the Evaluator-7T board is shown in Table 6-2.

**Table 6-2 Evaluator-7T memory map**

Address range	Size	Description
0x00000000 to 0x0003FFFF	256KB	SRAM bank1
0x00040000 to 0x0007FFFF	256KB	SRAM bank2
0x01800000 to 0x0187FFFF	512KB	Flash
0x03FE0000 to 0x03FE1FFF	8KB	Internal SRAM

On the Evaluator-7T development board:

- The bottom of 128Kb of Flash contains the bootstrap loader and Angel.

**Note**

It is not advisable to reprogram this area, because without this boot code, the board's memory map is not set up, and it can be difficult to recover.

- The available Flash address range for application code and data is 0x01820000 to 0x0187FFFF.
- No board-specific code (such as unlock codes for specific memory regions) is required to access Flash.

**See also**

- *Gathering information about your development platform.*

## 6.7 Creating algorithms for a Flash type supported by RealView Debugger

If your development platform uses one of the Flash types supported by RealView Debugger, then you must create the board-level assembly code.

See also:

- *The b\_flashwrapper.s template file*
- *Editing b\_flashwrapper.s* on page 6-22
- *Settings for the Evaluator-7T* on page 6-22.

### 6.7.1 The b\_flashwrapper.s template file

A template board-level assembly code file (b\_flashwrapper.s), complete with comments. Example 6-1 shows the b\_flashwrapper.s template.

You might want to rename your version of b\_flashwrapper.s file to something more meaningful.

#### ———— Note ————

The file b\_flashwrapper.s is also available in *Application Note 110 Flash Programming with RealView Debugger*, which can be found from the Documentation link on the ARM web site <http://www.arm.com>.

This code is designed to be linked to reside in a contiguous area of RAM on your target. Therefore, you must specify only an RO base in the linker options.

This does not have to be free scratch memory. By default, RealView Debugger saves the contents of affected RAM and restores it after the Flash operation has completed.

#### Example 6-1 b\_flashwrapper.s template

---

```

;*****/
; b_flashwrapper.s
; Sample board level wrapper code for RVD flash algorithms
;
; 27/3/2003 ARM Ltd.
; Revision: A
; Date: 21/03/2003
;
;*****/

;*****/
;* DEFINE MACROS AND EQUATES */
;*****/
P_WIDTH EQU    NumberofFlashROMs ; number of FlashROMs in parallel across bus.

WIDTH EQU    FlashWidth ; Define resultant width of flash.
; 32bit=4, 16bit=2, byte=1.

;*****/
;* DEFINE SMALL STACK AREA */
;*****/
AREA FLASH_STACK, CODE
DCD 0,0,0,0,0,0,0,0,0,0,0,0 ; 12 words for call depth
stack_top
;*****/

```

---



```

;*****/
;* FLASH_init - initialize the board (memory access controls, etc). */
;*****/
; This function is exported and called directly by RVD.
;
; Should contain any board specific initialisation code.
;
; RVD API register usage as follows
; In: R1 will contain base of flash
; Out: R0=status code (0=OK, else error)
; Scratch registers available for use in your code: R6,R7,R8.
;*****/
        AREA    FLASH_TEXT, CODE, READONLY

        EXPORT    FLASH_init
        ENTRY

FLASH_init

        ldr SP, = stack_top

; ----- */
; Your board specific code goes below
;
; Examples include disabling watchdogs, enabling
; chip selectors/enables to allow write to this memory, etc.
; ----- */

; not required for ARM Evaluator7T board

; place_your_code_here

; ----- */
; end of board specific code
; ----- */
        ;bl      check_status          ; check Flash status and return in R0
                                           ; not used for SST flash

        b        Local_init           ; finish init, then Stop on breakpoint

;*****/
;* Include the appropriate Flash algorithm here to do actual work */
;*****/

        INCLUDE flash_algorithm_assembly_code ; include appropriate flash algorithm

;*****/
;* DEFINE BUFFER FOR WRITE/VERIFY */
;*****/

        AREA BUFFER, NOINIT
buffer % Size ; define area of RAM for RVD to buffer data in
           ; prior to writing to flash

        END

```

### See also

- *Introduction to Flash programming with RealView Debugger* on page 6-2
- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20.

## 6.7.2 Editing b\_flashwrapper.s

The areas of `b_flashwrapper.s` that you can edit are (see Example 6-1 on page 6-20):

### P\_WIDTH and WIDTH equates

You must set *NumberofFlashROMs* and *FlashWidth* to match your target.

### FLASH\_init routine

The `FLASH_init` label is exported to enable the routine to be called directly by RealView Debugger.

Executing this function ultimately results in a branch to a label `FLASH_break`. RealView Debugger automatically places a breakpoint at `FLASH_break` so that it can halt the target while it displays the Flash Memory Control dialog.

If you want to insert your own code, replace the comment *place\_your\_code\_here*. For example, you might want to include code to disable watchdogs.

### ———— Note ————

Only registers `r6`, `r7` and `r8` are available as scratch registers.

### buffer % Size

Buffer defines an area of uninitialized RAM where RealView Debugger can store data before writing it to Flash. *Size* is not Flash dependent and does not usually require editing. However, for large Flash memory devices an increased RAM buffer size might improve write speeds.

### INCLUDE flash\_algorithm\_assembly\_code

*flash\_algorithm\_assembly\_code* identifies the assembly file that contains the appropriate Flash algorithm.

### See also

- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20.

## 6.7.3 Settings for the Evaluator-7T

For the Evaluator-7T example, rename `b_flashwrapper.s` to `b_flashwrapper_eval7t.s`, and set the values shown in Table 6-3.

**Table 6-3 Evaluator-7T example settings**

Setting	Value
<i>numberofFlashROMs</i>	1
<i>FlashWidth</i>	2
<i>place_your_code_here</i>	no code is required
<i>Size</i>	1024
<i>flash_algorithm_assembly_code</i>	<code>f_amd_sst_arm.s</code>

### See also

- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20.

## 6.8 Creating algorithms for a Flash type not provided with RealView Debugger

If your development platform uses a Flash device that cannot be programmed using the Flash algorithms provided with RealView Debugger, you must provide your own Flash programming code.

See also:

- *Basic Procedure*
- *C source file containing the Flash-specific C functions* on page 6-24
- *C header file* on page 6-25
- *Assembly wrapper* on page 6-26.

### 6.8.1 Basic Procedure

To provide your own Flash programming code:

1. Create a C source file containing the Flash-specific C functions.  
For example, you might provide these functions in a file `myflash_sst.c`.
2. Create the following files:
  - `flash.h`.  
This is a C header file containing prototypes for the functions called by RealView Debugger.
  - `rvd2apcs.s`.  
This is an assembly wrapper that acts as an interface between the RealView Debugger API and your Flash programming code.

---

#### Note

The files `rvd2apcs.s` and `flash.h` are also available in *Application Note 110 Flash Programming with RealView Debugger*, which can be found from the Documentation link on the ARM web site <http://www.arm.com>.

---

This code is designed to be linked to reside in a contiguous area of RAM on your development platform. Therefore, you must only specify an RO base in the linker options.

This does not have to be free scratch memory. By default, RealView Debugger saves the contents of affected RAM and restores it after the Flash operation has completed.

#### See also

- *Creating algorithms for a Flash type not provided with RealView Debugger*
- *ARM® Compiler toolchain Using the Linker*
- *ARM® Compiler toolchain Linker Reference*.

## 6.8.2 C source file containing the Flash-specific C functions

Example 6-2 shows a template C source file for defining your Flash-specific C functions.

**Example 6-2 myflash.c**

---

```
#include "flash.h"

/*****
 * Initialise the Flash device
 *****/
UINT32 RVDFlash_Init(UINT32 base_of_flash)
{
    // Place your initialization code here

    return 0;    // success
}

/*****
 * Erase the Flash memory
 *****/
UINT32 RVDFLASH_Erase(UINT32 base_address,
                      UINT32 block_count,
                      UINT32 block_size)
{
    // Place your flash erase code here

    return 0;    // success
}

/*****
 * Write to the Flash memory
 *****/
UINT32 RVDFLASH_Write(UINT32 base_address,
                      UINT32 byte_count,
                      UINT32 block_offset,
                      UINT32 from_buffer,
                      UINT32 verify)
{
    // Place your flash write code here

    return 0;    // success
}

/*****
 * Validate the Flash memory
 *****/
UINT32 RVDFLASH_Validate(UINT32 base_address,
                         UINT32 byte_count,
                         UINT32 block_offset,
                         UINT32 from_buffer)
{
    // Place your flash validate code here

    return 0;    // success
}
```

---

**See also**

- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23.

**6.8.3 C header file**

Example 6-3 shows a template C header file (flash.h) containing prototypes for the functions called by RealView Debugger, shown in Example 6-2 on page 6-24.

The flash.h file is available in *Application Note 110 Flash Programming with RealView Debugger*, which can be found from the Documentation link on the ARM web site <http://www.arm.com>.

**Example 6-3 flash.h**


---

```

/*****
 * Flash.h
 * Contains function prototypes for flash specific functions

 * 27/3/2003 ARM Ltd.
 * Revision: A
 * Date: 21/03/2003
 *****/

typedef unsigned long int UINT32;

UINT32 RVDFlash_Init(UINT32 base_of_flash);

UINT32 RVDFLASH_Erase(UINT32 base_address,
                      UINT32 block_count,
                      UINT32 block_size);

UINT32 RVDFLASH_Write(UINT32 base_address,
                      UINT32 byte_count,
                      UINT32 block_offset,
                      UINT32 from_buffer,
                      UINT32 verify);
UINT32 RVDFLASH_Validate(UINT32 base_address,
                        UINT32 byte_count,
                        UINT32 block_offset,
                        UINT32 from_buffer);

/*****
 * base_of_flash: base memory address of flash device
 * base_address: base memory address of first flash block
 * block size: size of a single flash block in bytes
 * block count: number of flash blocks
 * byte_count: number of bytes to read/write
 * block_offset: offset into block (in bytes) to start read/write at
 * from_buffer: address of buffer to copy from
 * verify: 0 if no verify requested, else same value as byte_count
 * return 0=OK, else error.
 *****/

```

---

**See also**

- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23.

## 6.8.4 Assembly wrapper

The assembly wrapper, `b_flashwrapper.s`, acts as an interface between the RealView Debugger API and your Flash programming code.

Example 6-4 shows a template for an assembly wrapper called `rvd2apcs.s`. The `rvd2apcs.s` file is available in *Application Note 110 Flash Programming with RealView Debugger*, which can be found from the Documentation link on the ARM web site <http://www.arm.com>.

### Example 6-4 rvd2apcs.s

---

```

;*****
; RVD2APCS
; This code is intended as an Interface between the RVD API and your flash
; algorithm code.
;
; 27/3/2003 ARM Ltd.
; Revision: A
; Date: 21/03/2003

;*****
; You must provide flash specific implementations of the exported functions
;
; Please refer to the prototypes for these functions provided in the header
; file flash.h
; The code below ensures the correct information provided by RVD is passed
; to your functions using the normal ATPCS registers.
; After calling each of these functions RVD will halt the processor and
; return control to the host by placing a breakpoint at the label
; FLASH_break

;*****

EXPORT      FLASH_init
EXPORT      FLASH_erase
EXPORT      FLASH_write
EXPORT      FLASH_validate
EXPORT      FLASH_break

AREA FLASH, CODE, READONLY

;*****
; FLASH_init - Initialise any board specific memory access controls, etc.
;*****

CODE32
PRESERVE8

ENTRY

FLASH_init

IMPORT RVDFlash_Init
LDR sp, =stacktop
MOV r0, r1 ; base memory address of flash device
BL RVDFlash_Init ; branch to customer function
B FLASH_break ; return

;*****
; FLASH_erase - erase a Flash block(s)

```

---

```
*****
```

#### FLASH\_erase

```
IMPORT RVDFLASH_Erase ; i is in r0
LDR sp, =stacktop
; STR lr, [sp, #-4]! ; preserve lr
MOV r0, r1 ; base memory address of first flash block
MOV r1, r2 ; number of flash blocks
MOV r2, r3 ; size of a single flash block in bytes
BL RVDFLASH_Erase ; branch to customer function
B FLASH_break ; return
```

```
*****
```

```
; FLASH_write - write data to a Flash block
```

```
*****
```

#### FLASH\_write

```
IMPORT RVDFLASH_Write ; write with image
LDR sp, =stacktop
MOV r0, r1 ; base memory address of first flash block
MOV r1, r2 ; number of bytes to read/write
MOV r2, r4 ; offset into block (in bytes) to start read/write at
MOV r3, r5 ; address of buffer
STR r9, [sp, #-4]! ; verify flag
BL RVDFLASH_Write ; branch to customer function
B FLASH_break ; return
```

```
*****
```

```
; FLASH_validate - validate a write to a Flash block
```

```
*****
```

#### FLASH\_validate

```
IMPORT RVDFLASH_Validate ; i is in r0
LDR sp, =stacktop
MOV r0, r1 ; base memory address of first flash block
MOV r1, r2 ; number of bytes to read/write
MOV r2, r4 ; offset into block (in bytes) to start read/write at
MOV r3, r5 ; address of buffer
BL RVDFLASH_Validate ; branch to customer function
B FLASH_break ; return
```

```
*****
```

```
; FLASH_break - end of function entry for all Flash routines *
```

```
*****
```

#### FLASH\_break

```
    nop
forever
    b forever ; should never be reached.
```

```
*****
```

```
; * DEFINE BUFFER FOR WRITE/VERIFY and STACK
```

```
*****
```

```
    AREA BUFFER, NOINIT
buffer % 1024 ; buffer for copying
```

```
    AREA STACK, NOINIT
stackbottom
    % 512
```

```
stacktop  
% 4  
END
```

---

**See also**

- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23.



## 6.9 Creating the Flash-level and board-level AME files

Flash-level and board-level AME files are required in addition to the code that provide the Flash algorithms. This section defines the minimum information that you must specify in the Flash-level and board-level AME files used to describe the Flash memory of your target.

See also:

- *Flash-level AME file format*
- *Board-level AME file format* on page 6-31.

### 6.9.1 Flash-level AME file format

Example 6-5 shows the format of a Flash-level AME file. You can specify multiple FLASH entries in the same AME file, but each entry must have a unique device name. For an example, see:

`install_directory\RVD\F\ash\...\platform\IntegratorCP\flash_intel.ame`

#### Example 6-5 Flash-level AME file format

---

```
[FLASH=DeviceName]
flash_name="description"
width=FlashWidth
# Each Flash has 64 blocks(32K-halfwords x 2) as main blocks
block.GroupID={count=nBlocks:size=BlockSize:swap=False}
...
lock_block=BlockNum
...
proc_name="ARM"
[FLASH=DeviceName]
...
```

---

#### Mandatory settings

The following Flash-level AME settings are mandatory:

`FLASH=DeviceName`

Defines a name for a specific Flash device. This is the name you specify for the `from_flash=` setting in the board-level AME file (see Example 6-7 on page 6-31). The name can have a maximum of 31 characters.

`block.GroupID`

A list of block groups, where *GroupID* is an alphanumeric string (for example, `block.boot` or `block.1`).

You must specify at least one block group. You can specify a maximum of eight block groups.

For each block group you can specify:

`count=nBlocks`

Defines the number of blocks in each block group. You can specify this in:

- decimal (for example, 256)
- hexadecimal, using the 0x prefix (for example, 0x0100).

`size=BlockSize`

The size in bytes of each block. You can specify this in:

- decimal (for example, 131072)
- hexadecimal, using the 0x prefix (for example, 0x20000)
- kilobytes, using the K suffix (for example, 128K).

## Optional settings

The following Flash-level AME settings are optional:

`flash_name="description"`

A text description that is displayed by RealView Debugger.

`lock_block=LockedBlockID`

A list of blocks that are not to be modified by RealView Debugger, where *LockedBlockID* is an alphanumeric string (for example, `block.boot` or `block.1`).

You can specify a maximum of eight lock blocks.

`proc_name` One of the following:

- "CXM" to generate the FME file to be used for programming Flash with a Cortex-M3 processor
- "ARM" to generate the FME file to be used with all other ARM architecture-based processors. This is the default processor name.

`width=FlashWidth`

The width of the Flash device:

- |          |                |
|----------|----------------|
| <b>1</b> | 8bit (default) |
| <b>2</b> | 16bit          |
| <b>4</b> | 32bit          |

———— **Note** —————

The width= setting in the board-level AME file overrides this setting.

## Evaluator-7T example

Example 6-6 shows a sample Flash-level AME file for the Evaluator-7T. This is the file `flash_sst.ame` in the directory you created.

### Example 6-6 Evaluator-7T Flash-level AME file

---

```
[FLASH=SST39VF400A]
flash_name="Silicon Storage Tech 39VF400A"
width=2
no_erase=False
block.1={count=0x080:size=0x0800}
proc_name="ARM"
```

---

## See also

- *Creating the Flash-level and board-level AME files* on page 6-29
- *Preparing to follow the procedure for a supported Flash type* on page 6-16

- *Preparing to follow the procedure for a custom Flash type* on page 6-17.

## 6.9.2 Board-level AME file format

Example 6-7 shows the format of a board-level AME file. You can specify multiple BOARD entries in the same AME file, but each entry must have a unique device name. For an example, see:

*install\_directory\RVD\Flash\...\platform\IntegratorCP\board\_intel\_arm.ame*

### Example 6-7 Board-level AME file format

---

```
[BOARD=BoardName]
proc_name=ARM
from_flash=DeviceName
width=DataWidth
reloc.start_addr=R0baseAddress
reloc.pc_rel=True
needs_clock=false

[BOARD=BoardName]
...

[INCLUDE] flash_files_dir\flash-level.ame
```

---

### Mandatory Settings

The following board-level AME settings are mandatory:

**BOARD=BoardName**

Defines the board name. You must use this name for the BOARD=BoardName entry in the BCD file that defines the Flash memory area.

This is the name you use with the `pakflash -f name` option (see *pakflash utility command syntax* on page 6-10).

**proc\_name** One of the following:

- CXM to generate the FME file to be used for programming Flash with a Cortex-M3 processor
- ARM to generate the FME file to be used with all other ARM architecture-based processors. This is the default processor name.

**from\_flash=DeviceName**

References the relevant Flash device in the Flash-level AME file, and is the same name that you assign to the FLASH= setting (see Example 6-5 on page 6-29). The name can have a maximum of 31 characters.

**[INCLUDE] flash\_files\_dir\flash-level.ame**

Specifies the location and name of the Flash-level AME file. The path is not required if your board-level AME file is in the same location as the Flash-level AME file you are using.

## Optional Settings

The following board-level AME settings are optional:

`needs_clock={true|false}`

Usually, the FME file can program your Flash device directly, without having to know the clock speed of your device. However, if your device must be informed of the clock speed to program Flash correctly, then you must specify `needs_clock=true`. The default is `false` if this setting is not specified.

When `needs_clock` is set to `true`:

- the Clock Frequency field on the Flash Memory Control dialog box is enabled
- you can specify the `clk:(frequency)` qualifier to the FLASH command.

`reloc.pc_rel` Indicates PC-relative code. This is normally set to `True`.

`reloc.start_addr=RObaseAddress`

Specifies the address from which the Flash programming code runs.

This setting overrides the `--ro_base` linker option.

The `pakflash -s addr` option overrides this setting.

