

RealView[®] Debugger

Version 1.8

Target Configuration Guide



RealView Debugger

Target Configuration Guide

Copyright © 2002-2004 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Change
April 2002	A	RealView Debugger v1.5 release
September 2002	B	RealView Debugger v1.6 release
February 2003	C	RealView Debugger v1.6.1 release
September 2003	D	RealView Debugger v1.6.1 release for RVDS v2.0
January 2004	E	RealView Debugger v1.7 release for RVDS v2.1
December 2004	F	RealView Debugger v1.8 release for RVDS v2.2

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by ARM Limited. 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.

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

Preface

About this book	viii
Feedback	xii

Chapter 1

Introduction

1.1	About target connections and configuration	1-2
1.2	Viewing board file properties	1-5
1.3	Configuration files	1-12

Chapter 2

Connecting to Targets

2.1	Working with the Connection Control window	2-2
2.2	Managing connections	2-6
2.3	Connecting to a target	2-11
2.4	Failing to make a connection	2-19
2.5	Disconnecting from a target	2-21

Chapter 3

Configuring Custom Targets

3.1	About target configuration	3-2
3.2	The supplied target description (.bcd) files	3-7
3.3	Linking a board, chip, or component to a connection	3-11
3.4	Creating new target descriptions	3-20
3.5	Using RealMonitor	3-28

3.6	Example descriptions	3-38
-----	----------------------------	------

Chapter 4

Programming Flash with RealView Debugger

4.1	Introduction to Flash programming with RealView Debugger	4-2
4.2	RealView Debugger files used for Flash programming	4-7
4.3	Using the pakflash utility	4-10
4.4	Programming Flash on the ARM boards	4-11
4.5	Programming Flash for a custom development platform	4-16
4.6	Gathering information about your target	4-19
4.7	Creating algorithms for a Flash type supported by RealView Debugger ..	4-20
4.8	Creating algorithms for a Flash type not provided with RealView Debugger	4-24
4.9	Creating the Flash-level and board-level AME files	4-30
4.10	Generating the FME file	4-36
4.11	Checking the FME file with the dispflash utility	4-41
4.12	Creating a BCD file	4-42
4.13	Programming an image into Flash	4-46
4.14	Troubleshooting	4-48

Chapter 5

Configuring Custom Connections

5.1	Working with connection properties	5-2
5.2	Working with RealView ICE targets	5-7
5.3	Working with RDI targets	5-14
5.4	Working with Simulator Broker connections	5-28
5.5	Working with JTAG files	5-36

Chapter 6

Working with Remote Targets

6.1	Accessing remote targets with RealView ICE and Multi-ICE	6-2
6.2	Working with RealView Network Broker	6-3
6.3	Connecting to a remote host simulator	6-6
6.4	Using a hosts file	6-11
6.5	Disconnecting remote connections	6-17

Appendix A

Configuration Settings Reference

A.1	About this appendix	A-2
A.2	Target configuration and connections	A-3
A.3	Generic groups and settings	A-5
A.4	Target configuration reference	A-8
A.5	Custom connection reference	A-27

Glossary

Preface

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

- *About this book* on page viii
- *Feedback* on page xii.

About this book

RealView Debugger provides a powerful debugging tool for ARM® software projects. This book shows you how to configure RealView Debugger for your chosen debug target. This book also describes how target connections are managed and displayed in RealView Debugger. It contains:

- a description of the target configuration model used by RealView Debugger
- a description of target configuration using the RealView Debugger configuration facility
- a glossary of terms for users new to RealView Debugger.

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 does not assume that you are familiar with RealView Debugger.

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 and target configuration system that is used by RealView Debugger. It is recommended that you read this chapter.

Chapter 2 *Connecting to Targets*

This chapter describes how you connect to your target using the RealView Debugger connection control window. It expands on the introduction in the *RealView Debugger v1.8 Essentials Guide*, including details of the context menus and connection to targets in specific ways.

Chapter 3 *Configuring Custom Targets*

This chapter describes how you configure RealView Debugger with details of the memory and registers available on your custom target.

If you are using the ARM Integrator™ or Evaluator development platforms you do not have to configure the target because the required information is included with the product. See Chapter 2 *Connecting to Targets* for details of how to invoke this configuration.

If you are using another target, the information in this chapter enables you to set up configuration information for RealView Debugger to define the target memory map.

Chapter 4 *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.

Chapter 5 *Configuring Custom Connections*

This chapter describes how you create new connection types if the connection types configured into the product are not suitable for your target. It also describes how you install and configure *Remote Debug Interface* (RDI) target interfaces into the RealView Debugger Connection Control window.

Chapter 6 *Working with Remote Targets*

This chapter describes how RealView Debugger handles remote connections. It includes information on using RealView Network Broker, and explains how connections to different targets are configured to enable network access.

Appendix and Glossary

Appendix A *Configuration Settings Reference*

Read this appendix for details on setting options to configure your targets and connections using RealView Debugger.

Glossary

An alphabetically arranged glossary defines the special terms used in this book.

Typographical conventions

The following typographical conventions are used in this book:

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

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information.

ARM periodically provides updates and corrections to its documentation. See <http://www.arm.com> for current errata, addenda, and Frequently Asked Questions.

ARM publications

This book is part of the RealView Debugger documentation suite. Other books in this suite include:

- *RealView Debugger v1.8 Essentials Guide* (ARM DUI 0181)
- *RealView Debugger v1.8 User Guide* (ARM DUI 0153)
- *RealView Debugger v1.8 Project Management User Guide* (ARM DUI 0227)
- *RealView Debugger v1.8 Command Line Reference Guide* (ARM DUI 0175)
- *RealView Debugger v1.8 Extensions User Guide* (ARM DUI 0174).

For details on using the *RealView Compilation Tools* (RVCT), see the books in the RVCT documentation suite.

For details on using RealView ARMulator® ISS, see the following documentation:

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

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

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

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

- *RealView ICE User Guide* (ARM DUI 0155)
- *Multi-ICE® Version User Guide* (ARM DUI 0048)
- *ARM Agilent Debug Interface User Guide* (ARM DUI 0158)
- *ARM Firmware Suite Version User Guide* (ARM DUI 0136)
- *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).

Feedback

ARM Limited welcomes feedback on both RealView Debugger, and its documentation.

Feedback on RealView Debugger

If you have any problems with RealView Debugger, submit a Software Problem Report:

1. Select **Help** → **Send a Problem Report...** from the RealView Debugger main 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. Email the report to your supplier.

Feedback on this book

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

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.

Chapter 1

Introduction

This chapter introduces the connection and target configuration system used by RealView® Debugger. It contains the following sections:

- *About target connections and configuration* on page 1-2
- *Viewing board file properties* on page 1-5
- *Configuration files* on page 1-12.

1.1 About target connections and configuration

RealView Debugger works in conjunction with either a hardware or a software debug target. An ARM® development board, communicating through RealView ICE or Multi-ICE®, is an example of a hardware debug target system. *RealView ARMulator® ISS* (RVISS) is an example of a software debug target system.

RealView Debugger uses a *board file* to access information about the debugging environment and the debug targets available. The board file, therefore, describes how RealView Debugger connects to, and interacts with, your debug targets.

The default board file is set up when you first install RealView Debugger. It is called *rvdebug.brd* and, along with files that it references, is stored in your default home directory. For details on your home directory, and what it contains, see *The home directory* on page 1-14.

This section describes target connection and configuration, and introduces the features of RealView Debugger that enable you to define these settings:

- *Comparing target connection and target configuration*
- *Connection Control window* on page 1-3
- *Connection Properties window* on page 1-4.

1.1.1 Comparing target connection and target configuration

RealView Debugger makes a distinction between:

Target connection

Describes how the debugger accesses your debug target.

Target configuration

Describes your debug target to the debugger, for example how memory is mapped.

Specifically, the board file enables you to specify high-level connection details such as:

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

And low-level, target configuration details such as:

- target processor characteristics, for example processor type and endianness
- target peripheral register and memory configuration.