### ———— Note ————

This does not have to be free scratch memory. By default RealView Debugger saves the contents of affected RAM and restores it after the Flash operation has completed.

`width=DataWidth`

Resultant data width when writing to the Flash:

- |          |                |
|----------|----------------|
| <b>1</b> | 8bit (default) |
| <b>2</b> | 16bit          |
| <b>4</b> | 32bit          |

If you are programming one of the supported Flash types, this entry must match the `WIDTH` value you defined in the board-level assembly code.

### ———— Note ————

This setting overrides the `width=` setting in the Flash-level AME file.

## Evaluator-7T example

Example 6-8 shows a sample board-level AME file for the Evaluator-7T. This is the file `board_sst_eval7T.ame` in the directory you created.

In your copy of `board_sst_eval7T.ame`, change the `BOARD=` value to that shown in Example 6-8.

In this example there are two 16bit Flash devices in parallel across the data bus. Therefore, the `width = 4` so we write to the bottom 16bits of each Flash simultaneously.

### Example 6-8 Evaluator-7T board-level AME file

---

```
[BOARD=ARM_Eval7T]
proc_name=ARM
# uses AMD AM29LV400
```

```
# or uses SST 39VF400A
from_flash=SST39VF400A
reloc.pc_rel=True
reloc.start_addr=0x40000
[INCLUDE] flash_sst.ame
```

---

### See also

- *Creating the Flash-level and board-level AME files* on page 6-29
- *pakflash utility command syntax* on page 6-10
- *Programming an image into Flash on the Integrator/AP* on page 6-14
- *Preparing to follow the procedure for a supported Flash type* on page 6-16
- *Preparing to follow the procedure for a custom Flash type* on page 6-17
- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20
- *Specifying the linker options* on page 6-36
- *Creating the BOARD group in the BCD file* on page 6-40
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

## 6.10 Generating the FME file

Before you can generate the FME file, you must:

1. Create Flash algorithm code.
2. Create Flash-level and board-level AME files.

The following sections describe how to generate the FME file for the Evaluator-7T Flash example:

- *Locating the source and AME files required*
- *Specifying the compiler options* on page 6-35
- *Specifying the assembler options* on page 6-35
- *Specifying the linker options* on page 6-36
- *Running the pakflash utility* on page 6-36.

### 6.10.1 Locating the source and AME files required

The source files you require depend on whether or not you are using the Flash algorithms that are provided with RealView Debugger:

- If you are using Flash algorithms provided with RealView Debugger, then use the following source files:
  - `b_flashwrapper_eval7t.s`, the board-specific code
  - `flash_algorithm.s`, the Flash algorithm code (for the Evaluator-7T example, use `...\eval7t_support\f_amd_sst_arm.s`).
- If you are using Flash algorithms that you have created, then use the following source files:
  - `rvd2apcs.s`
  - `flash.h`
  - `flash_algorithm.c`, for example `myflash_sst.c`.

You must also locate the AME files you want to use.

#### Evaluator-7T example

For the Evaluator-7T example, use the following AME files:

- `board_sst_eval7t.ame` (board-level AME file)
- `flash_sst.ame` (Flash-level AME file).

#### See also

- *Generating the FME file*
- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20
- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23
- *Creating the Flash-level and board-level AME files* on page 6-29.

## 6.10.2 Specifying the compiler options

If you are using Flash algorithms that you have created, then compile the source file `flash_algorithm.c`.

Specify other options as required. For example, make sure that the endianness is the same as the endianness for your debug target.

### Evaluator-7T example

For the Evaluator-7T example, specify the options:

```
-c --littleend myflash_sst.c -o myflash_sst.o
```

### See also

- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20
- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23
- *Creating the Flash-level and board-level AME files* on page 6-29
- *Locating the source and AME files required* on page 6-34
- *Generating the FME file* on page 6-34
- *ARM® Compiler toolchain Using the Compiler*
- *ARM® Compiler toolchain Compiler Reference.*

## 6.10.3 Specifying the assembler options

Select the source files you are using:

- If you are using Flash algorithms provided with RealView Debugger, then use your version of `b_flashwrapper.s`.

### ———— Note ————

Do not specify the Flash algorithm file, because this is included by `b_flashwrapper.s`.

For the Evaluator-7T example, specify the options:

```
--keep --littleend b_flashwrapper_eval7t.s -o b_flashwrapper_eval7t.o
```

- If you are using Flash algorithms that you have created, then use the source file `rvd2apcs.s`.

For the Evaluator-7T example, specify the options:

```
--keep --littleend rvd2apcs.s -o rvd2apcs.o
```

Specify other options as required. For example, make sure that the endianness is the same as the endianness for your debug target.

### See also

- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20
- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23
- *Creating the Flash-level and board-level AME files* on page 6-29
- *Locating the source and AME files required* on page 6-34
- *Generating the FME file* on page 6-34
- *ARM® Compiler toolchain Using the Assembler*
- *ARM® Compiler toolchain Assembler Reference.*

## 6.10.4 Specifying the linker options

Specify up the linker options:

- If you are using Flash algorithms provided with RealView Debugger, then select your version of `b_flashwrapper.s`:

```
--debug --ro_base address --no_remove -o flash_filename.axf b_flashwrapper.o
```

---

### Note

---

Do not specify the Flash algorithm file, because this is included by `b_flashwrapper.s`.

---

For the Evaluator-7T example, specify the options:

```
--debug --ro_base 0x40000 --no_remove -o Flash_Eval7T.axf b_flashwrapper_eval7t.o
```

- If you are using Flash algorithms that you have created, then use the source file `rvd2apcs.s`:

```
--debug --ro_base address --no_remove -o flash_filename.axf flash_algorithm.o  
rvd2apcs.o
```

For the Evaluator-7T example, specify the options:

```
--debug --ro_base 0x40000 --no_remove -o Flash_Eval7T.axf flash_algorithm.o  
rvd2apcs.o
```

The `--debug` option is optional.

The `--ro_base` setting is overridden by the `reloc.start_addr=` setting in the build-level AME file, and by the `pakflash -s addr` option.

---

### Note

---

You must always use the `--no_remove` option.

---

### See also

- pakflash utility command syntax* on page 6-10
- Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20
- Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23
- Creating the Flash-level and board-level AME files* on page 6-29
- Board-level AME file format* on page 6-31
- Generating the FME file* on page 6-34
- ARM® Compiler toolchain Using the Linker*
- ARM® Compiler toolchain Linker Reference.*

## 6.10.5 Running the pakflash utility

After you have built the `.axf` image file, you must run the `pakflash` utility to create the FME file:

```
'$RVDEBUG_BASE\bin\pakflash' -f BoardName debug\flash_filename.axf  
board-level_AME_filename.ame -o flash_filename.fme
```

**BoardName**     The name assigned to the `BOARD=` in the board-level AME file.

**flash\_filename**

The name of the file that you specified in *Specifying the linker options*.



*board-level\_AME\_filename*

The name of the board-level AME file that you are using.

### Evaluator-7T example

For the Evaluator-7T example, specify:

```
'install_directory\RVD\Core\...\win_32-pentium\bin\bin\pakflash' -f ARM_Eval7T
Flash_Eval7T.axf board_sst_eval7T.ame -o Flash_Eval7T.fme
```

### See also

- *pakflash utility command syntax* on page 6-10
- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20
- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23
- *Creating the Flash-level and board-level AME files* on page 6-29
- *Board-level AME file format* on page 6-31
- *Generating the FME file* on page 6-34
- *Specifying the linker options* on page 6-36
- *ARM® Compiler toolchain Using the Linker*
- *ARM® Compiler toolchain Linker Reference.*

## 6.11 Checking the FME file with the dispflash utility

After you have created the FME file, you can check the contents with the `dispflash` utility. This utility has the syntax:

```
dispflash filename.fme
```

You can run this command at the RealView Debugger CLI as follows:

1. Select the **Cmd** tab in the Output view.
2. Enter the following command:  
`host dispflash path\filename.fme`

See also:

- *Evaluator-7T example.*

### 6.11.1 Evaluator-7T example

For example, to check the contents of the file created in *Generating the FME file* on page 6-34, enter the command:

```
host dispflash Flash_Eval7T.fme
```

This produces the following output:

```
Flash 'Silicon Storage Tech 39VF400A' for processor 'ARM' (Little Endian)
Width=2 with erase value of 0xFFFF
1 Block Groups:
  (0) 128 blocks with byte size 4096/0x1000
Routine Code PC-rel. Can load at 0x40000
Init routine 0x0030 bytes from start
Erase routine 0x007C bytes from start
Erase then Write routine 0x010C bytes from start
Write routine 0x0110 bytes from start
Validate routine 0x015C bytes from start
Breakpoint routine 0x0188 bytes from start
Separate Data image (vars) 0x01CC bytes from start
RAM Buffer at 0x280001CC (page 0) with byte size of 1024
0 Locked blocks
0 Memory Regions to Restore
0 Registers to Restore
```

#### ———— Note ————

You can also display this information in RealView Debugger. To do this, click **Flash:** on the Flash Memory Control dialog box.

#### See also

- the following in the *RealView Debugger User Guide*:  
 — Chapter 6 *Writing Binaries to Flash*.

## 6.12 Creating a BCD file

BCD files are provided for the various ARM development boards. You can create a new BCD from within RealView Debugger by creating a template BCD file, or by modifying an existing BCD file. RealView Debugger searches for BCD files using the search path described in *The RealView Debugger search path* on page 1-17.

---

### Note

---

This section does not describe all aspects of configuring BCD files (such as enumeration of registers). It describes only those aspects required to program the Flash.

---

See also:

- *Basic procedure for the Evaluator-7T example*
- *Saving a copy of the template BCD file*
- *Creating the BOARD group in the BCD file* on page 6-40
- *Describing the memory map* on page 6-40
- *Assigning your board/chip definitions to a Debug Configuration* on page 6-41
- *Viewing the memory map* on page 6-42

### 6.12.1 Basic procedure for the Evaluator-7T example

To create a new BCD file for the Evaluator-7T example follow these steps:

1. Create a template BCD file as described in *Creating a BCD file to use as a template* on page 4-13.
2. Create your BCD file by saving a copy of the template BCD file.
3. Create a new BOARD group in the BCD file.
4. Describe the memory map for your hardware.
5. Assign the new BCD file to the required Debug Configuration.
6. View the new memory map.

#### See also

- *Creating a BCD file to use as a template* on page 4-13
- *Creating a BCD file*
- *Creating the BOARD group in the BCD file* on page 6-40
- *Describing the memory map* on page 6-40
- *Assigning your board/chip definitions to a Debug Configuration* on page 6-41
- *Viewing the memory map* on page 6-42.

### 6.12.2 Saving a copy of the template BCD file

To save a copy of the template BCD file:

1. Create a template BCD file as described in *Creating a BCD file to use as a template* on page 4-13, if you have not already done this.
2. Select **Connection Properties...** from the **Target** menu to display the Connection Properties window.
3. Expand the (\*.bcd) Board/Chip Definitions group.

4. Save a copy of the template BCD file:
  - a. Right-click on the ...\\template.bcd file entry to display the context menu.
  - b. Select **Save As...** from the context menu to display the Enter New Name dialog box.
  - c. Locate your RealView Debugger home directory.
  - d. Enter **ARM\_Eval7T.bcd** for the file name to save.
  - e. Click **Save**. The ARM\_Eval7T.bcd is created in your RealView Debugger home directory. Also, the template BCD file changes to ...\\ARM\_Eval7T.bcd
5. Select **Save Changes** from the **File** menu to save your changes.
6. Select **Refresh** from the **File** menu to refresh the BCD file list.
7. Expand the (\*.bcd) Board/Chip Definitions group. The ...\\ARM\_Eval7T.bcd entry is available.

#### See also

- *Creating a BCD file to use as a template* on page 4-13
- *Creating a BCD file* on page 6-39.

### 6.12.3 Creating the BOARD group in the BCD file

To create the BOARD group in your BCD file:

1. Expand the entry for your BCD file to display the context menu.  
For the Evaluator-7T example, expand the ...\\ARM\_Eval7T.bcd entry.
2. Right-click on BOARD entry in your BCD file to display the context menu.  
For the Evaluator-7T example, right-click on the BOARD=ARM\_TEMPLATE entry.
3. Select **Rename** from the context menu to display the Group Type/Name selector dialog box.
4. Change the name for the group.  
This name must match the BOARD= entry in your board-level AME file.  
For the Evaluator-7T example, change ARM\_TEMPLATE to **ARM\_Eval7T**.
5. Click **OK**. The Group Type/Name selector dialog box closes, and the BOARD=ARM\_TEMPLATE entry changes to \*BOARD=ARM\_Eval7T.
6. Select **Save Changes** from the **File** menu to save your changes.

#### See also

- *Board-level AME file format* on page 6-31
- *Creating a BCD file* on page 6-39.

### 6.12.4 Describing the memory map

To describe the memory map for your hardware:

1. Locate the Memory\_block settings group in your BCD file.  
For the Evaluator-7T example:
  - a. Expand the ...\\ARM\_Eval7T.bcd entry.
  - b. Expand the \*BOARD=ARM\_Eval7T entry.

- c. Expand the Advanced\_Information entry.
  - d. Expand the Memory\_block entry.
2. Create an entry for each area of memory for your board. To do this:
  - a. In the right pane, right-click on the Memory\_block entry to display the context menu.
  - b. Select **Make New...** from the context menu to display the Enter Value dialog box.
  - c. Enter a name for the area of memory you are defining.  
For the Evaluator-7T example, enter **Eval\_App\_Flash**.
  - d. Repeat these steps to create the entries for other memory areas.  
For the Evaluator-7T example, create entries with the following names:
    - **SRAM\_bank1**
    - **SRAM\_bank2**.
3. Set the attributes for each memory area entry you created in the previous step.  
For the Evaluator-7T example, set the attributes as shown in Table 6-4.

**Table 6-4 Memory map attributes**

Attribute	Eval_App_Flash	SRAM_bank1	SRAM_bank2
Start	0x01820000	0x0	0x40000
Length	0x60000	0x40000	0x40000
Access <sup>a</sup>	Flash	RAM	RAM
Flash_type <sup>b</sup>	Location of the Flash_Eval7T.fme file created in <i>Generating the FME file</i> on page 6-34	-	-
Description	Application Flash 384K	SRAM bank1 256K	SRAM bank2 256K
Flash_write_mode <sup>c</sup>	Prompt	-	-

a. The default setting is RAM.

b. When this is set, RealView Debugger displays the Flash Memory Control dialog box when you write to a Flash memory location. For example, when you load an binary image into Flash.

c. The default setting is Prompt.

4. Delete the Default entry:
  - a. Right-click on the Default entry to display the context menu.
  - b. Select **Delete** from the context menu. The Default entry is deleted.
5. Select **Save Changes** from the **File** menu to save your changes.

#### See also

- *Gathering information about your development platform* on page 6-19
- *Creating a BCD file* on page 6-39
- *Memory\_block group settings* on page A-21.

### 6.12.5 Assigning your board/chip definitions to a Debug Configuration

When you have modified your new board/chip definition, you must assign it to the Debug Configuration for your development platform.

---

**Note**

---

This section assumes that you have access to a compatible JTAG debug interface, such as DSTREAM or RealView ICE.

---

To assign a board/chip definition to a connection:

1. Select **Connect to Target...** from the **Target** menu to display the Connect to Target window.
2. Select **Configuration** from the Grouped By list.
3. Expand the Debug Interface containing the Debug Configuration to which you want to assign the new board/chip definition. For example, expand RealView ICE.
4. Make sure that all targets are disconnected in the Debug Configuration.
5. Right-click on the Debug Configuration to display the context menu. For example, right-click on RVI.
6. Select **Properties...** from the context menu to display the Connection Properties dialog box.
7. Click the **BCD files** tab to show the list of available board/chip definition.
8. Select your board/chip definition in the Available Definitions list.  
For the Evaluator-7T example, select ARM\_Eval7T.
9. Click the **Add** button. The ARM\_Eval7T definition is moved to the Assigned Definitions list.
10. Click the **OK** button to save the changes and close the Connection Properties dialog box.

**See also**

- *Creating a BCD file* on page 6-39.

**6.12.6 Viewing the memory map**

To view the memory map:

1. Connect to a target in the Debug Configuration to which you have assigned the new board/chip definition.
2. Select **Memory Map Tab** from the **View** menu of the Code window to display the **Memory Map** tab.
3. Expand the memory map areas you added in *Describing the memory map* on page 6-40. Figure 6-5 on page 6-43 shows an example:

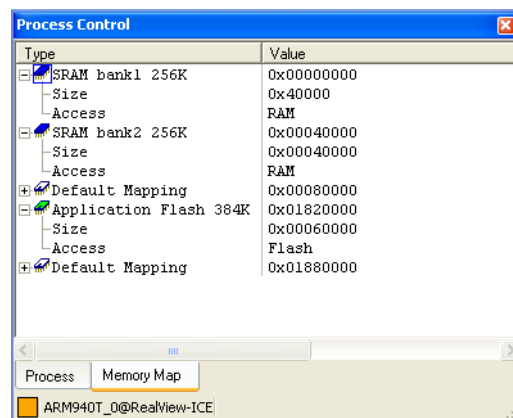


Figure 6-5 Viewing the new memory map

**See also**

- *Creating a BCD file* on page 6-39.

## 6.13 Programming an image into Flash

The following sections describe how to program an image to Flash:

- *Writing the image to Flash*
- *Using CLI commands to program Flash*
- *Checking the contents of Flash* on page 6-45.

### 6.13.1 Writing the image to Flash

To program an image into Flash using RealView Debugger:

1. Assign the board/chip definition that describes the memory map and specifies the FME file to the Debug Configuration.
2. Build your Flash image using the linker command line option `--ro_base`. Make sure that this points to a valid Flash address.
3. Connect to a target in the Debug Configuration.
4. Queue your Flash image for programming into the Flash device. You can do this using one of the following methods:
  - Load a Flash image.
  - Load a Flash binary.
  - Load the Flash image or binary using the Upload/Download file from/to Memory dialog box.

In each case, the Flash Memory Control dialog box is displayed (see Figure 6-2 on page 6-14).

5. The Clock Frequency field is enabled if it is required by your Flash device. The format is  $f[\text{Hz}|\text{kHz}|\text{MHz}]$ .  
Enter the clock frequency  $f$  as a positive floating point number, for example, 14.175MHz. The unit specifier is optional, and defaults to Hz.
6. Click **Write** on the Flash Memory Control dialog box to write the image to the Flash device. RealView Debugger uses the information and Flash routines in the FME file that is specified in the assigned board/chip definition.

#### See also

- *Assigning your board/chip definitions to a Debug Configuration* on page 6-41
- *Programming an image into Flash*
- the following in the *RealView Debugger User Guide*:
  - *Loading an executable image* on page 4-4
  - *Loading a binary* on page 4-12
  - *Specifying the linker options* on page 6-36
  - Chapter 3 *Target Connection*
  - Chapter 6 *Writing Binaries to Flash*.

### 6.13.2 Using CLI commands to program Flash

You can use the following CLI commands to program an image or binary to Flash:

- `LOAD`, to load the Flash image, for example:  
`LOAD 'C:\my_projects\flash\my_flash_image.axf'`
- `READFILE`, to load a binary to Flash, for example:



```
READFILE,raw 'C:\my_projects\flash\my_flash_binary.bin'=0x1820000
```

- FLASH, to perform operations on Flash. For example, to write the image or binary to Flash enter:  
FLASH,write

#### See also

- *Programming an image into Flash* on page 6-44
- the following in the *RealView Debugger Command Line Reference Guide*:  
— *Alphabetical command reference* on page 2-12.