To do this, the default board file contains a series of elements arranged in a hierarchy. At the top level are connection entries or *target vehicles* that enable RealView Debugger to connect to different targets. As you drill down the connection, you can modify entries, or create new elements, that define the target configuration in more detail.

At the lowest level, RealView Debugger is able to access *Extended Target Visibility* (ETV) information about your debug target using a special group of settings, the Advanced_Information block that is found in all the main groups.

RealView Debugger can also access Board/Chip definition files that contain ETV information about a particular board or chip as supplied by the manufacturer, including peripheral registers and memory regions.

1.1.2 Connection Control window

You can use the Connection Control window to make your first connection to a debug target. The RealView Debugger base product includes built-in configuration files to enable you to make a connection without having to modify any configuration details.

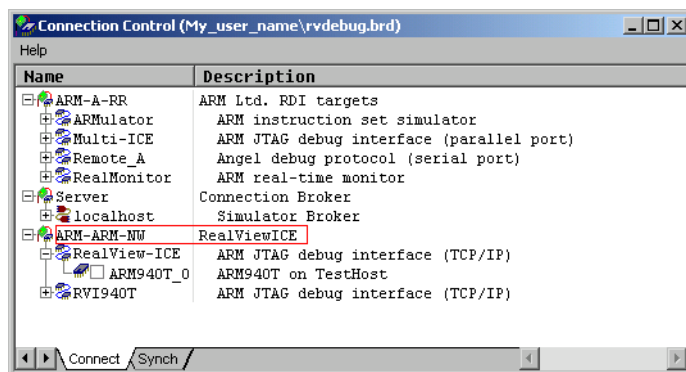


Figure 1-1 Connection Control window

The Server vehicle in the Connection Control window is also used to connect to DSP simulators, if you installed DSP support and you have the appropriate license. See the chapter that describes DSP support in *RealView Debugger v1.8 Extensions User Guide* for more information.

1.1.3 Connection Properties window

The contents of the Connection Control window are defined by elements of the board file. To change target configuration, or to add new debug targets, you change the entries in the board file and this modifies the elements displayed in the Connection Control window. RealView Debugger provides a GUI interface, the Connection Properties window, to make these changes. See *Connection Properties window* on page 1-5 for more details.

———— **Note** —————

Do not configure the board file when the debugger is connected to a target.

The rest of this chapter describes RealView Debugger target configuration in more detail.

Getting more information

Appendix A *Configuration Settings Reference* provides a detailed reference that describes all the configuration settings available in the board file. It provides examples of when to use settings and describes how to make changes. This might be useful when you are working through the examples in this chapter and in the rest of this book.

1.2 Viewing board file properties

RealView Debugger provides the Connection Properties window to enable you to examine, and change, target connections and configuration details stored in the board file. This section describes these entries in more detail and contains the following sections:

- *Connection Properties window*
- *Connection entries on page 1-8*
- *Configuration entries on page 1-9.*

1.2.1 Connection Properties window

Select **Target** → **Connection Properties...** to display the Connection Properties window, shown in Figure 1-2. This enables you to examine your current configuration settings and edit these settings to set up new connections and change target configuration.

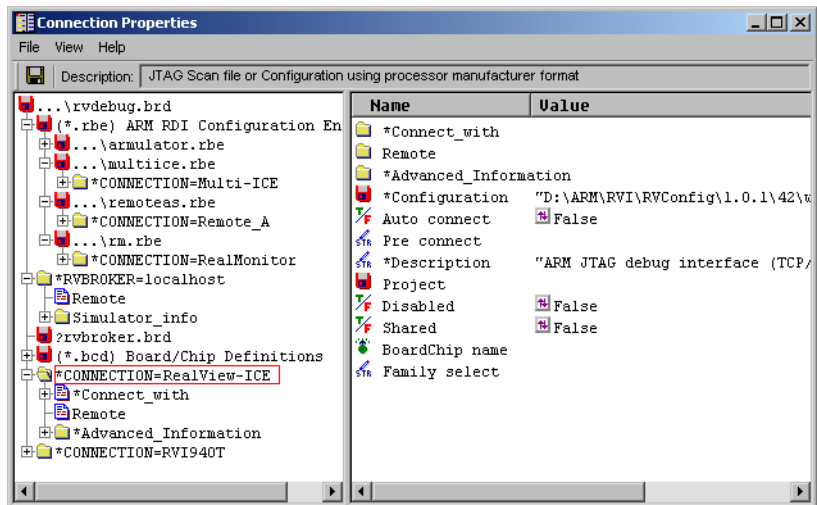


Figure 1-2 Connection Properties window

Note

Do not make changes in the Connection Properties window when the debugger is connected to a target.

Connection Properties window interface components

The main interface components of the Connection Properties window are:

Main menu This contains:

- File** Displays the **File** menu where you can save the board file after you have made changes.
- View** Displays the **View** menu to toggle the display to show all the settings or only those that have been edited.
- Help** Displays the online **Help** menu.

Toolbar The toolbar contains:

Save icon

Click this icon to save the configuration settings file to disk. The name of the current file is shown as the first entry in the left pane.

Description

This field displays a one-line description about an entry selected in the panes below.

List of Entries pane

The left pane is the List of Entries pane, and shows configuration entries as a hierarchical tree with node controls (see *List of Entries pane* on page 1-7).

Settings Values pane

The right pane is the Settings Values pane, and shows the settings for the group selected in the List of Entries pane (see *Settings Values pane* on page 1-7).

————— Note —————

See the RealView Debugger online help topic *Changing Settings* for details on all the entries in the Connection Properties window.

RealView Debugger uses the board file to configure the target and the connection with two types of entry:

- *Connection entries* on page 1-8
- *Configuration entries* on page 1-9.

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.

RealView Debugger uses this, for example, to specify an include file.



Yellow folder

This is a parent group containing other groups (*rules pages*) and/or entries.



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.

An asterisk (*) is placed at the front of an entry to show that it has changed from the default or that it was changed by RealView Debugger.

Settings Values pane

If you click on an entry in the left pane, a red 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. Groups of settings are associated with an icon to explain their function:



Red disk This specifies a disk file.

RealView Debugger uses this, for example, to specify a target device configuration file.



Yellow folder

This is a parent group or rules page containing other groups (*rules pages*) and/or entries.



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.

An asterisk (*) is placed at the front of a setting to show that it has changed from the default or that it was changed by RealView Debugger.

1.2.2 Connection entries

RealView Debugger sets up target connection entries in the board file based on targets it finds during installation, for example where you have installed RealView ICE, Multi-ICE, or ARM ADI, the appropriate DLL is autodetected by RealView Debugger. If you install a new DLL after installing RealView Debugger, for example RealMonitor, the display list is automatically updated for you.

The Connection Control window, shown in Figure 1-1 on page 1-3, uses the connection entries in the board file to create a tree of possible connections. The entries in the board file reference device configuration files (such as .jtg and .rvc) that specifies the processors that can be reached using that connection, for example ARM940T_0.

To see these connection entries you must install other ARM products because RealView Debugger does not include any connection software of its own. However, the debugger does include connection interface components, or target vehicles, to provide the interface to each target, For example, shown in Figure 1-1 on page 1-3:

- the ARM-A-RR vehicle provides the interface to ARM RDI targets
- the Server vehicle is used to connect to simulators such as RVISS
- the ARM-ARM-NW vehicle provides the interface to RealView ICE targets.

Figure 1-1 on page 1-3 shows the Connection Control window generated by the board file entries shown in Figure 1-2 on page 1-5. Each connection entry in the board file, for example CONNECTION=Multi-ICE or CONNECTION=RealView-ICE, has a corresponding entry in the Connection Control window, for example Multi-ICE and RealView-ICE.

You can create your own target connection groups, for example CONNECTION=RVI940T shown in Figure 1-2 on page 1-5. This new connection appears as an entry in the Connection Control window, that is RVI940T, shown in Figure 1-1 on page 1-3. You can create new groups based on existing connection groups or set these up using RealView

Debugger defaults. You can also add any groups that are missing when you run RealView Debugger. See Chapter 5 *Configuring Custom Connections* for details on how to do this.