### 6.13.3 Checking the contents of Flash

You can check the contents of Flash as follows:

1. Make sure that the Memory view is visible. If it is not visible, select **Memory** from the **View** menu.
2. In the Memory view, right-click on an address in the left-hand column to display the context menu.
3. Select **Set Start Address...** from the context menu.
4. Enter the start address of your Flash memory.
  - for the Evaluator-7T example, enter 0x1820000
  - for the Integrator/AP, enter 0x24000000.

The contents of memory starting at this address are displayed in the Memory view. The contents of Flash are shown in green.

#### See also

- *Programming an image into Flash* on page 6-44
- the following in the *RealView Debugger User Guide*:  
— *Writing to specific locations in Flash memory* on page 6-6.

## 6.14 Troubleshooting

If you are having problems programming an image to Flash, check the following:

- Does the board/chip definition correctly describe the Flash memory area for your board, and does it specify the correct FME file?
- Does your Debug Configuration have the correct board/chip definition assigned to it?
- Is the icon for the Flash area in the **Memory Map** tab green, and are the Flash memory locations in the Memory view shown with a yellow background?  
This indicates that the Flash\_type setting does not point to a valid FME file, or is not set.
- Is the image you are loading a valid Flash image?  
Use the `dispf1ash` utility to display information about the FME file, and examine the following line:  
Routine Code PC-rel. Can load at *address*  
The *address* must be a valid RAM area on your board. If it is not, then rebuild the FME file with the `--ro_base address` linker option.
- Does your Flash image have the same endianness as the FME file?
  - To check the endianness of the FME file, use the `dispf1ash` command described in .
  - Rebuild the image using the compiler and assembler options `--littleend` or `--bigend` as appropriate.
  - Make sure that your targets and Debug Configuration have the correct endianness assigned.
- If you have created your own FME file:
  - Have you correctly specified the Flash and board settings in the AME files?
  - Are your Flash algorithms correct?

See also:

- *Creating algorithms for a Flash type supported by RealView Debugger* on page 6-20
- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 6-23
- *Creating the Flash-level and board-level AME files* on page 6-29
- *Checking the FME file with the dispf1ash utility* on page 6-38
- *Creating a BCD file* on page 6-39
- *Assigning your board/chip definitions to a Debug Configuration* on page 6-41
- *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 the Linker*
- *ARM® Compiler toolchain Linker Reference.*

# Appendix A

## Connection Properties Reference

This appendix contains reference details about board file entries that define target configurations and custom connections. It contains the following sections:

- *About connection properties reference* on page A-2
- *Debug Configuration generic groups and settings* on page A-6
- *Debug Configuration Advanced\_Information settings reference* on page A-10
- *Memory mapping Advanced\_Information settings reference* on page A-20.

## A.1 About connection properties reference

To view the settings described in this appendix, you must display the Connection Properties window. See *Displaying the Connection Properties window* on page 3-6 for details.

This appendix assumes that you are using the RealView Debugger base product that includes built-in configuration files to enable you to make a connection. If you have changed these files, or created new configuration files of your own, your Connect to Target window and Connection Properties window might look different to those shown in this appendix.

Connection properties are configured using different types of entry in the board file.

See also:

- *The CONNECTION group*
- *The DEVICE group* on page A-3
- *The BOARD, CHIP, and COMPONENT groups* on page A-3
- *Relationship between connections, boards, and chips* on page A-4

### A.1.1 The CONNECTION group

The CONNECTION group identifies a Debug Configuration that is to be used to provide the connections to the targets on your development platform.

It is suggested that you do not change the name of a CONNECTION group in the Connection Properties window. To rename a Debug Configuration, see *Changing the name of a Debug Configuration* on page 3-17 in the *RealView Debugger User Guide*.

The settings in this group specify:

- the Debug Interface used to access your development platform
- a target-specific configuration file that identifies:
  - the position of each target in the scan chain
  - the name of each target on your development platform.
- the connection-specific debugging environment how the connection is to be made to each target
- one or more *Board/Chip Definition* (BCD) files that define the memory map configuration of your development platform.

In some forms of JTAG file additional information such as speed adjust can also be specified. Using a CONNECTION group automatically pulls the list of targets from the named configuration file and provides an easy way to keep the two locked together.

---

#### **Note**

---

A CONNECTION group must not be used in a BCD file.

---

### Entries in the CONNECTION group

The following entries are available:

- Connect\_with
- Remote (deprecated)
- Advanced\_Information
- Configuration
- Auto\_connect
- Pre\_connect
- Description

- Project (deprecated)
- Disabled
- Shared (deprecated)
- Family\_select
- BoardChip\_name.

### See also

- *Debug Configuration generic groups and settings* on page A-6.

## A.1.2 The DEVICE group

Use a DEVICE entry when you have to specify a lot of information for a specific device. The name of this group must be a name within the target configuration file, for example DEVICE=ARM940T.

---

### Note

---

A DEVICE group must not be used in a BCD file.

---

### Entries in the DEVICE group

The following entries are available:

- Connect\_with
- Remote (deprecated)
- Advanced\_Information
- Description
- Project (deprecated)
- Configuration
- Disabled
- Shared (deprecated)
- Family\_select
- BoardChip\_name.

---

### Note

---

You can use the DEVICE group instead of a CONNECTION group.

---

### See also

- *Debug Configuration generic groups and settings* on page A-6
- *The CONNECTION group* on page A-2.

## A.1.3 The BOARD, CHIP, and COMPONENT groups

Use the BOARD, CHIP, and COMPONENT groups to hold memory map-related settings when a standard board or chip exists:

- use a BOARD group to define a target processor core module or development board from a commercial hardware vendor, such as the CM940T or ARM® Integrator™/CP, or a custom design
- use a CHIP group to define significant devices on a target or where the target itself is complex

- use a COMPONENT group to define a target plus ASICs, either commercial or custom.

A CONNECTION group can refer to one or more of these groups by name or ID. A reference to one of these groups enables automatic use of the appropriate Debug Interface configuration file (such as .rvc), settings, and additional information for your development platform (ASIC, peripherals, and memory).

---

**Note**

---

A BOARD, CHIP, or COMPONENT group must only be used in a BCD file.

---

### Entries in the BOARD, CHIP, and COMPONENT groups

When you create a new BOARD, CHIP, or COMPONENT group, the following entries are available:

- Connect\_with
- Advanced\_Information
- Configuration
- Description
- Project (deprecated)
- Family\_select
- BoardChip\_name.

### See also

- *Debug Configuration generic groups and settings* on page A-6
- *The CONNECTION group* on page A-2.

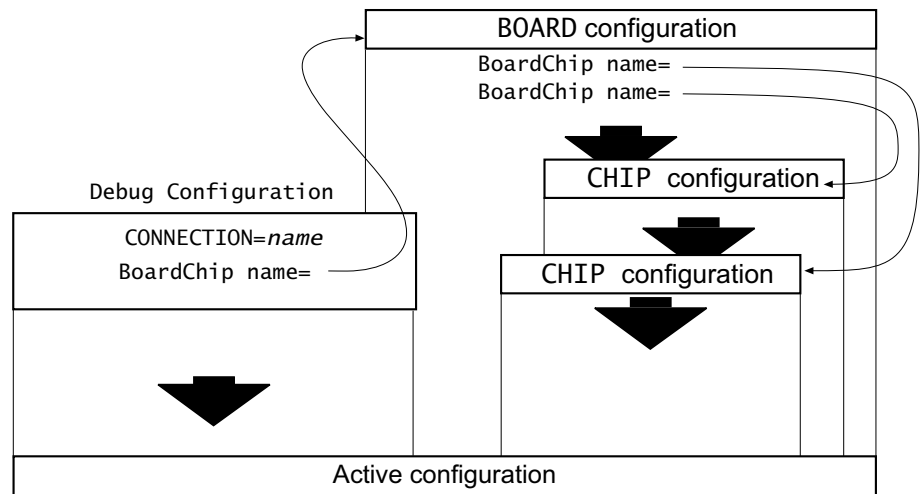
## A.1.4 Relationship between connections, boards, and chips

A CONNECTION group specifies the debugging environment for a Debug Configuration.

This includes the nature and addresses of the hardware interface, for example the port name of the JTAG interface that is connected to the development platform. This information is described in the Connect\_with block of the Debug Configuration and in the file associated with the Configuration setting.

A CONNECTION group can have one or more BoardChip\_name entries that are used to associate the Debug Configuration with one or more BOARD, CHIP, or COMPONENT descriptions that define memory maps, registers, and peripherals.

It is recommended that the descriptions of your development platform are only defined in BOARD, CHIP, or COMPONENT definitions, and that these descriptions are stored in *Board/Chip Definition* (BCD) files. For example, the definition of the memory map, registers and peripherals of the ARM Integrator/AP motherboard is stored in the file AP.bcd. Figure A-1 on page A-5 shows how these groups are linked.



**Figure A-1** How connections, boards, and chips fit together

The board file consists of Debug Configuration entries using the following groups:

- CONNECTION
- DEVICE.

These groups must not be used in a BCD file.

BCD files consist of memory map related configuration entries using the following groups:

- BOARD
- CHIP
- COMPONENT.

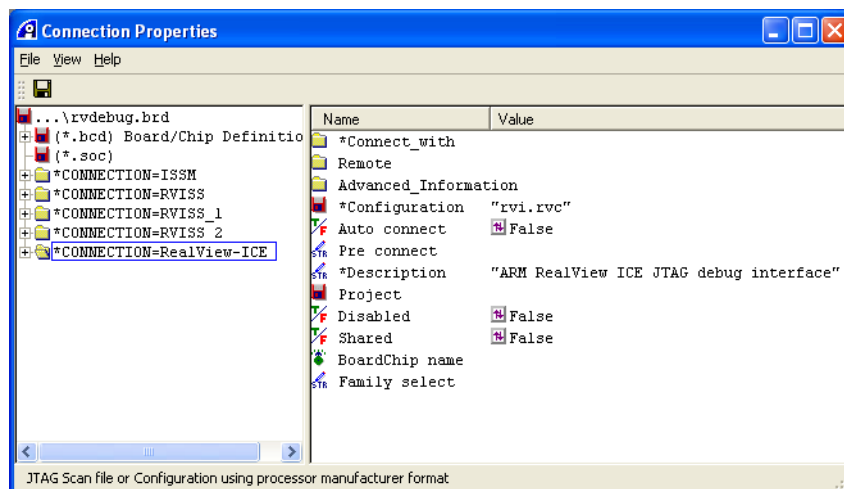
This appendix describes the groups and individual settings that appear in the board and BCD files. It assumes that you are familiar with the contents of the Connection Properties window.

### See also

- *Displaying the Connection Properties window* on page 3-6
- *The CONNECTION group* on page A-2
- *The DEVICE group* on page A-3
- *The BOARD, CHIP, and COMPONENT groups* on page A-3.

## A.2 Debug Configuration generic groups and settings

There are several board file entries that are common to many of the settings groups, shown in the example CONNECTION=RealView-ICE entry in Figure A-2.



**Figure A-2 Viewing generic settings for a Debug Configuration**

These settings are found in CONNECTION groups. In some cases, some or all of these settings are also found in other groups, such as the BOARD=AP group in the AP.bcd entry.

The following sections describe the generic groups and settings:

- *Connect\_with*
- *Remote* on page A-7
- *Advanced\_Information block (base group)* on page A-7
- *Configuration* on page A-7
- *Auto\_connect* on page A-8
- *Pre\_connect* on page A-8
- *Description* on page A-8
- *Project* on page A-8
- *Disabled* on page A-8
- *Shared* on page A-9
- *BoardChip\_name* on page A-9
- *Family\_select* on page A-9.

### A.2.1 Connect\_with

Identifies the Debug Interface making the connection. It includes the settings values:

- Manufacturer** The name and the type of the Debug Interface.
- Right-click to see a list of available Debug Interface types. You might also require an appropriate license and the hardware to use them. Valid options are:
- ARM-ARM-CML (Model Library)
  - ARM-ARM-CMP (Model process)
  - ARM-ARM-DS (DSTREAM)
  - ARM-ARM-MXD (Instruction Set System Model)
  - ARM-ARM-MXS (SoC Designer)



- ARM-ARM-NW (RealView ICE)
- ARM-ARM-RTS (Real-Time System Model)
- ARM-ARM-SW (RealView Instruction Set Simulator).

<b>IODEVICE</b>	Additional information about the target hardware. Not used in the current release of RealView Debugger.
<b>SPEED</b>	The emulation speed for some emulators. Not used in the current release of RealView Debugger.

---

**Note**

---

You must not change the Connect\_with group settings for your Debug Configuration.

---

## A.2.2 Remote

Deprecated.

## A.2.3 Advanced\_Information block (base group)

Provides additional configuration information about your development platform. By default, all settings in the Advanced\_Information block are provided under a base group called Default. Settings in this group are used for all the targets listed under the related Debug Configuration. However, if more than one target is present, each target might require different connection attributes. Therefore, you can create multiple, target-specific, Advanced\_Information base groups.

The settings you use in this group depend on the features you are configuring.

### See also

- *The Debug Configuration Advanced\_Information block* on page 3-8
- *Debug Configuration Advanced\_Information settings reference* on page A-10
- *Memory mapping Advanced\_Information settings reference* on page A-20.

## A.2.4 Configuration

Specifies a named Debug Interface configuration file. When you create a Debug Configuration in the Connect to Target window, RealView Debugger creates the configuration file for you. The file name is the same as the default Debug Configuration name assigned by RealView Debugger.

If the file specified here is a full path name, then only that location is used. Otherwise, the configuration file is searched using the RealView Debugger search path.

---

**Note**

---

If you rename the associated Debug Configuration, the name of the file specified here does not change.

---

### See also

- *The RealView Debugger search path* on page 1-17
- *The Debug Configuration Advanced\_Information block* on page 3-8.

### A.2.5 Auto\_connect

If set to True, RealView Debugger attempts to open the related Debug Configuration for the **Configuration** grouping in the Connect to Target window (the Connect to Target window does not have to be displayed for RealView Debugger to do this):

- If a valid Debug Interface configuration file is specified in the Configuration setting for the Debug Configuration, the list of targets on the associated development platform is displayed. The list of targets comes from the configuration file specified by the Configuration setting.
- If no Debug Interface configuration file is specified in the Configuration setting, or it is invalid, then a Target Connection Error dialog box is displayed. This dialog box enables you to choose how to configure the Debug Interface. In this case, select **Configure Device Information...**

#### See also

- the following in the *RealView Debugger User Guide*:  
— *Troubleshooting target connection problems* on page 3-60.

### A.2.6 Pre\_connect

Forces an order for target connection on a multiprocessor development platform, regardless of the target you use to initiate a connection. This enables pre-setup of specific targets to guarantee correct operation, such as initializations. If a listed target is already connected, then no attempt is made to re-connect to that target.

For each target you want to pre-connect:

1. Specify the full connection name for the target (that is, the name as it appears in the Connect to Target window.
  2. Press Enter to complete the entry.
- A new Pre\_Connect entry is created for the target.

For example, \*Pre\_Connect ARM940T\_0.

The targets are listed in reverse order. Therefore, the last target you enter is connected first. You can re-order this list if required.

#### See also

- *Changing the order of settings that have multiple values* on page 3-12
- *Setting a generic connection sequence for a Debug Configuration* on page 3-36
- *Pre\_connect* on page A-16.

### A.2.7 Description

A description for this Debug Configuration that describes how it is to be used.

### A.2.8 Project

Deprecated.

### A.2.9 Disabled

Disables this Debug Configuration entry so that it is hidden from the Connect to Target window.

## A.2.10 Shared

Deprecated.

## A.2.11 BoardChip\_name

Refers to the name of a BOARD, CHIP, or COMPONENT group listed in the (\*.bcd) Board/Chip Definitions group. If the group has more than one name separated by a slash (/), such as ID/name, any of them can be used. If not specified, the name of this group is used to match a board or chip.

### See also

- *Memory mapping Advanced\_Information settings reference* on page A-20.

## A.2.12 Family\_select

Ensures the correct family member is used (for example, for memory mapping, and registers) when the silicon ID is ambiguous. Some chip families do not use different silicon IDs for different members of the family, and this field enables you to specify which you are using. Specify the family member using one of these formats:

**name=family\_name** This enables you to specify the name of the device. Use either the name defined in the target configuration file or the processor name. This is used when multiple chips are housed on the same target but from different families.

**family\_name** Choose from the preconfigured list.

**silicon\_id** Can be expressed using one of the following formats:

- *num.num.num.num*, for example 15.255.15.15
- a value, for example 41029401.

## A.3 Debug Configuration Advanced\_Information settings reference

The Advanced\_Information block in a CONNECTION group enables you to provide additional connection information for your Debug Configuration, such as the connection mode and any OS-aware plug-in to use.

This section describes the settings and groups in the Advanced\_Information block of a CONNECTION group.

---

### Note

---

Although all settings are available under an Advanced\_Information block base group, you must use the settings described in this section only in a Debug Configuration. Therefore, make sure that any settings described in this section are not also set in any linked BCD files. This avoids conflicts with settings in any BCD files that you assign to the connection.

---

See also:

- *Application\_Load*
- *ARM\_config settings for a Debug Configuration* on page A-11
- *Logic\_Analyzer* on page A-13
- *Cross\_trigger* on page A-14
- *RTOS\_config* on page A-14
- *Monitor* on page A-15
- *Pre\_connect* on page A-16
- *Commands* on page A-16
- *Connect\_mode* on page A-17
- *Disconnect\_mode* on page A-17
- *Id\_chip* on page A-18
- *Id\_match* on page A-18
- *Chip\_name* on page A-18
- *Endianness* on page A-19
- *Sw\_bkpts* on page A-19
- *Advanced\_Information block (base group)* on page A-7
- *Memory mapping Advanced\_Information settings reference* on page A-20
- Chapter 3 *Customizing a Debug Configuration*
- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals*
- Chapter 5 *Debug Configuration Tutorial*.

### A.3.1 Application\_Load

Use the Application\_Load group to change the way that an executable image is loaded into target memory. The default is to write the memory using the emulator or EVM board. Settings in this group can be used to override the default and so disable all image load, for pure ROM or EPROM systems, or to run an external program to do the load.

#### Application\_Load group settings

The following settings are available in the Application\_Load group:

**Load\_using** Specifies how to perform the load.

**Load\_command**

This defines a shell command run to perform the load. The command might contain \$ variables, which are substituted by RealView Debugger before calling. The possible \$ variables are:

\$D        directory of the application  
 \$P        full path of the application  
 \$F        filename of the application  
 \$N        name of the application without the extension.

If the command starts with an exclamation mark (!), the return value of the shell command is not used to stop the load, otherwise a non-zero return aborts the load. In all cases, the output of the command is shown in the **Log** tab in the Output view.

**Load\_set\_pc** This controls how the PC is initialized during an image load. The default is to set the PC to the entry point if an entry point is defined and symbols are loaded and this is not an appended load (it is replace or new). This enables you to disable setting the PC under any situation, or to set it specifically to address 0.

**Include\_on\_load**

A script file that you want to execute after loading an image.