Note

For more details on the entries in the Connection Control window, see *Working with the Connection Control window* on page 2-2.

1.2.3 Configuration entries

Target configuration entries enable you to describe the target architecture to RealView Debugger. This makes it possible for the debugger to present peripheral registers in a more human-readable format, and enables operations involving target memory to take account of the target memory map, for example so that Flash memory can be accessed correctly.

Target configuration information, using BOARD, CHIP, and COMPONENT groups, is used to define a hierarchy, starting from the general board-level and becoming more specific, through whole chips to component modules on a chip. However, RealView Debugger does not distinguish, functionally, between the different group names and you can use them as you require.

Using .bcd files

Within the top-level board file, `rvdebug.brd`, you can have as many BOARD, CHIP, or COMPONENT entries as you require. However, there is a better way to store them.

When RealView Debugger starts up, it searches for files with the extension `bcd` and loads them into a group called `(*.bcd) Board/Chip Definitions`, shown in Figure 1-3 on page 1-10.

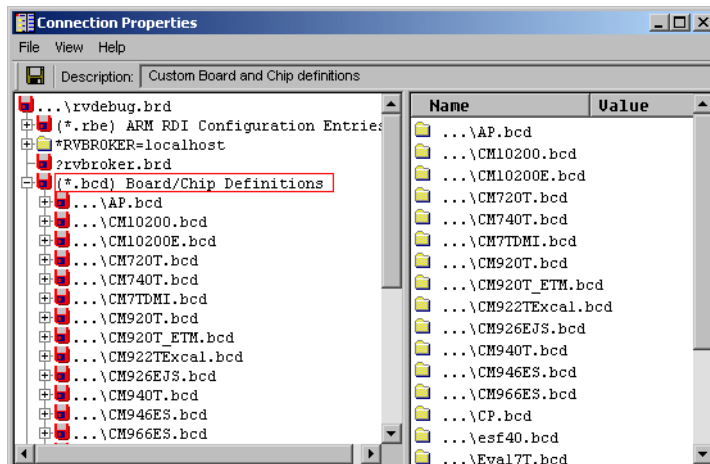


Figure 1-3 Viewing .bcd files in the Connection Properties window

Configuration entries in files loaded into this group can be referenced from any connection. For example, you can define a connection that references the file AP.bcd, to access the ARM Integrator/AP board, and the file CM940T.bcd, to access the ARM940T processor core module registers and memory map. In this way, the connection defines a hierarchy of configuration details from the board level to the processor level.

This makes the target description independent of the connection used to access it, and makes it easier to reuse target descriptions in different debugging sessions.

Using the Advanced_Information block

Extended target visibility is possible using a special group, the Advanced_Information block that is found in almost all the main groups, shown in Figure 1-2 on page 1-5. For example, there is an Advanced_Information block in the CONNECTION=RealView-ICE entry of the ARM-ARM-NW group, and in the BOARD=AP entry of the (*.bcd) Board/Chip Definitions group.

———— Note ————

In the board file, both target connection groups, for example CONNECTION=Multi-ICE, and target configuration groups, for example BOARD=AP, contain an Advanced_Information block. Although you can use the Advanced_Information block of the target connection group, it is suggested that you use target configuration groups and then reference these from the connection entry you are using. To ensure that these settings are used, do not change the default settings contained in the target connection group.

The search procedure, the way files are referenced, and the configuration options are described in more detail in Chapter 3 *Configuring Custom Targets*.

1.3 Configuration files

Default configuration files are supplied as part of the RealView Debugger base product that define standard ARM targets such as Integrator/AP. Configuration files are also created when you install additional ARM connection products, for example:

- If you install RealView ICE, configuration files are created for the standard ARM RV-msg DLLs included with this product.
- If you install Multi-ICE v1.4 or above, RealMonitor, or ADI, configuration files are created for the standard ARM RDI DLLs included with these products.
- If you are using a simulator such as RVISS, configuration files are created for these connection types.

1.3.1 What the configuration files contain

The configuration files that RealView Debugger stores in your home directory include the following files relating to this book:

- | | |
|-------|--|
| *.brd | Top-level board files. Normally this is rvdebug.brd. This file includes the filenames of the other configuration files, for example .bcd and .jtg files.

You might change this file when you save settings in the Connection Properties window (see the examples in <i>Creating new target descriptions</i> on page 3-20). |
| *.bcd | Board/Chip definition files supplied by hardware manufacturers.

Files that contain configuration information for specific targets. By default, these files are used from the default settings directory, \etc.

If you change a supplied file or you create your own, you are recommended to store them in your debugger home directory.

These files contain per-chip and per-board settings as named configurations.

You might change this file when you save settings in the Connection Properties window (see the examples in <i>Creating new target descriptions</i> on page 3-20). |
| *.cnf | Configuration files for ARM RDI target connections, for example Multi-ICE. You change one of these files when you modify the configuration of an RDI connection using the RDI configuration dialog box.

These files are referenced by the .rbe files. |

*.prc	These files are used if you install new products such as RealView ICE. For example, the first time RealView Debugger runs after you install RealView ICE, the information contained in these files is copied into the board file. After this happens, the .prc file is ignored.
*.rbe	This file references the configuration file for ARM RDI targets. This file is where the *.cnf file is named, and it also extends the RDI settings with chip and board specific configuration settings. You might change this file when you save settings in the Connection Properties window.
*.rvc	Configuration files for RV-msg connections. You change the contents of one of these files when you modify the configuration of an RV-msg connection using the RVConfig dialog box (see the example in <i>Configuring a RealView ICE interface unit</i> on page 5-10).

1.3.2 Finding configuration files

This section describes how RealView Debugger directories are created when you install the base product, and how they are used to find the configuration files. It contains the following sections:

- *The install directory*
- *The home directory* on page 1-14
- *The RealView Debugger search path* on page 1-15
- *Saving and restoring connection properties* on page 1-16.

Note

For information about the other files that are stored in the debugger home directory see *Appendix A Configuration Files Reference* in *RealView Debugger v1.8 Essentials Guide*.

For information about differences when using RealView Debugger on non-Windows platforms, see *Appendix B RealView Debugger on Sun Solaris and Red Hat Linux* in *RealView Debugger v1.8 User Guide*.

The install directory

RealView Debugger must be able to locate the product installation directory so that it can locate program extensions, data, and configuration files stored there. Use the following settings to do this:

- The default location, for example:
C:\Program Files\ARM\RVD\Core\1.8\build_number\win_32-pentium

- The RVDEBUG_INSTALL environment variable:
`set RVDEBUG_INSTALL=D:\ARM\RVD\Core\1.8\build_number\win_32-pentium`
 For information about environment variables in RealView Debugger, see the chapter that describes getting started in *RealView Debugger v1.8 User Guide*.
- The -install command line option:
`-install="D:\ARM\RVD\Core\1.8\build_number\win_32-pentium"`

If the debugger cannot find the install directory, it terminates.

The shortcuts that are installed on the Windows **Start** → **Programs** menu might include the -install option to define the directory. If you did not install the debugger in the default location and you create your own shortcuts, or run rvdebug.exe from the command line, you must either include the same -install option or define the RVDEBUG_INSTALL environment variable globally, for example using the Windows Control Panel.

The home directory

The first time you run RealView Debugger after installing on Windows, it creates a unique working directory, in your RealView Debugger home directory, for you to store your personal files, debugger settings, and target configuration files. RealView Debugger then creates or copies files into this directory ready for your first debugging session.