**A.3.2 ARM\_config settings for a Debug Configuration**

This group enables control of ARM processor settings used for ARM emulators, monitors, or simulators. These control features such as semihosting and vector catching, which must be set or unset depending on the type of runtime you have linked into your application.

**———— Note ————**

Although this group contains settings for memory control, only the semihosting and vector catch settings must be set in a Debug Configuration CONNECTION group.

You can also set many of these at runtime using pseudo-registers. To do this, use an Advanced\_Information block named Default if it applies to all devices or create a block with a name of the scan chain device to which it applies.

**ARM\_config group settings**

The following settings in the ARM\_config group must be set only in a Debug Configuration CONNECTION group:

**Vectors**        If Vector\_catch is set to True, the fields within this group enable individual control over each vector. They are used to catch possible program errors by setting breakpoints on (or otherwise trapping) the vectors. The default is to catch error-type vectors, but not IRQ, FIQ and SVC. SVC is caught separately by semihosting if enabled. The vectors must be writable.

The Vectors group contains:

**Reset**            Set this to catch Reset exceptions.  
**Undefined**       Set this to catch undefined instruction exceptions.  
**SVC**             Set this to catch *SuperVisor Call* (SVC) exceptions.  
**P\_Abort**         Set this to catch Prefetch abort exceptions.  
**D\_Abort**         Set this to catch Data abort exceptions.

<b>Address</b>	Set this to catch Address exceptions. Used only by the obsolete 26-bit ARM processor architectures.
<b>IRQ</b>	Set this to catch normal interrupt exceptions.
<b>FIQ</b>	Set this to catch Fast Interrupt exceptions.
<b>Error</b>	Set this to catch errors on <i>RealView ARMulator® ISS</i> (RVISS) targets only.

You can also set these during debugging as follows:

- Select **Processor Exceptions...** from the Code window **Debug** menu to display the Processor Exceptions dialog box.
  - Use the BGLOBAL CLI command.
  - Use the following pseudo-registers:
    - @vector\_catch for hardware connections
    - the @semlhost\_vector\_catch pseudo-register for connections through RVISS.
- Bits 0 to 8 of these pseudo-registers represent the vectors from Reset to Error, respectively.

---

**Note**

For non ARMv7-M processors with the semihosting vector set to the default (0x8), you must not enable the SVC vector catch if semihosting is enabled.

---

**Semihosting** Enables programs to communicate with the host workstation. Semihosting operations supported include stack and heap assignment and console I/O (printf and scanf type calls). Semihosting is implemented using the SVC instruction. You can change the semihosting vector during debug using the @semlhost\_vector pseudo-register.

On some targets you can also define a window or file number to display semihosting printf messages using setreg @SEMIHOST\_WINDOW=*number*. A window number must match a window opened with the VOPEN command, and a file number must match a file opened with the FOPEN command.

---

**Note**

If your program requires direct user input, do not change the value. The default value identifies the **StdIO** tab of the Output view. Direct user input is possible only from this tab.

---

The Semihosting group contains:

<b>Enabled</b>	Set this to enable semihosting.
<hr/> <p><b>Note</b></p> <p>For non ARMv7-M processors with the semihosting vector set to the default (0x8), you must not enable semihosting if the SVC vector catch is enabled.</p> <hr/>	
<b>Vector</b>	Address of semihosting vector. For non ARMv7-M processors, this is the address of the SVC vector catch.
<b>Arm_svc_num</b>	ARM SVC instruction for semihosting.
<b>Thumb_svc_num</b>	Thumb® SVC instruction for semihosting.

**Armulator** Contains:

<b>Clock_speed</b>	Clock speed in MHz as num.num.
--------------------	--------------------------------

<b>Fpoint_emu</b>	True if floating point emulation.
<b>Config_file</b>	The name of the configuration file.

**Vector\_catch**

When set to True, the fields within the Vectors group determine the which vectors are enabled.

**Properties** This enables free-form definition of the properties required by a Debug Interface (emulator or simulator). The form of the string is *name=value*, where *name* is the name for the property as defined by the Debug Interface and *value* is a numeric value in hex or decimal.

**See also**

- *Configuring vector catch* on page 3-24
- *ARM\_config settings related to memory mapping* on page A-20.

**A.3.3 Logic\_Analyzer**

This group is used to define settings for external trace analyzer hardware.

RealView Debugger currently supports:

- *ARM Embedded Trace Macrocell™ (ETM™)*
- *XScale™ onchip trace.*

By default, RealView Debugger is automatically configured with tracing enabled for all targets using preset values.

If you do not set the Vendor setting, you can connect to the analyzer using the following option from the Code window main menu:

**Tools → Analyzer/Trace Control → Connect Analyzer/Analysis...**

**Logic\_Analyzer group settings**

The following settings are available in the Logic\_Analyzer group:

<b>Vendor</b>	The vendor for the supported TPA.
<b>Machine</b>	Name of analyzer or device.
<b>Config</b>	Configuration file required to support your analyzer or device.
<b>Load_when</b>	Defines when RealView Debugger enables tracing: <ul style="list-style-type: none"> <li>• enabled on connection, with Load_when set to connect</li> <li>• enabled on image load, with Load_when set to image_load</li> <li>• enabled when a specified symbol is matched, with Load_when set to symbol_match</li> <li>• enabled when first trigger is specified, with Load_when set to first_use.</li> </ul>
<b>Sym_match</b>	Defines the symbol match to enable tracing.

**See also**

- *Naming an Advanced\_Information block group* on page 3-8
- *RealView Debugger Trace User Guide.*

### A.3.4 Cross\_trigger

These settings control the cross-triggering of a stop command between multiple processors that are closely coupled in hardware. They specify whether stopping execution of one processor stops execution of other processors, either because of a break or another stop condition:

- input triggering means that the processor is stopped by other processors
- output triggering means that the processor can stop other processors.

#### Cross\_trigger group settings

The following settings are available in the Cross\_trigger group:

<b>Trig_in_ena</b>	List of commands to enable input triggering.
<b>Trig_in_dis</b>	List of commands to disable input triggering.
<b>Trig_out_ena</b>	List of commands to enable output triggering.
<b>Trig_out_dis</b>	List of commands to disable output triggering.

#### See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 7 *Debugging Multiprocessor Applications* for details of hardware cross-triggering.

### A.3.5 RTOS\_config

This group enables you to configure an OS-aware connection.

#### RTOS\_config group settings

The following settings are available in the RTOS\_config group:

<b>Events</b>	This group indicates the events that are to be captured. Up to eight events can be captured. The event settings reflect the state of the events on the Event Filter dialog box.
<b>Vendor</b>	A three letter value that identifies the OS plug-in, that is the *.dll file supplied by your vendor.

<b>Load_when</b>	Defines when RealView Debugger loads the OS plug-in: <ul style="list-style-type: none"> <li>• load the plug-in on connection, with Load_when set to connect</li> <li>• wait until an OS image is loaded, with Load_when set to image_load.</li> </ul> <p>The OS features of the debugger are not enabled until the plug-in is loaded, the OS has been found on your target, and the OS is initialized.</p>
------------------	--

#### Base\_address

Defines a base address, overriding the default address used to locate the OS data structures. See your OS documentation for details.

<b>Cpu_id</b>	Specifies a CPU identifier so that you can associate the OS-aware connection to a specific CPU. See your OS documentation for details.
---------------	--



**Exit\_Options**

Defines how OS awareness is disabled. Use the context menu to specify the action to take when an image is unloaded or when you disconnect. You can also specify a prompt.

**RSD**

Controls whether RealView Debugger enables or disables RSD. This setting is only relevant if your debug target can support RSD.

**System\_Stop**

Use this setting to specify how RealView Debugger responds to a processor stop request when running in RSD mode.

In some cases, it is important that the processor does not stop. This setting enables you to specify this behavior, use:

- Never to disable all actions that might stop the processor.
- Prompt to request confirmation before stopping the processor.
- Don't prompt to stop the processor. This is the default.

**Event\_capture**

Controls whether RealView Debugger enables or disables the capturing of events. When enabled, RealView Debugger checks which event to capture by examining the settings in the Events group. See your OS documentation for details of the events that are supported.

**See also**

- the following in the *RealView Debugger RTOS Guide*:  
— Chapter 2 *Configuring OS-aware Connections*.

**A.3.6 Monitor**

This group enables you to configure a debug monitor, such as RealMonitor.

**Monitor group settings**

The following settings are available in the Monitor group:

<b>Type</b>	The type of debug monitor. The options available depend on the debug monitor you have installed.  For example, if you have installed RealMonitor, then select <b>RealMonitor</b> from the list.
<b>Option</b>	If your debug monitor requires configuration options, then specify them here.

**See also**

- *Customizing the Debug Configuration used for debugging with RealMonitor* on page 3-47.

### A.3.7 Pre\_connect

Forces an order for target connection. This depends on whether the name of the Advanced\_Information block is Default or has a target name:

- If the name is Default, then this setting operates in the same way as the generic Pre\_connect setting.
- If the name is a target name, for example ARM940T, then this setting identifies one or more targets that are to be connected before the ARM940T processor. This setting is used:
  - when you connect to the named target
  - the named target is connected as part of another Pre\_connect sequence.

If a listed target is already connected, then no attempt is made to re-connect to that target.

For each target you want to pre-connect:

1. Specify the full connection name for the target. That is, the name as it appears in the Connect to Target window, for example ETM\_1.
2. Press Enter to complete the entry.  
A new Pre\_Connect entry is created for the target.

The targets are listed in reverse order. Therefore, the last target you enter is connected first.

#### See also

- *Changing the order of settings that have multiple values* on page 3-12
- *Specifying a target-specific connection sequence* on page 3-38
- *Debug Configuration generic groups and settings* on page A-6
- *Advanced\_Information block (base group)* on page A-7.

### A.3.8 Commands

This enables you to specify RealView Debugger commands to run after a target connection is established. The most common example is to include commands from a file, using the INCLUDE command. The commands are run immediately after the connection is completed.

#### Considerations when using Commands settings

Be aware of the following:

- If the commands are defined in the Default group of the Advanced\_Information block, the commands are executed when you connect to each target in a multiprocessor Debug Configuration.
- If the commands are defined in a target-specific group of the Advanced\_Information block, the commands are executed only when you connect to a target that matches the group name.

#### See also

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

### A.3.9 Connect\_mode

When you connect to a target, RealView Debugger attempts to establish the connection using the default connect mode, that is No Reset and Stop.

Before connecting, RealView Debugger checks to see if a user-defined connect mode has been specified by the Connect\_mode setting in your board file. If such a setting is found, it becomes the default connect mode for this connection.

#### ———— Note ————

Make sure that the Connect\_mode setting is not set in any linked BCD file.

Use this setting to specify a connection mode. The options are:

**no\_reset\_and\_stop** Do not submit a reset and halt any process currently running.

**no\_reset\_and\_no\_stop**

Do not submit a reset or halt any process currently running.

**reset\_and\_stop** Do a processor reset and halt any process currently running.

**reset\_and\_no\_stop** Do a processor reset but do not halt any process currently running.

**prompt** Display a prompt for the connection mode to use.

### Considerations when using Connect\_mode

Be aware of the following:

- The options available for the Connect\_mode setting are generic to all Debug Interfaces and supported processors and so might include options that are not supported by your Debug Interface.
- If you set connect mode from the Connect to Target window, using the **Connect Mode** menu, this might temporarily override any user-defined setting(s) in your target configuration file.
- If a prompt is specified in your board file, it takes priority over any other user-defined connect mode setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.

### See also

- *Considerations for DSTREAM or RealView ICE* on page 3-20
- the following in the *RealView Debugger User Guide*:
  - *Connecting to a target using different modes* on page 3-43.

### A.3.10 Disconnect\_mode

When you disconnect from a target, RealView Debugger attempts to disconnect using the default disconnect mode, that is As-is without Debug.

Before disconnecting, RealView Debugger checks to see if a user-defined disconnect mode has been specified by the Disconnect\_mode setting in your board file. If such a setting is found, it becomes the default disconnect mode for this connection.

---

**Note**

---

Make sure that the `Disconnect_mode` setting is not set in any linked BCD file.

---

Use this setting to specify a disconnection mode. The options are:

- as\_is\_with\_debug** Leave the target in its current state, whether stopped or running, and maintain any debugging controls.
- as\_is\_without\_debug** Leave the target in its current state, whether stopped or running, and remove any debugging controls. This is the default.
- prompt** Display a prompt for the disconnection mode to use.

### Considerations when using `Disconnect_mode`

Be aware of the following:

- All vector catch breakpoints are removed from the target on disconnection, irrespective of the disconnect mode.
- The options available for the `Disconnect_mode` setting are generic to all Debug Interface interfaces and supported processors and so might include options that are not supported by your Debug Interface.
- If you set disconnect mode from the Connect to Target window, using **Disconnect (Defining Mode)...** on the **Target** menu, this temporarily overrides any user-defined setting(s) in your target configuration file.
- If a prompt is specified in your board file, it takes priority over any other user-defined disconnect mode setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.

### See also

- the following in the *RealView Debugger User Guide*:  
     — *Disconnecting from a target using different modes* on page 3-54.

#### A.3.11 `Id_chip`

By default, the chip-id, or silicon-id, is loaded from the processor. When accessing special custom chips, you might have to force the ID explicitly. The ID can be expressed as a 16-bit number or in *num.num.num* format. It can also be expressed as a name of the family member if known.

#### A.3.12 `Id_match`

This contains the expected silicon ID from the processor. If it does not match this value, you are prompted to choose whether or not to continue the connect operation. The ID can be expressed as a 16-bit number or in *num.num.num* format.

#### A.3.13 `Chip_name`

This defines the manufacturer name of the actual device, such as family name or processor name. The `Chip_name` field enables you to specify a name to use in messages and lists displayed by RealView Debugger. It does not enforce the chip family selection. For that, you must use the `Id_chip` field.

### A.3.14 Endianness

This applies to ARM architecture-based targets. Use it to set the byte order of the simulated processor. The options are:

- big** Forces the endianness to big-endian when connecting.
- little** Forces the endianness to little-endian when connecting.
- hardware** RealView Debugger automatically attempts to determine the endianness of the target when connecting.

Be aware of the following:

- If set to **hardware**, RealView Debugger sets the initial endianness to little.
- If set to **big**, RealView Debugger sets the initial endianness to big.
- If the target has a CP15\_CONTROL register, RealView Debugger compares the endianness of the target to the value used for the initial endianness. If the values do not match, then RealView Debugger issues a warning. Valid combinations are:
  - target processor set to little-endian or BE8, and endianness connection property set to **little** or **hardware**
  - target processor set to BE32, and endianness connection property set to **big**.
- If the target processor does not have a CP15\_CONTROL register (for example, some ARM7TDMI® processors), RealView Debugger cannot determine the endianness of the connected processor. Therefore, because RealView Debugger cannot make a comparison and no warning is produced.

### A.3.15 Sw\_bkpts

This specifies whether or not software breakpoints can be set on a connected target.

#### See also

- *Memory\_block group settings* on page A-21 for details of No\_sw\_bkpts.

## A.4 Memory mapping Advanced\_Information settings reference

The Advanced\_Information block enables you to provide additional memory map related information about your development platform and its components. That is, the memory map, memory mapped registers, and peripherals.

---

### Note

---

If you are not familiar with using the memory map related settings, it is suggested that you follow the tutorial described in Chapter 5 *Debug Configuration Tutorial*.

---

These settings can be nested so that one setting might refer to another, which in turn might refer to another setting. These references cause the information to be concatenated. References are made to other board and chip definitions, as required.

Although all settings are available under an Advanced\_Information block base group, you must use the settings described in this section only in a BCD file. Other setting must be used only in a Debug Configuration.

This section describes the memory map related settings and groups in the Advanced\_Information block base group.

---

### Note

---

Although all settings are available under an Advanced\_Information block base group, you must use the settings described in this section only in a BCD file. Therefore, make sure that any settings described in this section are not also set in the Debug Configuration. This avoids conflicts with any Debug Configuration settings when you assign a BCD file to that configuration.

---

See also:

- *ARM\_config settings related to memory mapping*
- *Memory\_block* on page A-21
- *Map\_rule* on page A-25
- *Register\_enum* on page A-26
- *Register* on page A-27
- *Concat\_Register* on page A-28
- *Peripherals* on page A-29
- *Register\_Window* on page A-30
- *Advanced\_Information block (base group)* on page A-7
- *Debug Configuration Advanced\_Information settings reference* on page A-10
- Chapter 3 *Customizing a Debug Configuration*
- Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals*
- Chapter 5 *Debug Configuration Tutorial*.

### A.4.1 ARM\_config settings related to memory mapping

This group enables control of ARM processor settings used for ARM emulators, monitors, or simulators. Although this group contains other settings for controlling features such as semihosting and vector catching, only the memory control settings for top of memory, and stack and heap assignment must be set in a BCD file.

You can also set many of these at runtime using pseudo-registers. To do this, use an Advanced\_Information block named Default if it applies to all devices or create a block with a name of the scan chain device to which it applies.

## ARM\_config group settings

The following settings in the ARM\_config group must be set only in a BCD file:

**Stack\_Heap** The ARM tools automatically set the stack and heap based on the top\_of\_memory using semihosting.

---

**Note**

RealView Debugger does not use the settings in this group.

---

### Top\_memory

Enables the semihosting mechanism to return the top of stack and base of heap. If not defined here, the default for each target is used. Any defined value is set into each tool to force this address base. You can use explicit stack and heap sizes and locations, but this might not be supported by all debug targets. You can also set this during a debugging session using the @top\_of\_memory pseudo-register.

---

**Note**

You cannot set top of memory for RVISS, ISSM, RTSM, and SoC Designer targets. There is no @top\_of\_memory symbol for these targets.

---

### See also

- *ARM\_config settings for a Debug Configuration* on page A-11
- *Naming an Advanced\_Information block group* on page 4-4.

## A.4.2 Memory\_block

Entries in the Memory\_block group are used to build up:

- Fixed memory blocks that refer to memory that is always enabled, always at the same place, and always the same size.
- Enabled memory blocks that are enabled or disabled by a register value.
- Based memory blocks that have the start or length attributes adjusted or set by a register value. This value might be added to an offset or might itself be the required value.

---

**Note**

If you configure any entries in this group and in the Map\_rule group, then memory mapping is enabled when you connect to the configured target.

---

## Memory\_block group settings

The Memory\_block group includes a base group called Default, but typically you create one or more named memory block groups. The group includes the following settings:

**Attributes** Additional attributes can be specified for the memory. These are used by simulator models to guide the debugger when accessing this block of memory. The following settings are available:

**Internal** This memory block is internal to the processor and not treated as external. This affects wait-state timings and other factors.

**Access-rule**

The access rule information is only used by simulators to control timing issues. It is noted if a link command file is generated.

**Access\_size**

Use this field to control how RealView Debugger accesses memory internally.

By default, this is set to 0, to indicate the default memory access size for the processor, for example, byte-size for ARM architecture-based targets.

Specify a value greater than 0 to set access size in bytes, for example 1 or 2. If you do this, you must also specify a size for `Memory_block`, otherwise RealView Debugger displays an error message.

For external memory with only byte-wide or halfword-wide access enabled, this can be used to ensure proper access to the memory.

Depending on the processor, this might have no effect.

**Volatile** Set to True if access either destroys the contents or the contents change in response to external events.

**No\_sw\_bkpts**

Use this when setting up a shared memory block to prevent software breakpoints being set in a shared memory region on a multiprocessor development platform.

Also, see *Sw\_bkpts* on page A-19.

**Shared** This field indicates if the memory block is shared with other targets:

- set to none if the memory block is not shared
- set to direct if the memory block is accessed directly using the bus.

If this is set, this field determines what other targets see this memory.

**Shared\_id**

This field contains a number that identifies this memory block. You must use the same number for each device that uses this memory block. RealView Debugger can then correctly update all connection views when this memory is modified.

You can use any integer value in the range 0 to 65535.

**Register\_Pos\_Len**

Used when one or two memory-mapped registers are used to set the base address and length of the memory block, such as for cross-bar switches, and chip-selects. These are not used for enables which are set using map rules. The following settings are available:

**Register\_base**

Enables you to specify a memory block position based on a memory-mapped register. The value of the register is added to the start field to construct the block start address. It can be masked and scaled (multiplied or divided) first.

**Base\_mask**

Mask to apply to register.



**Base\_scale**

Use this to change the value of the register contents, after masking, to define the actual base. If the number is positive, it is multiplied against the register content. If the number is negative, it is divided from the register content.

For example, a byte register might select the 64Kbyte region to map to. If the scale is 0x10000 (64K), then register values of 0, 1, 2, 3, ... each select a 64Kbyte region. If the selector occupies part of a register, the mask is applied to select only the selector portion and the scaling value itself might be scaled. Using this example, if the byte selector portion is the upper byte of a register, the scale value is 0x10000/0x100=0x100 (256). So, mask with 0xFF00 and multiply by 256 to get a 64Kbyte selection.

**Register\_length**

Enables a block of memory to be sized by a register. This is commonly used in multiprocessor shared memory systems. The content of the register is added to the specified length to compute the block length.

**Len\_mask**

Mask to apply to register.

**Len\_scale**

Used like Base\_scale.

**Len\_table**

Enables table indexing for the length. The length register is masked and scaled and then used as an index in a table of values. The last value is used if the scaled register value is too large. The table value is added to the length field of the block.

**Update\_rule**

Indicates how often to check the register to see if the mapping has changed. For cases where the mapping is set by jumpers, which read as registers, it must be inspected only when first connecting to the device. If the program changes it, it must be tested on each stop. Valid values are:

init_time	Test when connecting to device.
update_init	Test on connect and when register changes.
stop_update	Test on connect, change, and execution stop.

**Start** The base address of the block. If the block is mapped using a register, this is the offset from that register.

**Note**

You cannot include the TrustZone® world prefix in the address. To specify the TrustZone world for this memory region, use the Tz\_world setting.

**Length** The length in memory units of the block. If Length is set by a register, this must be 0 or the amount to add to that register.

**Type** The type of memory depends on the device type. The default is to map to data space. Otherwise, a memory space can be specified. Set to default for ARM architecture-based targets.

- Access** Indicates how the memory is to be treated. For simulators, this affects the target use of the memory. For hardware targets, this only affects how the debugger uses the memory and any generated linker command files. Right-click on this setting to see available access types.
- Tz\_world** Specifies the default world for this memory block on a processor that supports TrustZone technology:
- Normal** Specifies that the memory block is in the Normal World.
  - Secure** Specifies that the memory block is in the Secure World. This is the default.
- Wait\_states** Used with simulators to calculate the cycles used when accessing this memory. The default is based on the wait-state model used by the processor for external memory. This value is noted when link command files are generated from this data to enable careful positioning of sections to this memory.
- Flash\_type** Contains the name of a file containing the Flash programming code and information for this processor. Example files for selected ARM targets are provided in the *install\_directory\RVD\Flash\...\platform* directory. Files are collected in subdirectories based on the target Flash device, for example *install\_directory\RVD\Flash\...\IntegratorAP*. These files have the file extension *.fme*.
- By using the routines in this file, RealView Debugger can erase, modify and verify the contents of Flash memory.
- You can create your own FME file if required.

**Description** Description of the memory block.

#### Flash\_write\_mode

Indicates whether or not a prompt is displayed when a write operation is performed to a Flash memory block:

- Auto** Specifies that a Flash write operation is to occur automatically. Also, specify the clock speed with *Flash\_write\_clock*.
- When you load a binary file, RealView Debugger issues a *READFILE* command. The operation is complete when the Flash Memory Control dialog box is displayed, and messages similar to the following appear in the **Cmd** tab of the Output view:
- ```
Flash opened on ARM940T_0@RealView-ICE for 'Intel DT28F320S3
2Mx16 x2 x4' at '0x24000000'
Flash opened on ARM940T_0@RealView-ICE for 'Intel DT28F320S3
2Mx16 x2 x4' at '0x24000000'
Written memory from 0x24000000 to 0x2401922B.
```
- When you write a value directly in a Flash location with the Memory view, then RealView Debugger issues a *CEXPRESSION* command. The operation is complete when the Flash Memory Control dialog box is displayed, and messages similar to the following appear in the **Cmd** tab of the Output view:
- ```
Flash opened on ARM940T_0@RealView-ICE for 'Intel DT28F320S3
2Mx16 x2 x4' at '0x24000000'
Result is: 97 0x61 'a'
```
- When you write to Flash locations with the Fill Memory dialog box, then RealView Debugger issues a *FILL* command. The operation is complete when the Flash Memory Control dialog box is displayed, and a message similar to the following appears in the **Cmd** tab of the Output view:

Flash opened on ARM940T\_0@RealView-ICE for 'Intel DT28F320S3 2Mx16 x2 x4' at '0x24000000'

**Prompt** Specifies that you are prompted before the Flash write operation occurs. This is the default.

In this mode, RealView Debugger displays the Flash Memory Control dialog box, where you can perform the Flash write operation manually.

### Flash\_write\_clock

When Flash\_write\_mode is set to Auto, specify the clock speed in Hz.

**Volatile** Enables you to define a memory block that is volatile on read (and so is marked specially in the Memory view). The format is an offset from within this block (0 relative). A range can be specified, for example 0x10..0x20 or 0x40..+4. If not a range, it defines a single value.

### See also

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Creating a memory map block* on page 4-29
- *Setting up controlled memory blocks* on page 4-49
- *Sw\_bkpts* on page A-19
- *Map\_rule*
- Chapter 6 *Programming Flash with RealView Debugger*
- the following in the *RealView Debugger User Guide*:
  - Chapter 6 *Writing Binaries to Flash*
  - Chapter 9 *Mapping Target Memory*.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for a description of the CEXPRESSION, FILL, and READFILE commands.

## A.4.3 Map\_rule

This group controls the enabling and disabling of memory blocks using target registers. You specify a register to be monitored, and when the contents match a given value, a set of memory blocks is enabled. You can define several map rules, one for each of several memory blocks.

### ————— Note —————

If you configure entries in this group and in the Memory\_block group, then memory mapping is enabled when you connect to the configured target.

### Map\_rule group settings

The Map\_rule group includes a base group called Default, but typically you create one or more named map rule groups. The group includes the following settings:

**Register** This is the name of a memory-mapped target register that controls the visibility of a memory block. This register is read to determine the current mappings. You must define the register using the Register block.

You are recommended to name the register itself instead of the bit fields within it when more than one bit field controls the mapping.

- Mask** This is ANDed with the contents of the register as described in **Value**.
- Value** This is compared with the register contents after the mask is added. The comparison is  $(\text{reg-value} \& \text{mask}) == \text{value}$ . For example, if bit 3 being HIGH means the map is enabled, mask is 0x8 (1<<3) and value is also 0x8.
- On\_equal** This contains the name of one or more memory blocks to enable when the value matches, or disable when it does not match. To replace one block with another, create one rule that tests for one value and another that tests for a different value.

#### Update\_rule

Indicates how often to check the register to see if the mapping has changed. For cases where the mapping is set by jumpers, which read as registers, it must be inspected only when first connecting to the device. If the program changes it, it must be tested on each stop. Valid values are:

init_time	Test when connecting to device.
update_init	Test on connect and when register changes.
stop_update	Test on connect, change, and execution stop.

#### See also

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Creating a memory map block* on page 4-29
- *Setting up controlled memory blocks* on page 4-49
- *Memory\_block* on page A-21
- *Register* on page A-27
- the following in the *RealView Debugger User Guide*:
  - Chapter 9 *Mapping Target Memory*.

### A.4.4 Register\_enum

Enumerations can be used, instead of values, when a register is displayed in the Registers view. This setting enables you to define the names associated with different values. Names defined in this group are displayed in the Registers view, and can be used to change register values.

Register bit fields are numbered 0, 1, 2,... regardless of their position in the register.

#### Register\_enum group settings

The Register\_enum group includes a base group called Default, but typically you create one or more named register enumeration groups. The group includes the following setting:

- Names** You can specify a list of names in one of the following formats:
- *name, name, name, ...*
  - *name=value, name=value, name=value, ...*

#### See also

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Creating a custom memory mapped register* on page 4-37
- *Register* on page A-27
- *Peripherals* on page A-29

- the following in the *RealView Debugger User Guide*:  
— Chapter 9 *Mapping Target Memory*.

## A.4.5 Register

This group enables you to define memory-mapped registers provided at the board or ASIC level. Each register is named and typed and can be subdivided into bit fields (any number of bits) which act as subregisters.

### Register group settings

The Register group includes a base group called `Default`, but typically you create one or more named register groups. The group includes the following settings:

<b>Bit_fields</b>	This group contains settings to define bit fields within the registers.
<b>Start</b>	The base address of the register. If the register is mapped using a memory block, this is the offset from that register (see <code>Base</code> ).
<b>Length</b>	The length of the register. If <code>Length</code> is set by a register, this must be 0 or the amount to add to that register.
<b>Base</b>	This specifies how to interpret <code>Start</code> . You can select the value from a context menu. This context menu contains the option <code>Absolute</code> , and is also populated with the names of any memory blocks that you have configure in the <code>Memory_block</code> group: <ul style="list-style-type: none"> <li>• If you select <code>Absolute</code>, then <code>Start</code> is the absolute memory address of the register.</li> <li>• If you select the name of a memory block, then <code>Start</code> is an offset from the base address of that memory block.</li> </ul>

### Memory\_type

The type of memory depends on the device type. The default is to map to data space. Otherwise, a memory space can be specified. Set to `default` for ARM architecture-based targets.

<b>Type</b>	Specifies how to interpret the value contained in the register. The type names are as in the C language.
<b>Read_only</b>	If set to <code>True</code> , the register is read-only and the debugger does not let you write to it. Otherwise, you can modify the value using the Registers view in the Code window and using CLI expressions.
<b>Write_only</b>	If set to <code>True</code> , the register cannot be read. The debugger does not attempt to query the hardware for the current value when the Registers view is displayed.
<b>Volatile</b>	If set to <code>True</code> , the register value can change without the debugger explicitly modifying it. For example, a hardware timer continues to count even when the processor is halted.
<b>Enum</b>	The name of a <code>Register_enum</code> block that maps a register value to a textual string describing the value.
<b>Gui_name</b>	The name of the register as it appears in the Registers view. If no name is specified here, then the Register group name is displayed.

## Register Bit\_fields group settings

The Register Bit\_fields group contains a Default group, but typically you create one or more named Bit\_fields groups. The group contains the following settings:

<b>Position</b>	Bit position from 0 (LSbit).
<b>Size</b>	Size in bits.
<b>Signed</b>	Set to True if signed, otherwise set to False.
<b>Enum</b>	Enumeration name to show values in the Registers view, derived from the Register_enum group.
<b>Read_only</b>	Set to True if read-only (cannot modify).
<b>Volatile</b>	Indicates that the register has side effects when it is read or written. A common read side effect is loss of data (when pulled from a UART, for example). A common write side effect is for the device to take some action on write (triggering a DMA, for example). This information is used in the Registers view. Right-click to see available options.
<b>Gui_name</b>	Optional name for showing in Registers view. If no name is specified here, then the bit field group name is displayed.

### See also

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Creating a custom memory mapped register* on page 4-37
- *Setting up controlled memory blocks* on page 4-49
- *Memory\_block* on page A-21
- *Register\_enum* on page A-26
- *Register Bit\_fields group settings*
- the following in the *RealView Debugger User Guide*:
  - Chapter 9 *Mapping Target Memory*.

## A.4.6 Concat\_Register

You can define a concatenated register that is built up using specific bits from other registers. Concatenated registers are usually used only for memory mapping, but you can also use them for control and status. The recommended approach is to name two registers and then shift and mask them into the new register. If you want to concatenate parts from more than two registers, you can build them up in stages.

### Concat\_Register group settings

The Concat\_Register group includes a base group called Default, but typically you create one or more named concatenated register groups. The group includes the following settings:

<b>Low_name</b>	Name of the register to be used as the source for the low part of the concatenated register.  This can be the name of another concatenated register.
<b>Low_shift</b>	Amount to shift the source register specified by Low_name. Specify a negative value for left shift.
<b>Low_mask</b>	The mask to be applied to the source register specified by Low_name.

---

**Note**

---

The mask is applied after the shift operation.

---

**High\_name** Name of the register to be used as the source for the high part of the concatenated register.

This can be the name of another concatenated register.

**High\_shift** Amount to shift the source register specified by High\_name.

Specify a negative value for left shift.

**High\_mask** The mask to be applied to the source register specified by High\_name.

---

**Note**

---

The mask is applied after the shift operation.

---

**Length** Length of the concatenated register, in memory units. The default is 4.

**Type** An explicit type for the register. If you do not specify the type, the default is the signed scalar C type based on the register size.

**Enum** Enumeration name to show values in the Registers view, derived from Register\_enum group.

**Gui\_name** Optional name for showing in Registers view. If no name is specified here, then the concatenated register group name is displayed.

**See also**

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Creating a concatenated register* on page 4-54
- *Register\_enum* on page A-26
- the following in the *RealView Debugger User Guide*:
  - Chapter 9 *Mapping Target Memory*.

## A.4.7 Peripherals

This group enables you to define block peripherals so they can be mapped in memory, for display and control, and accessed for block data, when available. You define the peripheral in terms of the area of memory it occupies (for all its registers), and a breakdown of the registers used for access and control.

### Peripherals group settings

The Peripherals group includes a base group called Default, but typically you create one or more named peripherals groups. The group includes the following settings:

**Access\_Method**

This applies only when you can access blocks of data, and contains:

**Type** Method used to extract data.

**Method\_name**

Name of access method function if required.

**Start** Buffer or DMA start address.

**Length** Buffer or DMA length.

<b>Register</b>	Used to add memory mapped registers provided at the board or ASIC level. Each register is named and typed and can be subdivided into bit fields (any number of bits) which act as subregisters.
<b>Start</b>	Start address of first peripheral register.
<b>Length</b>	Register length.
<b>Base</b>	<p>This specifies how to interpret Start. You can select the value from a context menu. This context menu contains the option <i>Absolute</i>, and is also populated with the names of any memory blocks that you have previously configured in the <i>Memory_block</i> group:</p> <ul style="list-style-type: none"> <li>• If you select <i>Absolute</i>, then Start is the absolute memory address of the first peripheral register.</li> <li>• If you select the name of a memory block, then Start is an offset from the base address of that memory block.</li> </ul>

---

**Note**

---

If the memory block is disabled, then the related peripheral is too.

---

<b>Type</b>	<p>Basic type of the device. The available values are:</p> <ul style="list-style-type: none"> <li>• serial</li> <li>• parallel</li> <li>• block</li> <li>• network</li> <li>• display</li> <li>• other.</li> </ul>
-------------	--

**Description** Description of the peripheral.

**See also**

- *Suggested naming convention for memory map related settings groups* on page 4-5
- *Creating a custom peripheral* on page 4-40
- *Memory\_block* on page A-21
- *Register* on page A-27
- the following in the *RealView Debugger User Guide*:
  - Chapter 9 *Mapping Target Memory*.

## A.4.8 Register\_Window

This group defines the Registers view tab that is used to display your custom memory mapped registers and peripherals. The group contains a base group called *Default*, but typically you rename this group to a meaningful name for the tab in the Registers view where the registers and peripherals are to be displayed. You can create additional groups if you want to split your registers between multiple tabs in the Registers view.

If you specify the same name in multiple BCD files, and those BCD files are assigned to the same Debug Configuration, then:

- all the lines defined in each BCD file are displayed in a single tab
- the lines are displayed in the order that the BCD files are listed in the Debug Configuration.



---

**Note**

---

Be aware that if you specify the name of a tab that already exists for the target, then your memory mapped registers appear on that tab. For example, if you define Core as a name, then your memory mapped registers appear at the bottom of the **Core** tab.

---

**Register\_Window group settings**

The Register\_Window group contains the following setting:

**Line** Specifies a list of custom registers and peripherals that are to be displayed on the line. If you want the registers and peripherals to appear on more than one line, then you specify multiple Line settings as required.

The format of a line is *name,name,name,...* where each *name* is the name of a register or bit field. You can also apply additional formatting with the following characters:

- If the string starts with an equals sign, =, all the registers are shown as name value in the window, otherwise they are shown in table form (that is, the name is above the value).
- If a line starts with an underscore character, \_, the line shows as a comment label (non-active).
- If the line starts with an exclamation mark (!) it provides a description line for the tab.
- If the line starts with:
 

\$	the next line starts or ends an expansion block, controlled by + or -.
\$+	indicates a collapsed block
\$-	indicates an expanded block
\$\$	this ends a previously opened block.

**See also**

- *Creating the register tab for displaying custom registers and peripherals* on page 4-44
- *Register* on page A-27
- *Concat\_Register* on page A-28
- *Peripherals* on page A-29
- the following in the *RealView Debugger User Guide*:
  - Chapter 9 *Mapping Target Memory*.

## Appendix B

# ISSM Configuration Reference

This appendix describes the configuration parameters and memory map details for the *Instruction Set System Model* (ISSM) simulated targets supplied with ARM® RealView® Development Suite. It includes:

- *Cortex-A8 model configuration* on page B-2
- *Cortex-M0 model configuration* on page B-10
- *Cortex-M1 model configuration* on page B-17
- *Cortex-M3 model configuration* on page B-25
- *Cortex-R4 model configuration* on page B-33.

See also:

- *Customizing an ISSM Debug Interface configuration* on page 2-15.

## B.1 Cortex-A8 model configuration

The following sections describe the configuration details of the Cortex™-A8 model:

- *Summary of supported features for the Cortex-A8 model*
- *Limitations of the Cortex-A8 model*
- *Cortex-A8 configuration parameters* on page B-3
- *Cortex-A8 memory map* on page B-4
- *Cortex-A8 peripheral register mapping* on page B-5.

### B.1.1 Summary of supported features for the Cortex-A8 model

The following features are supported by the Cortex-A8 model:

- The Cortex-A8 model can execute all the instructions supported by Cortex-A8 processor, including ARM, Thumb®-2, Thumb-2EE, NEON™ and VFPv3 instructions.

———— **Note** ————

Disable the NEON and VFPv3 instruction sets to model the Cortex-A8-mini.

- Execution performance is a minimum of 5 MIPS on a 2GHz Intel workstation.
- The behavior of the L1 and L2 caches are modeled.
- All CP15 controls are modeled, except for CP15 register 15 (internal TLB registers).

———— **Note** ————

Be aware that the cycle counting registers are not functional and do not provide any cycle information.