The location of the home directory depends on the environment variables and command-line options defined when the debugger is started. RealView Debugger uses the first defined item from the following ordered list:

1. On Windows, you can use the -home argument when starting RealView Debugger from the command line to specify an explicit path:
`-home="C:\Documents and Settings\user_name\RVD"`
 The home directory is C:\Documents and Settings\user_name\RVD
2. You can use the RVDEBUG_HOME environment variable to use a directory other than the default home directory. For example, on Windows:
`set RVDEBUG_HOME=C:\RVD\Test_home`
 For information about environment variables in RealView Debugger, see the chapter that describes getting started in *RealView Debugger v1.8 User Guide*.
3. You can use the -install command line option on its own or together with -user or \$USER:
`-install="D:\WinApp\RVD\" -user="DevTeam"`

The home directory is D:\WinApp\RVD\home\DevTeam\.

You can use -user or \$USER to specify an alternative for DevTeam.

4. You can use the RVDEBUG_INSTALL environment variable on its own or together with -user or \$USER:

```
set RVDEBUG_INSTALL=D:\WinApp\RVD\
```

```
set USER=DevTeam
```

The home directory is D:\WinApp\RVD\home\DevTeam.

5. The default location, for example:

```
install_directory\RVD\Core\...\home\user_name\
```

————— Note —————

If the debugger home directory location is defined, but the directory does not exist, the debugger creates it and copies the standard set of configuration files there from the default settings directory (\etc).

RealView Debugger on Sun Solaris and Red Hat Linux

For details on the debugger home directory for developers using RealView Debugger on Sun Solaris and Red Hat Linux see Appendix B in *RealView Debugger v1.8 User Guide*.

The RealView Debugger search path

RealView Debugger searches several directories for board 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. The location defined by the RVDEBUG_SHARE environment variable, if it is set.
For information about environment variables in RealView Debugger, see the chapter that describes getting started in *RealView Debugger v1.8 User Guide*.
3. The RealView Debugger home directory, described in *The home directory* on page 1-14, using the order specified there.
4. The default settings directory, \etc.

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

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. There are backup systems you can use:

- You can rely on the automatic backup that RealView Debugger makes whenever it saves the Connection Properties window, for example `rvdebug.brd.bak`.
- You can copy specific files, for example `rvdebug.brd`, or the whole home directory to a backup area.

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.

Using manual file or directory backups

For safer backups, you are recommended to make tape or disk copies of the files in another place. The simplest policy is to save the whole directory when you make a backup, but 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, project 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.
See *The home directory* on page 1-14 for more information about the location of your debugger home directory.
2. Determine the files to restore.
3. Copy the backup files to the debugger home directory.

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 1-12 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.
- If you have changed RV-msg connection settings, for example by changing items in the RVConfig dialog box, select the *.rvc files for the RealView ICE connections you have reconfigured. These files are saved by default in your RealView ICE installation directory.
- If you have only changed RDI connection settings, for example by changing items in the Multi-ICE configuration dialog box, select the *.cnf files for the RDI connections you have reconfigured.
- If you have changed the linkage between RDI connections and your target, select the *.rbe file for the RDI connections you have reconfigured.

Chapter 2

Connecting to Targets

This chapter describes how to use the Connection Control window to view connection details. It describes how to connect to a target using the default configuration files, and how to disconnect. It contains the following sections:

- *Working with the Connection Control window* on page 2-2
- *Managing connections* on page 2-6
- *Connecting to a target* on page 2-11
- *Failing to make a connection* on page 2-19
- *Disconnecting from a target* on page 2-21.

2.1 Working with the Connection Control window

The Connection Control window, shown in Figure 2-1, enables you to make connections, change existing connections, and configure new ones if required.

You can display the Connection Control window in the following ways:

- Click on the blue hyperlink in the File Editor pane, if available.
- Select **Target** → **Connect to Target...** from the Code window.
- Click the **Connection Control** button, in the Connection group on the Actions toolbar. If the window is hidden, click the button twice.

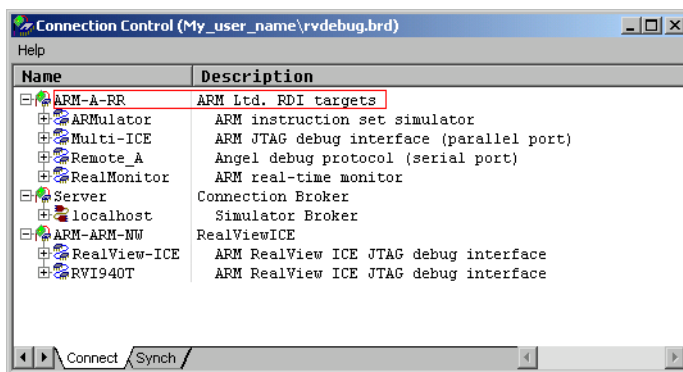


Figure 2-1 Connection Control window

Figure 2-1 shows a Connection Control window where you are licensed to work with multiprocessor debug target systems using different processor families. In this example, the window contains two tabs, **Connect** and **Synch**. In single processor debugging mode, these tabs are not visible and the window is used only to make connections.

See the chapter that describes working with multiple targets in *RealView Debugger v1.8 Extensions User Guide* for details on using the **Synch** tab to synchronize processor operations.

This section describes the Connection Control window in more detail:

- *Groups in the Connection Control window* on page 2-3
- *Using the Connection Control window* on page 2-4
- *Changing your board file* on page 2-5.

2.1.1 Groups in the Connection Control window

Figure 2-2 shows the Connection Control window for the default board file, stored in `\home\My_user_name\rvdebug.brd`, in the root installation. In this example, some groups have been expanded and a connection has been made (see *Expanding and collapsing groups* on page 2-4 for details). If you select a group, a box is drawn around it.

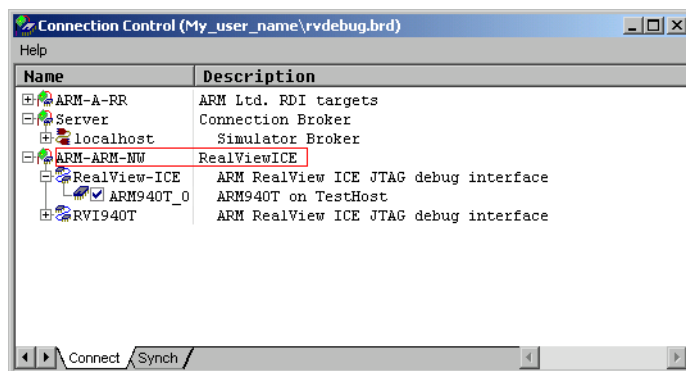


Figure 2-2 Groups in the Connection Control window

The Connection Control window is arranged in two columns or panes, Name and Description. Connection and target details are displayed in the left pane as a hierarchical tree with node controls, + and -. Connection and target details are displayed in the Name column as a hierarchy of entries:



Target vehicles

Top-level groups are supported target vehicles as specified by the target configuration settings, for example Server for targets that use the RealView® Connection Broker or ARM-ARM-NW for RealView ICE targets.

See *Working with target vehicles* on page 2-6 for more details on working with top-level groups.



Access-provider connections

Second-level groups show the type of vehicle, or the debug target interface used to support the connection, for example RealView-ICE, the ARM® JTAG debug tool for embedded systems, or Remote_A, the Angel debug monitor. The ARM RDI second-level groups are also enabled in your board file when you install ARM ADI, RealMonitor, or Multi-ICE®.

Simulator Broker connections can be local or remote:



local

Connections to simulators that are installed on the local host machine. By default, these are accessed using the localhost connection.

remote Connections to simulators that are installed on a remote host machine. Although you can modify the localhost connection to access simulators on a remote host, you might want to create a separate connection group, for example remotehost.



When you create another Simulator Broker connection group, a different icon is used to distinguish it from the default localhost connection.

See *Working with access-providers* on page 2-7 for more details on working with second-level groups.



Endpoint connections

Third-level entries show the target processors that are made available by the access provider, for example the ARM940T core connected using RealView ICE.



Each endpoint connection is accompanied by a check box to show the current state of the connection. When connected, this check box is checked, shown in Figure 2-2 on page 2-3. Where no connections have been made, the check boxes are unchecked.

RealView Debugger uses icons to help you identify the types of entries in the Name column.

2.1.2 Using the Connection Control window

The Connection Control window shows all the connections available to you as specified in your board file and the configuration files it references. The window title bar shows the location of the board file being used. In these examples, this is the default file, stored in `\home\My_user_name\rvdebug.brd` in the root installation.

Expanding and collapsing groups

Expand and collapse the groups in the Connection Control window by clicking on the plus sign or the minus sign at each node in the Name tree. Figure 2-2 on page 2-3 shows a Connection Control window after groups have been expanded and a connection has been made. If you select a group, a red box is drawn around it.

Context menus are available to change the way entries are displayed. To expand the top-level groups, right-click on a group, for example ARM-ARM-NW, and select **Expand Vehicles** from the context menu (see *Target Vehicle context menu options* on page 2-6). To expand the second-level entries, right-click on an entry, for example RealView-ICE, and select **Expand**. To collapse them again, select **Collapse** from the context menu.

Note

If you are not connected, you can also expand, or collapse, entries by double-clicking.

You can connect to a target by checking the connection state check box, shown in Figure 2-2 on page 2-3. If you collapse connected (checked) entries, RealView Debugger does not complete the collapse request so that a connection is not hidden. Instead, the control is grayed out.

The Connection Control window shows available connections as defined by the target configuration settings. When you first install RealView Debugger, this is based on information about the available target processors as defined in the default configuration files. You can connect to any of the default connections, shown in the Connection Control window, without making any more changes to configuration files.

Note

You must configure certain targets before you can make your first connection, for example RealView ICE, Multi-ICE, or RealMonitor. See Chapter 5 *Configuring Custom Connections* for more details.

Local host simulators are available immediately from the Connection Control window. If you expand the localhost Simulator Broker entry, ready to connect to a simulator, RealView Debugger starts RealView Connection Broker in local mode to manage your connection (see *Working with Simulator Broker connections* on page 5-28 for more details).

2.1.3 Changing your board file

If you work with a variety of targets and connections you might set up and save several board files, so that you can easily switch RealView Debugger from one to another. You can change the board file being used for the current session in two ways:

- right-click on a top-level entry in the Connection Control window, for example ARM-A-RR, and select the option **Select Board-File...** from the context menu (see *Target Vehicle context menu options* on page 2-6)
- change your workspace settings file to start the session with a specified board file, see the chapter that describes configuring workspaces in *RealView Debugger v1.8 User Guide* for details.

Note

To ensure a consistent target view, do not change the board file when the debugger is connected to a target.

2.2 Managing connections

Use context menus in the Connection Control window to:

- manage the displayed connections
- establish a connection to your chosen debug target
- use a different board file.

There are several context menus that can be displayed, depending on the entry you select:

- *Working with target vehicles*
- *Working with access-providers* on page 2-7
- *Working with endpoint connections* on page 2-9.

2.2.1 Working with target vehicles



Right-click on a top-level entry, for example ARM-A-RR, Server or ARM-ARM-NW, to see the **Target Vehicles** context menu.

Target Vehicle context menu options

The options available from this menu enable you to manage the vehicle used to make the connection. What options are available, and how you use these options, depends on the vehicle you are using:

Collapse All Collapses the hierarchical tree to display only the top-level entries.

Expand Vehicles

Expands the hierarchical tree to display the contents of top-level groups, and any previously expanded second-level groups.

Connection Properties...

Displays the Connection Properties window to amend current configuration details or to add target configurations (see *Viewing board file properties* on page 1-5 for details).

Add/Remove/Edit Devices...

Displays the RDI Target List window, used to add or remove RDI target DLLs from the ARM-A-RR vehicle, or to edit the configuration of one of these targets.

See *Adding and configuring RDI targets* on page 5-17 for full details on using this option.

Note

This is only appropriate for the ARM-A-RR vehicle. Do not use this menu option to add RV-msg connections.

Select Board-File...

Displays the Select Board-File to Read dialog box, where you can specify a new board file for target configuration in this session.

If you change the board file in this way, the new board is then used for all connections available with the chosen vehicle, that is local Multi-ICE connections and any remote connections that you might specify.

Note

To ensure that configuration information is maintained, do not change the active board file if:

- the Connection Properties window is open
 - you are connected to a debug target.
-

If there are any connections established, for any vehicle, the menu includes the option:

Disconnect All

Disconnects all connected targets, using the default disconnect mode, and collapses the tree to display the top-level and second-level entries.

2.2.2 Working with access-providers



Right-click on a second-level entry, for example RealView-ICE or Multi-ICE, to see the **Access-provider** context menu.

Access-provider context menu options

The options available from the **Access-provider** context menu enable you to manage the debug interface used to support the connection. What options are available, and how you use these options, depends on the interface you are using:

Expand

Expands the hierarchical tree to display the contents of the second-level group. This then changes to **Collapse**.

When you expand, RealView Debugger initializes and queries the selected interface. This might:

- cause a short delay while RealView Debugger queries the target to determine what connections are available

- result in error messages being displayed if a previously configured connection no longer operates because, for example, the JTAG cable is not connected or where a RealView ICE configuration file cannot be located
- result in the connection configuration dialog box being displayed for the interface software, for example Multi-ICE.

Connection Properties...

Displays the Connection Properties window to amend current configuration details or to add target configurations (see *Viewing board file properties* on page 1-5 for details).

Add/Remove/Edit Devices...

Displays the RDI Target List window, used to add or remove RDI target DLLs from the ARM-A-RR vehicle, or to edit target configuration files.

See *Adding and configuring RDI targets* on page 5-17 for full details on using this option.

————— **Note** —————

This is only appropriate for the ARM-A-RR vehicle. Do not use this menu option to add RV-msg connections.

Configure Device Info...

Enables you to configure the debug target, for example by displaying the RVConfig dialog box where you can configure RealView ICE debug targets.

If you want to configure Connection Broker connections, for example *RealView ARMulator® ISS* (RVISS), you must:

1. Expand the second-level localhost entry.
2. Right-click on the simulator you are using, for example new_arm.
3. Select **Configure Device Info...** from the context menu.

Test JTAG... Enables you to test that a JTAG connection exists to the devices specified for this emulator. This is useful when troubleshooting the connection setup.

This option is only available for connections that use *On-Chip Debugging* (OCD) JTAG-based connections.

Note

When installing RealView Debugger, you must choose the Custom option and install support for DSP. Do this to ensure that the required files are available to enable connection.

For details on using these menu options to configure different targets, see Chapter 5 *Configuring Custom Connections*.

2.2.3 Working with endpoint connections



Right-click on a third-level entry, for example `new_arm` or `ARM7TDMI_0`, to see the appropriate context menu:

- *Connection context menu options*
- *Disconnection context menu options* on page 2-10.

Connection context menu options

The options available from the **Connection** context menu enable you to manage the debug interface used to support the connection. What options are available, and how you use these options, depends on the interface you are using:

Connect (Defining Mode)...

Displays a List Selection dialog that lists the connect modes available for the selected target. Select the required mode and click **OK** to connect to the target. See *Connecting to a target* on page 2-11 for details on how this option is used.

Connect Connects to the selected target. See *Connecting to a target* on page 2-11 for details on how this option is used.

Connection Properties...

Displays the Connection Properties window to amend current configuration details or to add target configurations (see *Viewing board file properties* on page 1-5 for details).

Configure Device Info...

Displays the configuration dialog box for the target.

You must configure certain targets before you can make your first connection, for example RealView ICE, Multi-ICE, or RealMonitor. See Chapter 5 *Configuring Custom Connections* for more details.

Disconnection context menu options

When you select a target that is connected, the connect options on the **Connection** context menu (see *Connection context menu options* on page 2-9) change to:

Disconnect (Defining Mode)...

Displays a List Selection dialog that lists the disconnect modes available for the selected target. Select the required mode and click **OK** to disconnect from the target.

Disconnect Enables you to disconnect from the target connection.

The remaining options on the **Connection** context menu are also available depending on the target connection.

See *Disconnecting from a target* on page 2-21 for details on how these options are used.