- The model includes the following peripherals:
  - *Primary Interrupt Controller* (PIC)
  - *Real-Time Clock* (RTC)
  - two UARTs,
  - three Timers
  - Interrupt Controller.
- On reset, the model behaves similar to hardware. For example, with caches and VFP disabled, appropriate initialization code is required to set these up. Suitable initialization code in assembler is provided in the following directory:

`install_directory\RVDS\Examples\...\..\..\cached_dhry\CortexA8dhry`

**See also**

- *Cortex-A8 Technical Reference Manual.*

### B.1.2 Limitations of the Cortex-A8 model

The debug and trace hardware is not modeled on the Cortex-A8 model.

### B.1.3 Cortex-A8 configuration parameters

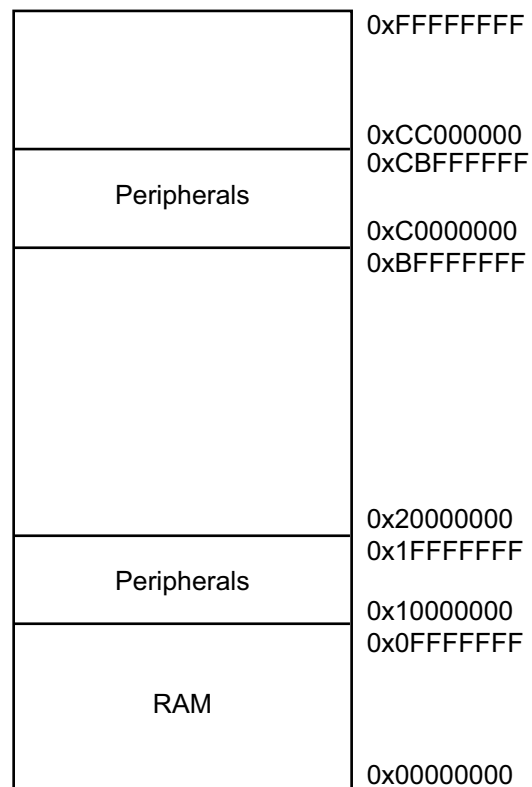
Table B-1 describes the configuration parameters for the Cortex-A8 model.

**Table B-1 Cortex-A8 model configuration parameters**

Parameter	Description	Default
cfgte	Set processor to Thumb-2 mode when entering an exception.	False
cfgend0	Set processor to BE-8 endianness when entering an exception.	False
cfgnmfi	FIQ is to be non-maskable.	False
clock-frequency	Processor clock frequency in Hz.	1000000000
cp15sdisable	Disable CP15 secure operation.	False
cpexist	Bitmask to enable some or all of CP0 to CP13.	3072
fast_invalidate	Encode value for Fast-Invalidate operations.	0
ISSCmpon	Enable ISS Compare.	False
l1_cachesize	Define L1 cache size in Kb (16 or 32).	32
l2_cachesize	Define L2 cache size in Kb (0, 64, 128, 256, ..., 2048).	256
NoNEON	Disable NEON and VFP.	False
semihosting-ARM_SVC	The ARM <i>SuperVisor Call</i> (SVC) used for semihosting. The value must be in the range 0 to 16777215 (0xFFFFFFFF).	0x123456
semihosting-cmd_line	Semihosting command line string used for passing command-line arguments to an image. Separate each argument with a space, and quote any arguments that include a space, for example: argument1 "argument 2" argument_3	
semihosting-debug	Print verbose messages for semihosting.	False
semihosting-enable	Enable or disable semihosting.	True
semihosting-heap_base	Semihosting heap base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0
semihosting-heap_limit	Semihosting heap limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x7000000
semihosting-stack_base	Semihosting stack base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x8000000
semihosting-stack_limit	Semihosting stack limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x7000000
semihosting-Thumb_SVC	The Thumb SVC used for semihosting. The value must be in the range 0 to 255 (0xFF).	0xAB
siliconid	Defined reset value of CP15 Silicon ID register.	0x41000000
vinithi	Exception vectors start at 0xFFFF0000.	False
warn-extra	Print additional warning messages.	False
warn-undefined	Print a warning when an undefined instruction is hit.	False

### B.1.4 Cortex-A8 memory map

Figure B-1 shows the default memory map for the Cortex-A8 model, which is a flat 4GB memory space.



**Figure B-1 Cortex-A8 memory map**

The undefined memory regions at 0x20000000 and 0xCB000000 do not cause an abort when accessed, but the return value is non-deterministic. If you set up the MMU appropriately, the Cortex-A8 can map the RAM (0x00000000 to 0x10000000) into other locations.

### B.1.5 Cortex-A8 peripheral register mapping

Table B-2 shows the peripheral register mapping for the Cortex-A8 model (see Figure B-1 on page B-4). The functional peripherals are PIC, RTC, three Timers, and two UARTs. These peripherals are described in the following sections. All other peripherals are represented by a basic register array in the memory map.

To access the peripheral registers, select the corresponding tab in the Registers view.

**Table B-2 Cortex-A8 peripheral register mapping**

Address	Peripheral
0x10000040 - 0x1000007F	CIC (Core Module Interrupt Controller)
0x13000000 - 0x13FFFFFF	Timers
0x14000000 - 0x14FFFFFF	PIC (Primary Interrupt Controller)
0x15000000 - 0x15FFFFFF	RTC (Real-Time Clock)
0x16000000 - 0x16FFFFFF	UART0
0x17000000 - 0x17FFFFFF	UART1
0x18000000 - 0x18FFFFFF	PS/2 Keyboard
0x19000000 - 0x19FFFFFF	PS/2 Mouse
0x1A000000 - 0x1AFFFFFF	LED/Alphanumeric/DIP Switches
0x1C000000 - 0x1CFFFFFF	MMC
0x1D000000 - 0x1DFFFFFF	Advanced Audio CODEC
0x1E000000 - 0x1EFFFFFF	TSCI (Smart Card Interface)
0xC0000000 - 0xC0FFFFFF	CP_LCD
0xC8000000 - 0xC8FFFFFF	Ethernet (SCMC 91C111)
0xC9000000 - 0xC9FFFFFF	CP_GPIO
0xCA000000 - 0xCAFFFFFF	SIC (Secondary Interrupt Controller)
0xCB000000 - 0xCBFFFFFF	CP_control

#### Primary Interrupt Controller

Table B-3 shows the PIC registers of the Cortex-A8 model.

**Table B-3 Cortex-A8 PIC registers**

Offset	Register
0x00	PIC_IRQ_STATUS
0x04	PIC_IRQ_RAWSTAT
0x08	PIC_IRQ_ENABLE
0x10	PIC_INT_SOFTSET
0x14	PIC_INT_SOFTCLR

**Table B-3 Cortex-A8 PIC registers (continued)**

Offset	Register
0x20	PIC_FIQ_STATUS
0x24	PIC_FIQ_RAWSTAT
0x28	PIC_FIQ_ENABLE

## Real-Time Clock (PL030)

Table B-4 shows the RTC registers of the Cortex-A8 model.

**Table B-4 Cortex-A8 RTC registers**

Offset	Register
0x000	RTC_DR
0x004	RTC_MR
0x008	RTC_STAT
0x00C	RTC_CLR
0x010	RTC_CR
0xFE0	RTC_PeriphID0
0xFE4	RTC_PeriphID1
0xFE8	RTC_PeriphID2
0xFEC	RTC_PeriphID3
0xFF0	RTC_PCellID0
0xFF4	RTC_PCellID1
0xFF8	RTC_PCellID2
0xFFC	RTC_PCellID3

## Timers

Table B-5 shows the Timer registers of the Cortex-A8 model.

**Table B-5 Cortex-A8 Timer registers**

Offset	Register
0x000	Timer0Load
0x018	Timer0BGLoad
0x004	Timer0Value
0x008	Timer0Control
0x00C	Timer0IntClr
0x010	Timer0RIS

**Table B-5 Cortex-A8 Timer registers (continued)**

Offset	Register
0x014	Timer0MIS
0x100	Timer1Load
0x118	Timer1BGLoad
0x104	Timer1Value
0x108	Timer1Control
0x10C	Timer1IntClr
0x110	Timer1RIS
0x114	Timer1MIS
0x200	Timer2Load
0x218	Timer2BGLoad
0x204	Timer2Value
0x208	Timer2Control
0x20C	Timer2IntClr
0x210	Timer2RIS
0x214	Timer2MIS
0xFE0	Timer_PeriphID0
0xFE4	Timer_PeriphID1
0xFE8	Timer_PeriphID2
0xFEC	Timer_PeriphID3
0xFF0	Timer_PCellID0
0xFF4	Timer_PCellID1
0xFF8	Timer_PCellID2
0xFFC	Timer_PCellID3

The three timers count down by only one tick for each instruction executed. The timers are mapped to the same memory locations as the timers on the Integrator™/CP development board.

———— **Note** —————

The Timer\_PCellID and Timer\_PeripID registers are not supported, but this does not affect the operation of the peripheral because it conforms to the ARM Generic Peripheral specification for Timers.



## UART

Table B-6 shows the UART0 and UART1 registers of the Cortex-A8 model.

**Table B-6 Cortex-A8 UART registers**

Offset	Register
0x000	UARTDR
0x004	UARTSR_ECR
0x018	UARTFR
0x020	UARTILPR
0x024	UARTIBRD
0x028	UARTFBRD
0x02C	UARTLCR_H
0x030	UARTCR
0x034	UARTIFLS
0x038	UARTIMSC
0x03C	UARTIS
0x040	UARTMIS
0x044	UARTICR
0x048	UARTDMACR
0xFE0	UARTPeriphID0
0xFE4	UARTPeriphID1
0xFE8	UARTPeriphID2
0xFEC	UARTPeriphID3
0xFF0	UARTPCe11ID0
0xFF4	UARTPCe11ID1
0xFF8	UARTPCe11ID2
0xFFC	UARTPCe11ID3

The Cortex-A8 models two serial ports that can be used for I/O. These are modeled through telnet windows, which open automatically on the first read/write operation from the UART after it has been initialized.

The serial ports are mapped to the same memory locations as the UARTs on the Integrator/CP development board.

### ————— **Note** —————

The default behavior for the telnet protocol is line mode. This means that characters typed into the telnet window are not seen in the model until you press Enter. To change this behavior type the telnet escape character, usually Ctrl+], and then enter the mode character. Doing this might disable local echo of the characters entered in the telnet window.

## Interrupts

Table B-7 shows the supported interrupts for the Cortex-A8 model.

**Table B-7 Cortex-A8 interrupts**

Interrupt	Peripheral
1	UART0
2	UART1
5	Timer0
6	Timer1
7	Timer2
8	RTC (Real-Time Clock)

## See also

- the following in the *RealView Debugger User Guide*:
  - *Viewing registers* on page 13-25.

## B.2 Cortex-M0 model configuration

The following sections describe the configuration details of the Cortex-M0 model:

- *Summary of supported features for the Cortex-M0 model*
- *Limitations of the Cortex-M0 model*
- *Cortex-M0 configuration parameters* on page B-11
- *Cortex-M0 memory map* on page B-12
- *Cortex-M0 peripheral register mapping* on page B-13.

### B.2.1 Summary of supported features for the Cortex-M0 model

The following features are supported by the Cortex-M0 model:

- The Cortex-M0 model can execute all the Thumb-1 (THUMBv3) instructions supported by Cortex-M0 processor.
- Execution performance is a minimum of 6 MIPS on a 2GHz Intel workstation.
- The model includes the following peripherals:
  - one UART
  - three Timers
  - *Nested Vectored Interrupt Controller (NVIC)*.
- On reset, the model behaves similar to hardware. Therefore, initial SP and PC values must be loaded at address 0x00000000, followed by system handler and interrupt vectors. An example is provided in the following directory to illustrate the use of the SP/PC addresses and Cortex-M0 libraries:

```
install_directory\RVDs\Examples\...\platform\Cortex-M0
```

#### See also

- *Cortex-M0 Technical Reference Manual*
- *Integrator CP User Guide* for details of Timers
- *PrimeCell UART (PL011) Technical Reference Manual* for details of UARTs.

### B.2.2 Limitations of the Cortex-M0 model

The Cortex-M0 model has the following limitations:

- The FPB and DWT debug components are not modeled.
- The debug and trace hardware is not modeled.
- The halt-mode debug is not modeled. Consequently, using BKPT results in the model stopping for the following conditions:
  - when `semihosting_enabled` is `False`
  - when `semihosting_enabled` is `True` and the BKPT immediate value does not match the `semihosting_BKPT` value.

In both cases, the following error message is displayed:

```
[core] Simulation terminated: Using BKPT for halt-debug is unsupported
```

- Lockup conditions are modeled as end-of-simulation conditions. This is because the model system does not include any peripherals to detect and correct these exception cases. See the section that describes unrecoverable exception cases in the *ARMv6-M Architecture Reference Manual*.

- Late-arrival preemption and tail chaining are not modeled. This is because exception entry and exit are modeled as atomic transitions. Therefore, it is impossible for an asynchronous exception to be delivered within exception entry. See the section that describes exceptions on exception entry in the *ARMv6-M Architecture Reference Manual*.

### B.2.3 Cortex-M0 configuration parameters

Table B-8 describes the configuration parameters for the Cortex-M0 model.

**Table B-8 Cortex-M0 model configuration parameters**

Parameter	Description	Default
BIGEND	Run in big-endian (BE8) mode.	False
CFGITCMEN	Reset values of AuxControl <i>Instruction Tightly-Coupled Memory</i> (ITCM) alias enable bits.	0x1
dbg_extension	Use debug extensions.	False
os_extension	Use OS extensions.	True
semihosting_BKPT	The breakpoint used for semihosting. The value must be in the range 0 to 255 (0xFF).	0xAB
semihosting_cmd_line	Semihosting command line string used for passing command-line arguments to an image. Separate each argument with a space, and quote any arguments that include a space, for example: argument1 "argument 2" argument_3	
semihosting_debug	Print verbose messages for semihosting.	False
semihosting_enabled	Enable or disable semihosting.  ———— <b>Note</b> ————— If you disable semihosting, the model hard faults if it encounters a semihosting BKPT instruction.	True
semi_heap_base	Semihosting heap base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x0
semi_heap_limit	Semihosting heap limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0xF000000
semi_stack_base	Semihosting stack base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x10000000
semi_stack_limit	Semihosting stack limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0xF000000
warn_extra	Print additional warning messages.	False
warn_undefined	Print a warning when an undefined instruction is hit.	False

## B.2.4 Cortex-M0 memory map

Figure B-2 shows the default memory map for the Cortex-M0 model, which is a flat 4GB memory space.

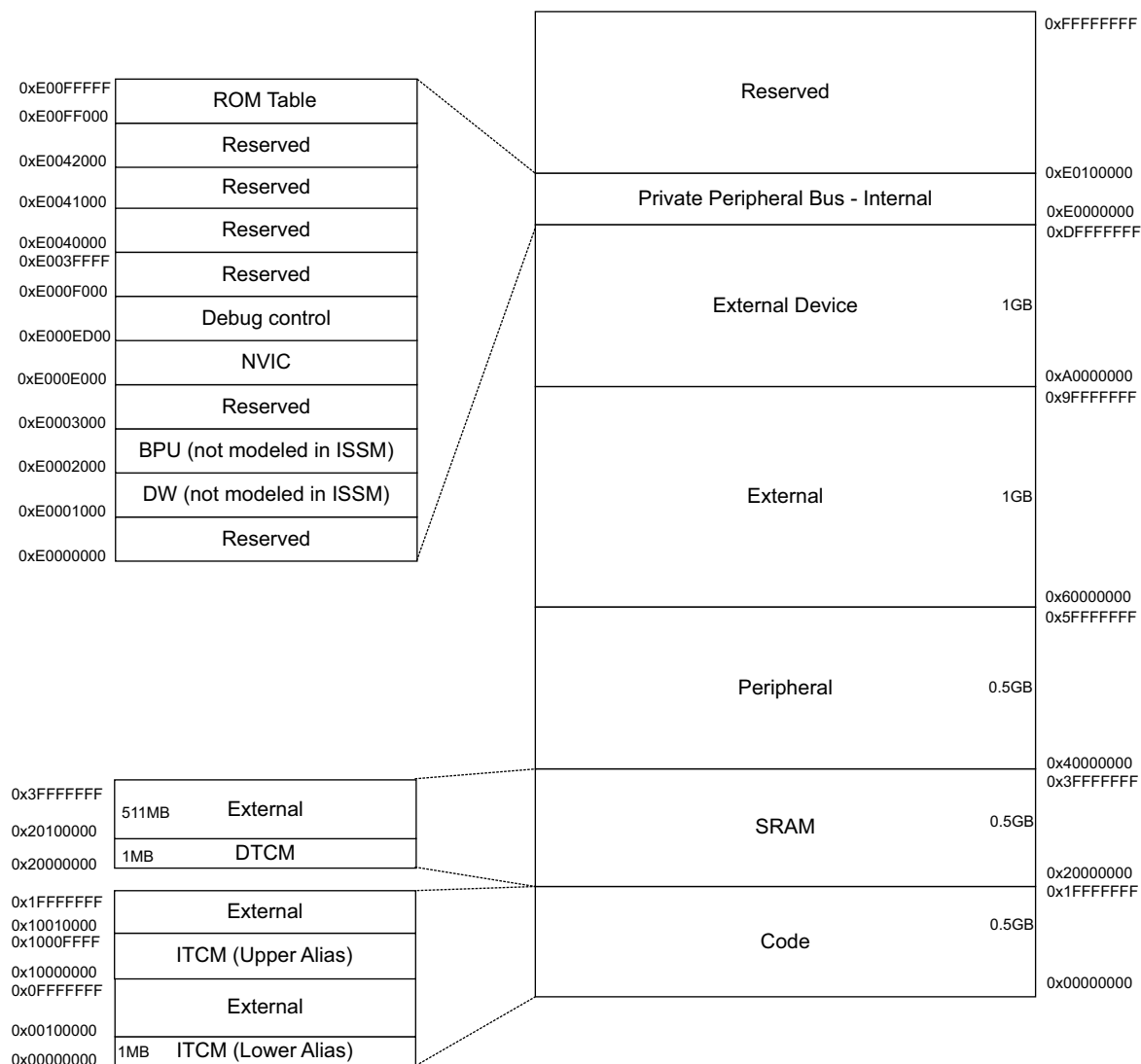


Figure B-2 Cortex-M0 memory map

## B.2.5 Cortex-M0 peripheral register mapping

Table B-9 shows the peripheral register mapping for the Cortex-M0 model (see Figure B-2 on page B-12). The functional peripherals are NVIC, three Timers, and one UART. These peripherals are described in the following sections. All other peripherals are represented by a basic register array in the memory map regions `0x1nnnnnnn` and `0xCnnnnnnn`.

To access the peripheral registers, select the corresponding tab in the Registers view.

**Table B-9 Cortex-M0 peripheral register mapping**

Base Address	Peripheral
0x40005000	Integrator CP Triple Timer
0x40018000	UART0 (Feature restricted)
0xE000E000	NVIC (Nested Vectored Interrupt Controller)

### Integrator CP Triple Timer

Table B-10 shows the Integrator CP Triple Timer registers of the Cortex-M0 model.