2.3 Connecting to a target

RealView Debugger offers different ways to connect to your debug target:

- *Connections in single processor debugging mode*
- *Using the Connection Control window*
- *Setting connect mode on page 2-12*
- *Connecting to multiple targets on page 2-17*
- *Including the connection in the workspace on page 2-17*
- *Using CLI commands on page 2-17.*

Note

You must configure certain debug targets before making your first connection, for example RealView ICE or Multi-ICE. If you try to connect to a target that has not been configured, RealView Debugger displays a dialog box to indicate that the connection has failed. See *Failing to make a connection* on page 2-19 for more details.

2.3.1 Connections in single processor debugging mode

If you do not have a multiprocessor debugging license, then you can only connect to one target at a time. That is, RealView Debugger runs in single processor debugging mode. If you already have a connection established, RealView Debugger can automatically disconnect any existing connection (see *Using auto-disconnect* on page 2-22 for more details).

2.3.2 Using the Connection Control window

Select **Target** → **Connect to Target...** from the Code window main menu, or click the **Connection Control** button, to display the Connection Control window where you can connect to your debug target (see Figure 2-2 on page 2-3).

You can connect to a target in the following ways:

- Double-click on an unconnected connection entry.
- Select the check box for a required connection entry so that it is checked.
- Right-click on a connection entry and select **Connect** from the **Connection** context menu (see *Connection context menu options* on page 2-9).
- Right-click on a connection entry and select **Connect (Defining Mode)...** from the **Connection** context menu.

Note

You must use this option if you do not want the processor to be stopped when you connect to a target. For more information on defining the connection mode, see *Setting connect mode*.

For example, to connect to the default ARM7TDMI® core using RVISS:

1. Display the Connection Control window.
2. Expand the top-level Server vehicle.
3. Expand the second-level localhost entry.
4. Right-click the processor connection, for example the default new_arm, to see the **Connection** context menu (see *Connection context menu options* on page 2-9).
5. Select **Connect**.

Note

When you connect to a multiprocessor JTAG target, for example a RealView ICE target with two or more processors, the processors are listed in according to the TAP ID.

If the chosen vehicle supports a target that has not been configured, a dialog box is displayed to indicate that the connection has failed. This enables you to configure the target first and then connect. See *Failing to make a connection* on page 2-19 for more details.

RealView Debugger connects to the specified target using the default connection mode. You can, however, specify the connection mode to use, see *Setting connect mode*.

Note

Be aware that the default connection behavior for RealView ICE differs from that for Multi-ICE. You can override this to define the connection behavior (see *Setting connect mode* for details).

2.3.3 Setting 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. This mode is used when you connect from the Connection Control window in any of the following ways:

- Double-click on an unconnected connection entry.

- Select the check box for a required connection entry so that it is checked.
- Right-click on a connection entry and select **Connect** from the **Connection** context menu (see *Connection context menu options* on page 2-9).

Before connecting, RealView Debugger checks to see if a user-defined connect mode has been specified in your board file (see *User-defined connect mode* on page 2-14 for details). If such a setting is found, it overrides the default connect mode and is used instead. If there is no connect mode specified in your target configuration file, RealView Debugger proceeds to connect using the default connect mode.

However, the connect mode that is actually used depends on the target hardware, the target vehicle, and the associated interface software that manages the connection. In some cases, the interface cannot complete the connection using the mode requested, for example where your RealView ICE unit configuration conflicts with the connect mode. In this case, the RealView ICE unit determines the connect mode and makes the connection. Similarly, some ARM processors require a reset before you can connect, for example XScale™, whereas other cores have no such restriction. RealView Debugger displays a warning message to say that the requested connect mode cannot be honored and tells you what connection mode was used instead.

Note

This behavior also applies to disconnect mode, for example mode substitution might occur when you try to disconnect from a running target. See *Setting disconnect mode* on page 2-23 for more details.

RealView Debugger provides a mechanism to enable you to override both the default connect mode and any user-defined settings in your board file. This is useful when debugging multiprocessor debug target systems or multithreaded applications, but can also be used when debugging a single processor target system, for example using RealMonitor. Therefore, you can control the way a processor starts for individual connections. To do this you must select the option **Connect (Defining Mode)...** from the **Connection** context menu in the Connection Control window (see *Connection context menu options* on page 2-9). See *Connect (Defining Mode)...* on page 2-14 for details on how to use this option.

User-defined connect mode

Before connecting, RealView Debugger checks to see if a user-defined connect mode has been specified by the `Connect_mode` setting in the `Advanced_Information` block in your board file, or in any `.bcd` file linked to the connection. If such a setting is found, it becomes the default connect mode for this connection and is used if you connect in any of the following ways:

- Double-click on an unconnected connection entry.
- Select the check box for a required connection entry so that it is checked.
- Right-click on a connection entry and select **Connect** from the **Connection** context menu (see *Connection context menu options* on page 2-9).

Note

Be aware that the default connection behavior for RealView ICE differs from that for Multi-ICE. Set connect mode in your target configuration file to ensure that you connect to your chosen target in the correct way.

For more information about setting connect mode in your board file, see:

- *About target configuration* on page 3-2 in Chapter 3 *Configuring Custom Targets*
- the example *Specifying connect and disconnect mode* on page 3-46 in Chapter 3 *Configuring Custom Targets*
- the description of the `Advanced_Information` block in Appendix A *Configuration Settings Reference*.

Connect (Defining Mode)...

To define a connection mode using the **Connect (Defining Mode)...** option:

1. Right-click on a processor connection entry, for example `ARM940T_0`, using RealView ICE, to display the **Connection** context menu (see *Connection context menu options* on page 2-9).
2. Select **Connect (Defining Mode)...** to display the Connect Mode list selection box shown in Figure 2-3 on page 2-15.

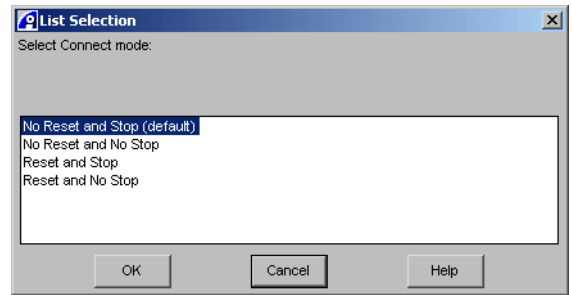


Figure 2-3 Connect Mode selection box

The state options shown depend on the interface software making the connection, but are always one or more from:

No Reset and Stop

Does not submit a processor reset but explicitly halts any process currently running by issuing a Stop command.

No Reset and No Stop

Does not submit a processor reset or attempt to halt any process currently running.

Reset and Stop

Submits a processor reset and explicitly halts any process currently running by issuing a Stop command.

Reset and No Stop

Submits a processor reset but does not attempt to halt any process currently running.

Highlight the required state and click **OK**.

Note

If you set connect mode from the Connection Control window, this temporarily overrides any user-defined setting in your target configuration file.

Default state

The selection box, shown in Figure 2-3 on page 2-15, offers a series of state options to make the connection. The options shown depend on the interface software that manages the target connection and so vary depending on the connection entry you select in the Connection Control window. In this example, the selection box contains an entry No Reset and Stop (default). This is either the:

- default connect mode chosen by the debugger (see *Setting connect mode* on page 2-12 for details)
- user-defined connect mode specified in the target configuration file (see *User-defined connect mode* on page 2-14 for details).

Any user-defined connect mode takes precedence over the default connect mode chosen by the debugger.

If you click **Cancel**, RealView Debugger establishes the connection using the connect mode shown as the default. Where this mode is not supported by your target, a different connect mode is used and you are warned about the substitution.

Note

This selection box is also displayed if you specify a prompt as your user-defined connect mode. In this case, there is no default entry (see *Using a prompt* for details).

Using a prompt

You can specify a user-defined connect mode to display a prompt to enable you to choose the connection mode for each connection. In this case, RealView Debugger displays the Connect Mode selection box shown in Figure 2-3 on page 2-15. However, where the prompt setting is defined, the state options offered are all supported by the target vehicle and so do not include a default option.

Note

If a prompt is specified in your board file, or in any .bcd file linked to the connection, 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.

For more information about using a prompt, see:

- the example *Specifying connect and disconnect mode* on page 3-46 in Chapter 3 *Configuring Custom Targets*
- the description of the Advanced_Information block in Appendix A *Configuration Settings Reference*.

2.3.4 Connecting to multiple targets

If you are licensed to work in multiprocessor debugging mode, you can make multiple target connections. In single processor debugging mode, however, you can make only one connection at a time, and making a second connection fails.

For details on working with multiprocessor connections see the multiprocessing chapter in *RealView Debugger v1.8 Extensions User Guide*.

As supplied in the base product, the default board file `rvdebug.brd` enables you to connect to one or more preconfigured debug targets on your local workstation. You can connect to these targets without making any changes to this board file.

For details on configuring different types of target, see Chapter 5 *Configuring Custom Connections*.

2.3.5 Including the connection in the workspace

If you exit the debugger with an active connection, a record of the connection details is kept in the active workspace. The next time that workspace is active when the debugger starts, the debugger attempts to set up the previous connection again. See the chapter that describes configuring workspaces in *RealView Debugger v1.8 User Guide* for more details.

2.3.6 Using CLI commands

The `CONNECT` and `RUN` commands can be used to make a connection to your debug target.

You can use the CLI `CONNECT` command to make numbered connections, for example:

```
connect,route 1
```

```
connect 6
```

where the connection id is used to identify the target.

You can also use the `CONNECT` command to specify the connect mode, for example:

```
connect,reset,halt 12
```

You can connect to remote connections in the same way.

Making named connections

You can also make named connections, for example:

```
connect @localhost
```

```
connect @new_arm
```

Specify the route name as defined in the board file, that is as it appears in the Connection Control window. Use the access-provider to avoid ambiguity, for example:

```
connect @new_arm@localhost
```

```
connect @new_arm@Remote_ARM_Debug
```

See the description of the CONNECT command in *RealView Debugger v1.8 Command Line Reference Guide* for full details on connecting to targets this way.

2.4 Failing to make a connection

When you click inside a check box, in the Connection Control window, RealView Debugger might fail to connect to the chosen debug target. This might be because:

- You do not have a valid license to use the debug target. RealView Debugger displays a message if you do not have a valid license.
- The debug target is not installed or the connection is disabled.
- The target hardware is in use by another user.
- The connection has been left open by software that exited incorrectly.
- The target has not been configured, or a configuration file cannot be located.
- The target hardware is not powered up ready for use.
- The target is on a scan chain that has been claimed for use by something else.
- The target hardware is not connected.

If RealView Debugger attempts to make a connection and fails, it normally displays a list selection box to offer possible actions, shown in Figure 2-4. Some endpoint connections, such as Multi-ICE, might also display their own dialog boxes or messages.

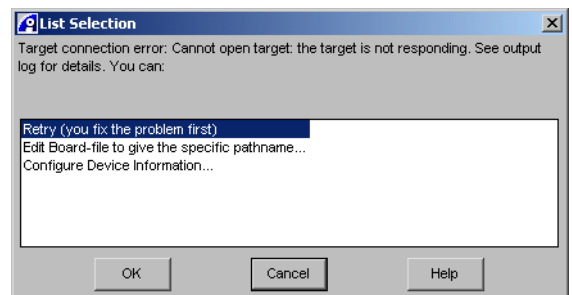


Figure 2-4 Failing to make a connection

The message displayed at the top of the selection box indicates the type of problem encountered by RealView Debugger. What message is displayed, and what options are available, depends on the interface you are using. The options are:

Retry... If you have identified the cause of the failure and corrected it, for example you have connected a board or switched on power, you can select this option and click **OK** to connect.

Edit Board file...

You can select this option and click **OK** to close the list selection box and display the Connection Properties window where you can edit your target configuration details (see *Viewing board file properties* on page 1-5 for details).

Save the new settings and then close the Connection Properties window before trying to make the connection using the Connection Control window.

Configure Device Information...

Select this option to display the configuration dialog box for the target.

You must configure certain targets before you can make your first connection, for example RealView ICE, Multi-ICE, or RealMonitor. See Chapter 5 *Configuring Custom Connections* for more details.

Click **Cancel** to close the message box and abandon the connection.

2.4.1 Troubleshooting

This section helps you to identify and fix connection problems you might encounter:

- The selection box, shown in Figure 2-4 on page 2-19, might include the entry *Display list of possible problems...* RealView Debugger might display this option if there are known problems with solutions to apply. Selecting the option displays a message box containing a list of possible causes for the failure to connect. The text describes ways to fix the problem.
This list provides only suggestions and might not be applicable to your debug target.
- If your working versions of configuration files are accidentally erased, or become corrupted, RealView Debugger might be unable to use them. See *Troubleshooting* on page 3-62 for information describing how to recover from this situation.
- If you are using RealView ICE and see a message asking you to Kill all other connections..., ensure that the required connection is available before terminating other connections. See *RealView ICE User Guide* for more details.
- If you are using Multi-ICE and see a message asking you to Reconnect to the server, you must disconnect from all processors using the Multi-ICE connection and then reconnect them. See *Multi-ICE User Guide* for more details.

2.5 Disconnecting from a target

There are several ways to disconnect when working with a target. Choosing the most appropriate method depends on:

- the number and attachment of Code windows
- which window has the focus when the disconnection option is used
- the state of the currently connected processor, and process if running
- the required state of the processor, or process, following disconnection.

This section includes:

- *Disconnect behavior in RealView Debugger*
- *Using auto-disconnect* on page 2-22
- *Using the Target menu* on page 2-22
- *Using the Connection Control window* on page 2-23
- *Setting disconnect mode* on page 2-23
- *Disconnecting by exiting* on page 2-28
- *Using the CLI* on page 2-29.

2.5.1 Disconnect behavior in RealView Debugger

Code windows are not closed on disconnecting. However, the contents might change depending on whether or not you have other connections:

- if you have other connections, and you disconnect the current connection, then the next connection in the list is set to the current connection, and the process details for that connection are displayed
- if you have other connections, and you disconnect a connection that is not the current connection, then the Code window does not change
- if there are no other connections, then all connection and process details are removed.

Additional behavior depends on the update options you set for the window and the disconnect state of the target processor:

- any images that are loaded on the disconnected target are unloaded
- the associated source files close
- entries displayed in a Register, Memory, or Process Control pane are cleared
- entries in the Watch pane remain unchanged.

For details on disconnecting during multiprocessor debugging sessions see the multiprocessing chapter in *RealView Debugger v1.8 Extensions User Guide*.

If you are working with projects, any open projects do not close if you disconnect from a debug target. Even where a project is bound to the connection, it does not close if you disconnect. However, it is unbound and if it was the current connection, its details are no longer shown in the Process Control pane.

For details on working with projects see the chapter that describes project binding in *RealView Debugger v1.8 Project Management User Guide*.

———— **Note** —————

Be aware that the default disconnection behavior for RealView ICE differs from that for Multi-ICE. You can override this to define the disconnection behavior, using the Advanced_Information block in your board file, or in any .bcd file linked to the connection. See *Setting disconnect mode* on page 2-23 for details.

2.5.2 Using auto-disconnect

Auto-disconnect only applies if you do not have a multiprocessor license. That is, when RealView Debugger runs in single processor debugging mode.

In single processor debugging mode, if you are already connected to a debug target processor, RealView Debugger automatically disconnects that connection when you make a new connection. The auto-disconnect does not occur until the new connection is successfully established.

However, if the target is running, RealView Debugger warns you and prompts for confirmation to disconnect:

- Click **Yes** to disconnect the connection with the running target.
- Click **No** to maintain the connection to the running target.