**Table B-10 Cortex-M0 Timer registers**

Offset	Register
0x000	Timer0Load
0x018	Timer0BGLoad
0x004	Timer0Value
0x008	Timer0Control
0x00C	Timer0IntClr
0x010	Timer0RIS
0x014	Timer0MIS
0x100	Timer1Load
0x118	Timer1BGLoad
0x104	Timer1Value
0x108	Timer1Control
0x10C	Timer1IntClr
0x110	Timer1RIS
0x114	Timer1MIS
0x200	Timer2Load
0x218	Timer2BGLoad
0x204	Timer2Value
0x208	Timer2Control
0x20C	Timer2IntClr

**Table B-10 Cortex-M0 Timer registers (continued)**

Offset	Register
0x210	Timer2RIS
0x214	Timer2MIS
0xFE0	Timer_PeriphID0
0xFE4	Timer_PeriphID1
0xFE8	Timer_PeriphID2
0xFEC	Timer_PeriphID3
0xFF0	Timer_PCellID0
0xFF4	Timer_PCellID1
0xFF8	Timer_PCellID2
0xFFC	Timer_PCellID3

The three timers count down by only one tick for each instruction executed. The timers are mapped to the same memory locations as the timers on the Integrator/CP development board.

---

**Note**


---

The Timer\_PCellID and Timer\_PeripID registers are not supported, but this does not affect the operation of the peripheral because it conforms to the ARM Generic Peripheral specification for Timers.

---

## UART (PL011)

Table B-11 shows the UART0 registers of the Cortex-M0 model.

**Table B-11 Cortex-M0 UART registers**

Offset	Register
0x000	UARTDR
0x004	UARTSR_ECR
0x018	UARTFR
0x020	UARTILPR
0x024	UARTIBRD
0x028	UARTFBRD
0x02C	UARTLCR_H
0x030	UARTCR
0x034	UARTIFLS
0x038	UARTIMSC
0x03C	UARTRIS
0x040	UARTMIS

**Table B-11 Cortex-M0 UART registers (continued)**

Offset	Register
0x044	UARTICR
0x048	UARTDMACR
0xFE0	UARTPeriphID0
0xFE4	UARTPeriphID1
0xFE8	UARTPeriphID2
0xFEC	UARTPeriphID3
0xFF0	UARTPCe11ID0
0xFF4	UARTPCe11ID1
0xFF8	UARTPCe11ID2
0xFFC	UARTPCe11ID3

The Cortex-M0 models a serial port that can be used for I/O. This is modeled through a telnet window, which open automatically on the first read/write operation from the UART after it has been initialized.

The serial port is mapped to the same memory location as the UART0 on the Integrator/CP development board.

#### **Note**

The default behavior for the telnet protocol is line mode. This means that characters typed into the telnet window are not seen in the model until you press Enter. To change this behavior type the telnet escape character, usually Ctrl+], and then enter the mode character. Doing this might disable local echo of the characters entered in the telnet window.

## **Interrupts**

Table B-12 shows the supported interrupts for the Cortex-M0 model.

**Table B-12 Cortex-M0 interrupts**

Interrupt	Peripheral
1	UART0
3	Reserved
4	Reserved
5	Timer0
6	Timer1
7	Timer2
8	Reserved



**See also**

- the following in the *RealView Debugger User Guide*:
  - *Viewing registers* on page 13-25.

## B.3 Cortex-M1 model configuration

The following sections describe the configuration details of the Cortex-M1 model:

- *Summary of supported features for the Cortex-M1 model*
- *Considerations when using the Cortex-M1 model*
- *Limitations of the Cortex-M1 model* on page B-18
- *Cortex-M1 configuration parameters* on page B-19
- *Cortex-M1 memory map* on page B-20
- *Cortex-M1 peripheral register mapping* on page B-21.

### B.3.1 Summary of supported features for the Cortex-M1 model

The following features are supported by the Cortex-M1 model:

- The Cortex-M1 model can execute all the Thumb-1 (THUMBv3) instructions supported by Cortex-M1 processor.
- Execution performance is a minimum of 6 MIPS on a 2GHz Intel workstation.
- The model includes the following peripherals:
  - one UART
  - three Timers
  - *Nested Vectored Interrupt Controller (NVIC)*.
- *Tightly-Coupled Memory (TCM)* components are supported as separate memory spaces with configurable size.
- On reset, the model behaves similar to hardware. Therefore, initial SP and PC values must be loaded at address 0x00000000, followed by system handler and interrupt vectors. An example is provided in the following directory to illustrate the use of the SP/PC addresses and Cortex-M1 libraries:

```
install_directory\RVDs\Examples\...\platform\Cortex-M1
```

#### See also

- *Cortex-M1 Technical Reference Manual*
- *Integrator CP User Guide* for details of Timers
- *PrimeCell UART (PL011) Technical Reference Manual* for details of UARTs.

### B.3.2 Considerations when using the Cortex-M1 model

Be aware of the following when using the Cortex-M1 model:

- By default, the Cortex-M1 model starts with the *Instruction Tightly-Coupled Memory (ITCM)* Lower Alias memory region enabled. Therefore, when you load an ELF image, it is loaded to both that ITCM and the backing-memory.
- Debug writes to the ITCM Lower Alias memory region go to the ITCM (if enabled) and to the backing-memory.

### B.3.3 Limitations of the Cortex-M1 model

The Cortex-M1 model has the following limitations:

- The FPB and DWT debug components are not modeled.
- The debug and trace hardware is not modeled.
- The halt-mode debug is not modeled. Consequently, using BKPT results in the model stopping for the following conditions:
  - when `semihosting_enabled` is `False`
  - when `semihosting_enabled` is `True` and the BKPT immediate value does not match the `semihosting_BKPT` value.

In both cases, the following error message is displayed:

[core] Simulation terminated: Using BKPT for halt-debug is unsupported

- Lockup conditions are modeled as end-of-simulation conditions. This is because the model system does not include any peripherals to detect and correct these exception cases. See the section that describes unrecoverable exception cases in the *ARMv6-M Architecture Reference Manual*.
- Late-arrival preemption is not modeled. This is because exception entry and exit are modeled as atomic transitions. Therefore, it is impossible for an asynchronous exception to be delivered within exception entry. See the section that describes exceptions on exception entry in the *ARMv6-M Architecture Reference Manual*.

### B.3.4 Cortex-M1 configuration parameters

Table B-13 describes the configuration parameters for the Cortex-M1 model.

**Table B-13 Cortex-M1 model configuration parameters**

Parameter	Description	Default
BIGEND	Run in big-endian (BE8) mode.	False
CFGITCMEN	Reset values of AuxControl <i>Instruction Tightly-Coupled Memory</i> (ITCM) alias enable bits.	0x1
dbg_extension	Use debug extensions.	False
dtdcm_size	Size of <i>Data Tightly-Coupled Memory</i> (DTCM) in bytes as a power of 2 (0, 4, 8, 16, 32, 64, 128, 256, 512, to 1024Kb).	0xC (4Kb)
itcm_size	Size of ITCM in bytes as a power of 2 (0, 4, 8, 16, 32, 64, 128, 256, 512, to 1024Kb).	0xC (4Kb)
os_extension	Use OS extensions.	True
semlhosting_BKPT	The breakpoint used for semihosting. The value must be in the range 0 to 255 (0xFF).	0xAB
semlhosting_cmd_line	Semihosting command line string used for passing command-line arguments to an image. Separate each argument with a space, and quote any arguments that include a space, for example: argument1 "argument 2" argument_3	
semlhosting_debug	Print verbose messages for semihosting.	False
semlhosting_enabled	Enable or disable semihosting.	True
	<p><b>Note</b></p> <p>If you disable semihosting, the model hard faults if it encounters a semihosting BKPT instruction.</p>	
semi_heap_base	Semihosting heap base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x0
semi_heap_limit	Semihosting heap limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0xF000000
semi_stack_base	Semihosting stack base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x10000000
semi_stack_limit	Semihosting stack limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0xF000000
warn_extra	Print additional warning messages.	False
warn_undefined	Print a warning when an undefined instruction is hit.	False



### B.3.6 Cortex-M1 peripheral register mapping

Table B-14 shows the peripheral register mapping for the Cortex-M1 model (see Figure B-3 on page B-20). The functional peripherals are NVIC, three Timers, and one UART. These peripherals are described in the following sections. All other peripherals are represented by a basic register array in the memory map regions `0x1nnnnnnn` and `0xCnnnnnnn`.

To access the peripheral registers, select the corresponding tab in the Registers view.

**Table B-14 Cortex-M1 peripheral register mapping**

Base Address	Peripheral
0x40005000	Integrator CP Triple Timer
0x40018000	UART0 (Feature restricted)
0xE000E000	NVIC (Nested Vectored Interrupt Controller)

#### Integrator CP Triple Timer

Table B-15 shows the Integrator CP Triple Timer registers of the Cortex-M1 model.

**Table B-15 Cortex-M1 Timer registers**

Offset	Register
0x000	Timer0Load
0x018	Timer0BGLoad
0x004	Timer0Value
0x008	Timer0Control
0x00C	Timer0IntClr
0x010	Timer0RIS
0x014	Timer0MIS
0x100	Timer1Load
0x118	Timer1BGLoad
0x104	Timer1Value
0x108	Timer1Control
0x10C	Timer1IntClr
0x110	Timer1RIS
0x114	Timer1MIS
0x200	Timer2Load
0x218	Timer2BGLoad
0x204	Timer2Value
0x208	Timer2Control
0x20C	Timer2IntClr

**Table B-15 Cortex-M1 Timer registers (continued)**

Offset	Register
0x210	Timer2RIS
0x214	Timer2MIS
0xFE0	Timer_PeriphID0
0xFE4	Timer_PeriphID1
0xFE8	Timer_PeriphID2
0xFEC	Timer_PeriphID3
0xFF0	Timer_PCellID0
0xFF4	Timer_PCellID1
0xFF8	Timer_PCellID2
0xFFC	Timer_PCellID3

The three timers count down by only one tick for each instruction executed. The timers are mapped to the same memory locations as the timers on the Integrator/CP development board.

---

**Note**

---

The Timer\_PCellID and Timer\_PeripID registers are not supported, but this does not affect the operation of the peripheral because it conforms to the ARM Generic Peripheral specification for Timers.

---

## UART (PL011)

Table B-16 shows the UART0 registers of the Cortex-M1 model.

**Table B-16 Cortex-M1 UART registers**

Offset	Register
0x000	UARTDR
0x004	UARTSR_ECR
0x018	UARTFR
0x020	UARTILPR
0x024	UARTIBRD
0x028	UARTFBRD
0x02C	UARTLCR_H
0x030	UARTCR
0x034	UARTIFLS
0x038	UARTIMSC
0x03C	UARTRIS
0x040	UARTMIS

**Table B-16 Cortex-M1 UART registers (continued)**

Offset	Register
0x044	UARTICR
0x048	UARTDMACR
0xFE0	UARTPeriphID0
0xFE4	UARTPeriphID1
0xFE8	UARTPeriphID2
0xFEC	UARTPeriphID3
0xFF0	UARTPCe11ID0
0xFF4	UARTPCe11ID1
0xFF8	UARTPCe11ID2
0xFFC	UARTPCe11ID3

The Cortex-M1 models a serial port that can be used for I/O. This is modeled through a telnet window, which open automatically on the first read/write operation from the UART after it has been initialized.

The serial port is mapped to the same memory location as the UART0 on the Integrator/CP development board.

#### **Note**

The default behavior for the telnet protocol is line mode. This means that characters typed into the telnet window are not seen in the model until you press Enter. To change this behavior type the telnet escape character, usually Ctrl+], and then enter the mode character. Doing this might disable local echo of the characters entered in the telnet window.

## **Interrupts**

Table B-17 shows the supported interrupts for the Cortex-M1 model.

**Table B-17 Cortex-M1 interrupts**

Interrupt	Peripheral
1	UART0
3	Reserved
4	Reserved
5	Timer0
6	Timer1
7	Timer2
8	Reserved



**See also**

- the following in the *RealView Debugger User Guide*:
  - *Viewing registers* on page 13-25.

## B.4 Cortex-M3 model configuration

The Cortex-M3 model in this release has been modified from that in earlier releases. The following sections describe the configuration details of the Cortex-M3 revision 1 model:

- *Summary of supported features for the Cortex-M3 model*
- *Limitations of the Cortex-M3 model*
- *Cycle and instruction counting feature* on page B-26
- *Cortex-M3 configuration parameters* on page B-27
- *Cortex-M3 memory map* on page B-28
- *Cortex-M3 peripheral register mapping* on page B-29.

### B.4.1 Summary of supported features for the Cortex-M3 model

The following features are supported by Cortex-M3 model:

- The Cortex-M3 model can execute all the instructions supported by Cortex-M3 processor.
- Execution performance is a minimum of 3 MIPS on a 2GHz Intel workstation.
- The behavior of the *Nested Vectored Interrupt Controller* (NVIC), *MPU*, *Flash Patch and Breakpoint unit* (FPB) and bus-matrix are modeled.
- The model includes the following peripherals:
  - one UART
  - three Timers
  - Interrupt Controller (provided by the NVIC).
- On reset, the model behaves similar to hardware. Therefore, initial SP and PC values must be loaded at address 0x0, followed by system-handler/interrupt vectors. An example is provided in the following directory to illustrate the use of the SP/PC addresses and Cortex-M3 libraries:
 

```
install_directory\RVDs\Examples\...\platform\Cortex-M3
```
- Cycle and instruction counting are supported.

#### See also

- *Cycle and instruction counting feature* on page B-26
- *Cortex-M3 Technical Reference Manual*.

### B.4.2 Limitations of the Cortex-M3 model

The Cortex-M3 model has the following limitations:

- The *Debug Access Port* (DAP), *Embedded Trace Macrocell™* (ETM™) and *Instrumentation Trace Macrocell* (ITM) are not modeled.
- The debug hardware is not modeled, except for some of the data watchpoint (hardware data breakpoints) and triggering.

### B.4.3 Cycle and instruction counting feature

A **Statistics** tab is available in the Registers view, and contains the registers shown in Table B-18.

**Table B-18 Cortex-M3 cycle count registers and symbols**

Register	Symbol
approximate_cycle_count	@Statistics_APPROXIMATE_CYCLE_COUNT
Instructions	@Statistics_INSTRUCTIONS

#### Considerations with cycle and instruction counts

Be aware of the following when viewing cycle and instruction counts:

- the counts assume dual-port, zero-wait-state RAM
- the counts are not reset to zero when the you set the PC to the image entry point.

## B.4.4 Cortex-M3 configuration parameters

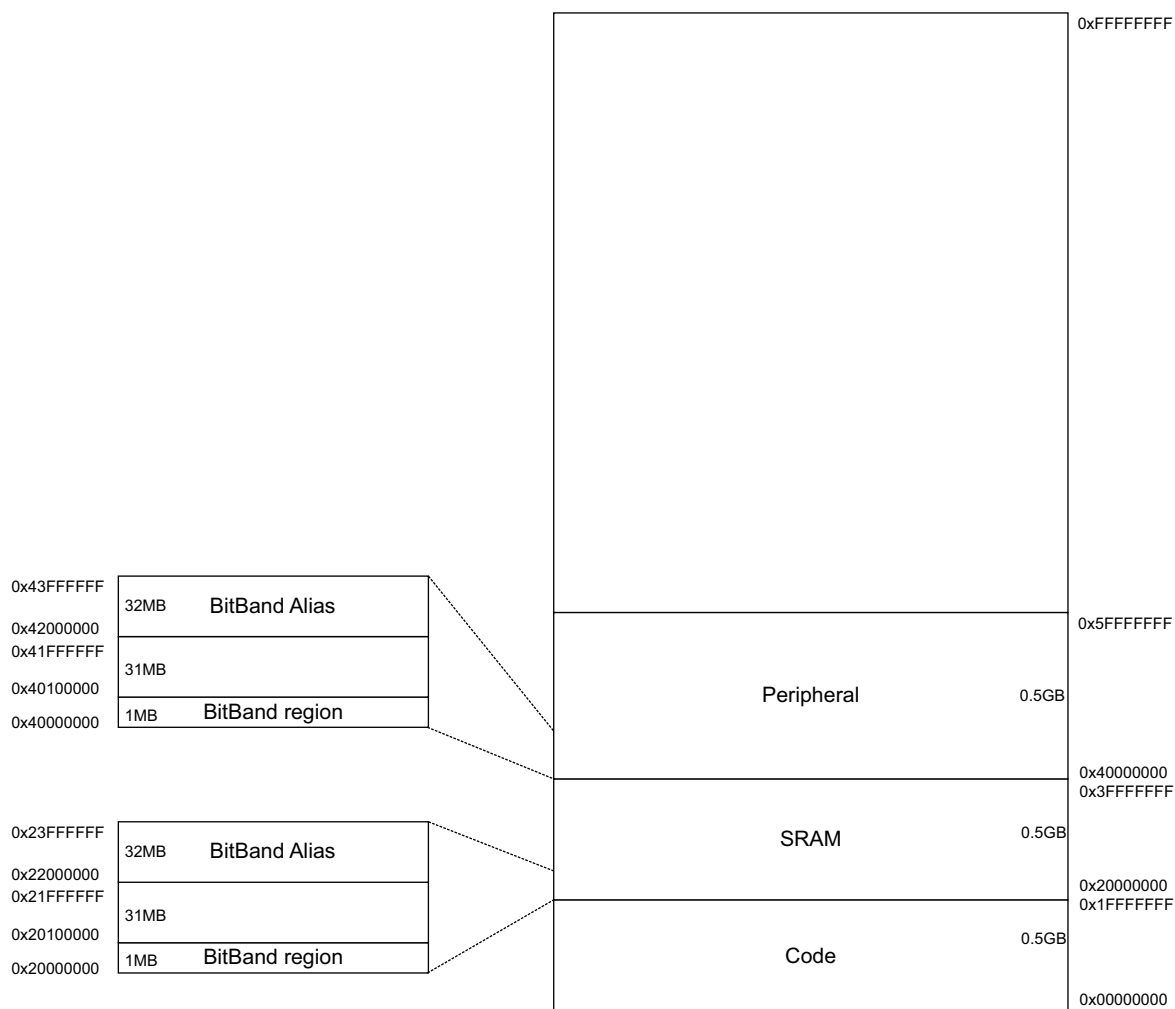
Table B-19 describes the configuration parameters for the Cortex-M3 model.

**Table B-19 Cortex-M3 model configuration parameters**

Parameter	Description	Default
BIGEND	Run in big-endian (BE8) mode.	False
clock_frequency	Clock frequency used in semihosting SYS_CLOCK.	0x5f5e100
LvlWidth	Number of bits in the interrupt Priority.	0x3
NumIRQ	Number of non-system interrupts. The value must be in the range 1 to 240.	0xF0 (240)
semihosting_BKPT	The BKPT instruction that is to be intercepted for semihosting.	0xAB
semihosting_cmd_line	Semihosting command line string used for passing command-line arguments to an image. Separate each argument with a space, and quote any arguments that include a space, for example: argument1 "argument 2" argument_3	
semihosting_debug	Print verbose messages for semihosting.	False
semihosting_enabled	Enable or disable semihosting.	True
<p style="text-align: center;"><b>————— Note —————</b></p> <p>If you disable semihosting, the model hard faults if it encounters the semihosting BKPT instruction.</p>		
semihosting_heap_base	Semihosting heap base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x20100000
semihosting_heap_limit	Semihosting heap limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x21800000
semihosting_stack_base	Semihosting stack base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x22000000
semihosting_stack_limit	Semihosting stack limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x21800000

### B.4.5 Cortex-M3 memory map

Figure B-4 shows the memory map for the Cortex-M3 model, which is a flat 4GB memory space. The memory map includes bit-band regions, which occupy the lowest 1MB of SRAM and Peripheral memory.



**Figure B-4 Cortex-M3 memory map**

#### See also

- *Cortex-M3 Technical Reference Manual* for details of bit-banding regions.

## B.4.6 Cortex-M3 peripheral register mapping

Table B-20 shows the peripheral register mapping for the Cortex-M3 model (see Figure B-4 on page B-28). The functional peripherals are three Timers and one UART. These peripherals are described in the following sections.

To access the peripheral registers, select the corresponding tab in the Registers view.

**Table B-20 Cortex-M3 peripheral register mapping**

Address	Peripheral
0x40018000 - 0x40018FFF	Peripheral Bus out (UART)
0x40050000 - 0x4005FFFF	Peripheral Bus out (Timer)

### Note

The rest of the peripheral memory space is unimplemented.

## Timers

Table B-21 shows the Timer registers of the Cortex-M3 model.

**Table B-21 Cortex-M3 Timer registers**

Offset	Register
0x000	TimerLoad0
0x004	TimerValue0
0x008	TimerControl0
0x00C	TimerIntClr0
0x010	TimerRIS0
0x014	TimerMIS0
0x018	TimerBGLoad0
0x100	TimerLoad1
0x104	TimerValue1
0x108	TimerControl1
0x10C	TimerIntClr1
0x110	TimerRIS1
0x114	TimerMIS1
0x118	TimerBGLoad1
0x200	TimerLoad2
0x204	TimerValue2
0x208	TimerControl2
0x20C	TimerIntClr2

**Table B-21 Cortex-M3 Timer registers (continued)**

Offset	Register
0x210	TimerRIS2
0x214	TimerMIS2
0x218	TimerBGLoad2
0xFE0	Timer PeriphID0
0xFE4	Timer PeriphID1
0xFE8	Timer PeriphID2
0xFEC	Timer PeriphID3
0xFF0	Timer PCellID0
0xFF4	Timer PCellID1
0xFF8	Timer PCellID2
0xFFC	Timer PCellID3

The three timers count down by only one tick for each instruction executed. The timers are mapped to the same memory locations as the timers on the Integrator™/CP development board.

---

**Note**

The NVIC timer counts one tick per notional cycle. This differs from the other per instruction timers, for example, for LDM and STM instructions.

---



---

**Note**

The Timer PCellID $n$  and Timer PeriphID $n$  registers are not supported, but this does not affect the operation of the peripheral because it conforms to the ARM Generic Peripheral specification for Timers.

---

## UART

Table B-22 shows the UART registers of the Cortex-M3 model.

**Table B-22 Cortex-M3 UART registers**

Offset	Register
0x000	UARTDR
0x004	UARTSR_ECR
0x018	UARTFR
0x020	UARTILPR
0x024	UARTIBRD
0x028	UARTFBRD
0x02C	UARTLCR_H
0x030	UARTCR

**Table B-22 Cortex-M3 UART registers (continued)**

Offset	Register
0x034	UARTIFLS
0x038	UARTIMSC
0x03C	UARTRIS
0x040	UARTMIS
0x044	UARTICR
0x048	UARTDMACR
0xFE0	UARTPeriphID0
0xFE4	UARTPeriphID1
0xFE8	UARTPeriphID2
0xFEC	UARTPeriphID3
0xFF0	UARTPCe11ID0
0xFF4	UARTPCe11ID1
0xFF8	UARTPCe11ID2
0xFFC	UARTPCe11ID3

The Cortex-M3 models one serial port that can be used for I/O. This is modeled through a telnet window, which opens automatically on the first read/write operation from the UART after it has been initialized.

The serial port is mapped to the same memory location as the UART on the Integrator/CP development board.

---

#### **Note**

The default behavior for the telnet protocol is line mode. This means that characters typed into the telnet window are not seen in the model until you press Enter. To change this behavior type the telnet escape character, usually Ctrl+], and then enter the mode character. Doing this might disable local echo of the characters entered in the telnet window.

---

## **Interrupts**

Table B-23 shows the supported interrupts for the Cortex-M3 model.

**Table B-23 Cortex-M3 interrupts**

Interrupt	Peripheral
1	UART
5	Timer0
6	Timer1
7	Timer2



**See also**

- the following in the *RealView Debugger User Guide*:
  - *Viewing registers* on page 13-25.

## B.5 Cortex-R4 model configuration

The following sections describe the configuration details of the Cortex-R4 model:

- *Summary of supported features for the Cortex-R4 model*
- *Limitations of the Cortex-R4 model*
- *Cortex-R4 configuration parameters* on page B-34
- *Cortex-R4 memory map* on page B-37
- *Cortex-R4 peripheral register mapping* on page B-38.

### B.5.1 Summary of supported features for the Cortex-R4 model

The following features are supported by the Cortex-R4 model:

- The model is compatible with the current uCLinux port for the Integrator/CP.
- Execution performance is a minimum of 4 MIPS on a 2GHz Intel workstation.
- All CP15 controls are modeled, except for CP15 register 15 (internal TLB registers).
- The model includes the following peripherals, but only when the *Memory Protection Unit* (MPU) is enabled:
  - *Primary Interrupt Controller* (PIC)
  - two UARTs,
  - three Timers
  - Watchdog (WDOG)
  - Real-Time Clock (RTC)
  - *TrustZone® Interrupt Controller* (TZIC)
  - *Nested Vectored Interrupt Controller* (NVIC)
  - *General Interrupt Controller* (GIC).
- TCM components are supported as separate memory spaces with configurable size. However, it does not support the external TCM interfaces and the AXI-Slave port specified in the device.
- On reset, the model behaves similar to hardware. For example, with caches and VFP disabled, appropriate initialization code is required to set these up. Suitable initialization code in assembler is provided in the following directory:
 

```
install_directory\RVDS\Examples\...\..\..\cached_dhry\CortexR4dhry
```

#### See also

- *Integrator CP User Guide* for details of PIC and Timers
- *PL011 UART Technical Reference Manual* for details of UARTs.

### B.5.2 Limitations of the Cortex-R4 model

The debug and trace hardware is not modeled on the Cortex-R4 model.

### B.5.3 Cortex-R4 configuration parameters

Table B-24 describes the configuration parameters for the Cortex-R4 model.

**Table B-24 Cortex-R4 model configuration parameters**

Parameter	Description	Default
cfgnmfi	Configure non-maskable Fast Interrupts.	False
dcachemax	Encodings for the data cache maximum size:	0xF
	<b>0x0</b> 4KB	
	<b>0x1</b> 8KB	
	<b>0x3</b> 16KB	
	<b>0x7</b> 32KB	
	<b>0xf</b> 64KB	
dcachemin	Encodings for the data cache minimum size:	0x0
	<b>0x0</b> 4KB	
	<b>0x1</b> 8KB	
	<b>0x3</b> 16KB	
	<b>0x7</b> 32KB	
	<b>0xf</b> 64KB	
dtcmsize	Encodings for the DTCM size:	0xE
	<b>0x0</b> 0KB	
	<b>0x3</b> 4KB	
	<b>0x4</b> 8KB	
	<b>0x5</b> 16KB	
	<b>0x6</b> 32KB	
	<b>0x7</b> 64KB	
	<b>0x8</b> 128KB	
	<b>0x9</b> 256KB	
	<b>0xA</b> 512KB	
	<b>0xB</b> 1MB	
	<b>0xC</b> 2MB	
	<b>0xD</b> 4MB	
	<b>0xE</b> 8MB	
hivecs	Let exception vectors start at 0xFFFF0000.	False
icachemax	Encodings for the instruction cache maximum size:	0xF
	<b>0x0</b> 4KB	
	<b>0x1</b> 8KB	
	<b>0x3</b> 16KB	
	<b>0x7</b> 32KB	
	<b>0xf</b> 64KB	
icachemin	Encodings for the instruction cache minimum size:	0x0
	<b>0x0</b> 4KB	
	<b>0x1</b> 8KB	
	<b>0x3</b> 16KB	
	<b>0x7</b> 32KB	
	<b>0xf</b> 64KB	
initee	Set data endianness.	False

Table B-24 Cortex-R4 model configuration parameters (continued)

Parameter	Description	Default
initie	Set instruction endianness.	False
initramd	Enable DTCM at reset.	False
initrami	Enable ITCM at reset.	False
itcmsize	Encodings for the ITCM size: <b>0x0</b> 0KB <b>0x3</b> 4KB <b>0x4</b> 8KB <b>0x5</b> 16KB <b>0x6</b> 32KB <b>0x7</b> 64KB <b>0x8</b> 128KB <b>0x9</b> 256KB <b>0xA</b> 512KB <b>0xB</b> 1MB <b>0xC</b> 2MB <b>0xD</b> 4MB <b>0xE</b> 8MB	0xE
loczrami	Set initial ITCM offset: <b>False</b> ITCM offset is 0x0 DTCM offset is 0x40000000 <b>True</b> ITCM offset is 0x40000000 DTCM offset is 0x0	False
mpuregions	Number of MPU regions (0, 8, or 12)	0xC
no_entcm1if	Disable B1 TCM interface	False
nodcache	No Data Cache	False
nofpu	Support floating-point.	False
noicache	No Instruction Cache	False
noie	Disable big-endian instruction support	False
randomfiq	Enable random FIQ	False
randomirq	Enable random IRQ	False
rmwenram	Enable read-modify-write for TCM interfaces	False
saveAndRestore-enable	When disabled the model: <ul style="list-style-type: none"> <li>can not save its state for a future session</li> <li>can not pick up a saved state from a previous session.</li> </ul> There is a minimal overhead on the performance of the simulation when this is enabled.	False
semihosting-ARM_SVC	The ARM SVC used for semihosting. The value must be in the range 0 to 16777215 (0xFFFFFFFF).	0x123456

**Table B-24 Cortex-R4 model configuration parameters (continued)**

Parameter	Description	Default
semihosting-clock_frequency	Simulated clock frequency in MHz, which uses the elapsed time based on the number of instructions executed. The value must be in the range 1 to 4294967295 (0xFFFFFFFF).  ———— <b>Note</b> ———— The number you specify has no relationship to the hardware processor.	0x32
semihosting-cmd_line	Semihosting command line string used for passing command-line arguments to an image. Separate each argument with a space, and quote any arguments that include a space, for example: argument1 "argument 2" argument_3	
semihosting-debug	Print verbose messages for semihosting.	False
semihosting-enable	Enable or disable semihosting.	True
semihosting-Thumb_SVC	The Thumb SVC used for semihosting. The value must be in the range 0 to 255 (0xFF).	0xAB
semihosting-heap_base	Semihosting heap base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x0
semihosting-heap_limit	Semihosting heap limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0xF000000
semihosting-stack_base	Semihosting stack base. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0x10000000
semihosting-stack_limit	Semihosting stack limit. The value must be in the range 0 to 4294967295 (0xFFFFFFFF).	0xF000000
sldtcmsh	Use MSB of the address to select D1 TCM, otherwise bit[3].	False
tcmhiinitaddr	TCM High initialization address offset.	0x40000000
teinit	Exception handling state. Set processor to Thumb mode when entering an exception.	False
warn-extra	Print some useful additional warning messages.	False
warn-undefined	Print a warning when an undefined instruction is hit.	False

### B.5.4 Cortex-R4 memory map

Figure B-5 shows the default memory map for the Cortex-R4 model, which is a flat 4GB memory space.

**Note**

Peripheral regions are accessible only when the MPU is enabled.

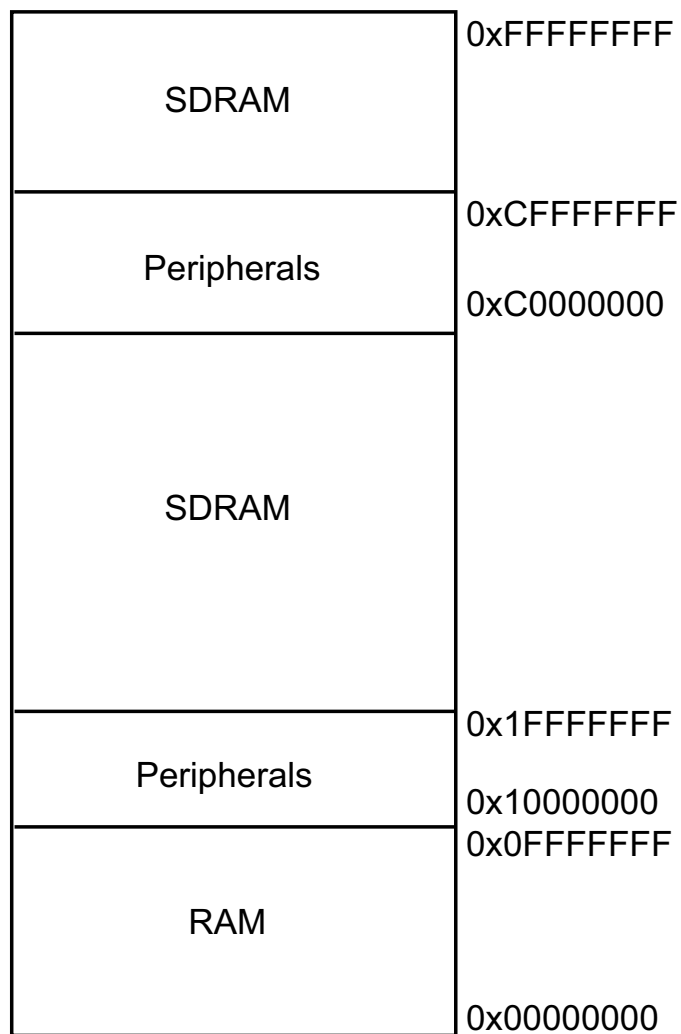


Figure B-5 Cortex-R4 memory map

## B.5.5 Cortex-R4 peripheral register mapping

Table B-25 shows the peripheral register mapping for the Cortex-R4 model (see Figure B-5 on page B-37). The functional peripherals are two UARTs, one Timer, a Watchdog, RTC, System Control, NVIC, *TrustZone Interrupt Controller* (TZIC), and *General Interrupt Controller* (GIC). These peripherals are described in the following sections. All other peripherals are represented by a basic register array in the memory map regions `0x1nnnnnnn` and `0xCnnnnnnn`.

### Note

Peripheral locations are accessible only when the MPU is enabled.

To access the peripheral registers, select the corresponding tab in the Registers view.

**Table B-25 Cortex-R4 peripheral register mapping**

Base Address	Peripheral
0x10009000	UART0 (Feature restricted)
0x1000A000	UART1
0x10011000	Timer
0x10010000	WDOG (Watchdog)
0x10017000	RTC (Real-Time Clock)
0x10001000	System Control (clock control for timers, configuration options, and remap signal)
Not applicable	NVIC (Nested Vectored Interrupt Controller)
0x10040000	TZIC (TrustZone Interrupt Controller)
0x10050000	GIC (General Interrupt Controller)

## Primary Interrupt Controller

Table B-26 shows the PIC registers of the Cortex-R4 model.

**Table B-26 Cortex-R4 PIC registers**

Offset	Register
0x00	PIC_IRQ_STATUS
0x04	PIC_IRQ_RAWSTAT
0x08	PIC_IRQ_ENABLE
0x10	PIC_INT_SOFTSET
0x14	PIC_INT_SOFTCLR
0x20	PIC_FIQ_STATUS
0x24	PIC_FIQ_RAWSTAT
0x28	PIC_FIQ_ENABLE

## Timers

Table B-27 shows the Timer registers of the Cortex-R4 model.

**Table B-27 Cortex-R4 Timer registers**

Offset	Register
0x000	Timer0Load
0x018	Timer0BGLoad
0x004	Timer0Value
0x008	Timer0Control
0x00C	Timer0IntClr
0x010	Timer0RIS
0x014	Timer0MIS
0x100	Timer1Load
0x118	Timer1BGLoad
0x104	Timer1Value
0x108	Timer1Control
0x10C	Timer1IntClr
0x110	Timer1RIS
0x114	Timer1MIS
0x200	Timer2Load
0x218	Timer2BGLoad
0x204	Timer2Value
0x208	Timer2Control
0x20C	Timer2IntClr
0x210	Timer2RIS
0x214	Timer2MIS
0xFE0	Timer_PeriphID0
0xFE4	Timer_PeriphID1
0xFE8	Timer_PeriphID2
0xFEC	Timer_PeriphID3
0xFF0	Timer_PCellID0
0xFF4	Timer_PCellID1
0xFF8	Timer_PCellID2
0xFFC	Timer_PCellID3



The three timers count down by only one tick for each instruction executed. The timers are mapped to the same memory locations as the timers on the Integrator™/CP development board.

---

**Note**

---

The Timer\_PCellID and Timer\_PeripID registers are not supported, but this does not affect the operation of the peripheral because it conforms to the ARM Generic Peripheral specification for Timers.

---

## Real-Time Clock (PL030)

Table B-28 shows the RTC registers of the Cortex-R4 model.

**Table B-28 Cortex-R4 RTC registers**

Offset	Register
0x000	RTC_DR
0x004	RTC_MR
0x008	RTC_STAT
0x00C	RTC_CLR
0x010	RTC_CR
0xFE0	RTC_PeriphID0
0xFE4	RTC_PeriphID1
0xFE8	RTC_PeriphID2
0xFEC	RTC_PeriphID3
0xFF0	RTC_PCellID0
0xFF4	RTC_PCellID1
0xFF8	RTC_PCellID2
0xFFC	RTC_PCellID3

## UART (PL011)

Table B-29 shows the UART0 and UART1 registers of the Cortex-R4 model.

**Table B-29 Cortex-R4 UART registers**

Offset	Register
0x000	UARTDR
0x004	UARTSR_ECR
0x018	UARTFR
0x020	UARTILPR
0x024	UARTIBRD
0x028	UARTFBRD

**Table B-29 Cortex-R4 UART registers (continued)**

Offset	Register
0x02C	UARTLCR_H
0x030	UARTCR
0x034	UARTIFLS
0x038	UARTIMSC
0x03C	UARTRIS
0x040	UARTMIS
0x044	UARTICR
0x048	UARTDMACR
0xFE0	UARTPeriphID0
0xFE4	UARTPeriphID1
0xFE8	UARTPeriphID2
0xFEC	UARTPeriphID3
0xFF0	UARTPCe11ID0
0xFF4	UARTPCe11ID1
0xFF8	UARTPCe11ID2
0xFFC	UARTPCe11ID3

The Cortex-R4 models two serial ports that can be used for I/O. These are modeled through telnet windows, which open automatically on the first read/write operation from the UART after it has been initialized.

The serial ports are mapped to the same memory locations as the UARTs on the Integrator/CP development board.

#### ———— **Note** ————

The default behavior for the telnet protocol is line mode. This means that characters typed into the telnet window are not seen in the model until you press Enter. To change this behavior type the telnet escape character, usually Ctrl+], and then enter the mode character. Doing this might disable local echo of the characters entered in the telnet window.

## Interrupts

Table B-30 shows the supported interrupts for the Cortex-R4 model.

**Table B-30 Cortex-R4 interrupts**

Interrupt	Peripheral
1	UART0
2	UART1
3	Reserved

**Table B-30 Cortex-R4 interrupts (continued)**

Interrupt	Peripheral
4	Reserved
5	Timer0
6	Timer1
7	Timer2
8	Reserved

**See also**

- the following in the *RealView Debugger User Guide*:
  - *Viewing registers* on page 13-25.