2.5.3 Using the Target menu

If you are connected to a single debug target processor, you can disconnect from the current connection, either:

- Select **Target** → **Disconnect** from the Code window main menu.
- Click the **Disconnect** button from the Connection group on the Actions toolbar.



This has the following results:

- the current connection is terminated immediately
- any windows attached to the current connection are unattached
- title bars and Color Boxes for all unattached windows are updated.

To close any unwanted windows, select **File** → **Close Window** from the main menu.

2.5.4 Using the Connection Control window

Select **Target** → **Connect to Target...** from the Code window main menu, or click the **Connection Control** button, to display the Connection Control window where you can disconnect from your debug target.

You can disconnect from a target in the following ways:

- Double-click on a connected entry.
- Select the check box for a required entry so that it is unchecked.
- Right-click on a connection entry and select **Disconnect** from the **Disconnection** context menu.
- Right-click on a connection entry and select **Disconnect (Defining Mode)...** from the **Disconnection** context menu (see *Disconnection context menu options* on page 2-10).

This option is also available from the **Target** menu.

Note

You must use this option if you do not want the processor to be stopped when you disconnect from a target. For more information on defining the disconnection mode, see *Setting disconnect mode*.

Using any of these methods immediately terminates the connection and updates the Code window display and the active connections list. This has the following results:

- the specified connection is terminated immediately
- any windows attached to the current connection are unattached
- title bars and Color Boxes for all unattached windows are updated.

RealView Debugger disconnects from the specified target using the default disconnection mode. You can, however, specify the disconnection mode to use, see *Setting disconnect mode*.

2.5.5 Setting disconnect mode

When you disconnect from a target, RealView Debugger attempts to disconnect using the default disconnect mode, that is As-is with Debug. This mode is used when you disconnect from the Connection Control window in any of the following ways:

- Double-click on a connected connection entry.
- Select the check box for a required connection entry so that it is unchecked.

- Right-click on a connection entry and select **Disconnect** from the **Disconnection** context menu (see *Disconnection context menu options* on page 2-10).

This default disconnect mode is also used when you disconnect from the Code window using either:

- Select **Target → Disconnect** from the Code window main menu.
- Click the **Disconnect** button from the Connection group on the Actions toolbar.

Before disconnecting, RealView Debugger checks to see if a user-defined disconnect mode has been specified in your board file (see *User-defined disconnect mode* on page 2-25 for details). If such a setting is found, it overrides the default disconnect mode and is used instead. If there is no disconnect mode specified in your target configuration file, RealView Debugger proceeds to disconnect using the default disconnect mode.

However, the disconnect mode that is actually used depends on the target hardware, the target vehicle, and the associated interface software that manages the connection. In some cases, the interface cannot complete the disconnection using the mode requested, for example where your RealView ICE unit configuration conflicts with the disconnect mode. In this case, the RealView ICE unit determines the disconnect mode and disconnects. RealView Debugger displays a warning message to say that the requested disconnect mode cannot be honored and tells you what disconnection mode was used instead.

Note

This behavior also applies to connect mode, for example mode substitution might occur when you try to connect to a running target. See *Setting connect mode* on page 2-12 for more details.

RealView Debugger provides a mechanism to enable you to override both the default disconnect mode and any user-defined settings in your board file. This is useful when debugging multiprocessor debug target systems or multithreaded applications, but can also be used when debugging a single processor target system, for example to download your application and leave it running without the debugger connected. Therefore, you can control the way a processor stops when you disconnect. To do this you must select the option **Disconnect (Defining Mode)...** from the **Disconnection** context menu in the Connection Control window or from the **Target** menu from the Code window main menu. See *Disconnect (Defining Mode)...* on page 2-25 for details on how to use this option.

User-defined disconnect mode

Before disconnecting, RealView Debugger checks to see if a user-defined disconnect mode has been specified by the `Disconnect_mode` setting in the `Advanced_Information` block in your board file, or in any `.bcd` file linked to the connection. If such a setting is found, it becomes the default disconnect mode for this connection and is used if you disconnect in any of the following ways:

- Double-click on a connected connection entry.
- Select the check box for a required connection entry so that it is unchecked.
- Right-click on a connection entry and select **Disconnect** from the **Disconnection** context menu (see *Disconnection context menu options* on page 2-10).
- Select **Target** → **Disconnect** from the Code window main menu.
- Click the **Disconnect** button from the Connection group on the Actions toolbar.

Note

Be aware that the default disconnection behavior for RealView ICE differs from that for Multi-ICE. Set disconnect mode in your target configuration file to ensure that you disconnect from your chosen target in the correct way.

For more information about setting disconnect mode in your board file, see:

- *About target configuration* on page 3-2 in Chapter 3 *Configuring Custom Targets*
- the example *Specifying connect and disconnect mode* on page 3-46 in Chapter 3 *Configuring Custom Targets*
- the description of the `Advanced_Information` block in Appendix A *Configuration Settings Reference*.

Disconnect (Defining Mode)...

To set the disconnect mode, either:

- select **Target** → **Disconnect (Defining Mode)...** from the Code window main menu
- select the **Disconnect (Defining Mode)...** option from the Connection Control window **Disconnection** context menu (see *Disconnection context menu options* on page 2-10).

To define a disconnect mode using the **Disconnect (Defining Mode)...** option:

1. Right-click on a connected connection entry, for example ARM940T_0, using RealView ICE, to display the **Disconnection** context menu (see *Disconnection context menu options* on page 2-10).
2. Select **Disconnect (Defining Mode)...** to display the Disconnect Mode list selection box shown in Figure 2-5.

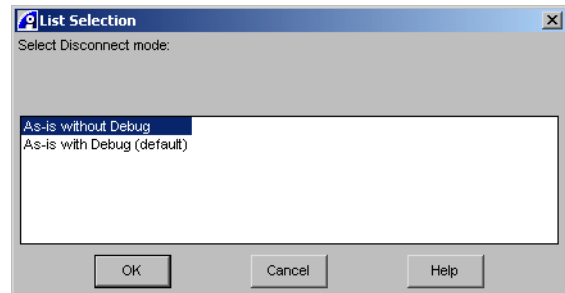


Figure 2-5 Disconnect Mode selection box

The state options shown depend on the interface software managing the connection, but are always one or more from:

As-is without Debug

Leaves the target in its current state, whether stopped or running, and removes any debugging controls such as software breakpoints.

If this leaves the processor running, any defined breakpoints are disabled. This means the program does not enter debug state after the debugger has disconnected.

As-is with Debug

Leaves the target in its current state, whether stopped or running, and maintains any debugging controls such as software breakpoints.

If this leaves the processor running, any defined breakpoints are still active. This means the program might enter debug state after the debugger has disconnected, depending on the code paths the program takes.

Highlight the required state and click **OK**. This has the following results:

- the current connection is disconnected
- the command is reflected in the **Cmd** tab of the Output pane
- the toolbar state group is set to Unknown

- any windows attached to the current connection are unattached
- title bars and Color Boxes for all unattached windows are updated.

The state options specify that the target is left in the current state, that is running or stopped. There is no option to stop the processor before disconnecting. To ensure that the processor is left in a particular state, you must stop (or start) the processor before disconnecting.

Remember that the disconnect mode that is actually used depends on the target hardware, the target vehicle, and the associated interface software that manages the connection. In some cases, the interface cannot complete the disconnection using the mode requested, for example where your RealView ICE unit configuration conflicts with the disconnect mode. In this case, the RealView ICE unit determines the disconnect mode and disconnects. RealView Debugger displays a warning message to say that the requested disconnect mode cannot be honored and tells you what mode was used instead.

Note

If you set disconnect mode from the Connection Control window, this temporarily overrides any user-defined setting in your target configuration file.

Default state

The selection box, shown in Figure 2-5 on page 2-26, offers one or more state options to disconnect from your target. The options shown depend on the interface software that manages the target connection and so vary depending on the connection entry you select in the Connection Control window. In this example, the selection box contains an entry As-is with Debug (default). This is either the:

- default disconnect mode chosen by the debugger (see *Setting disconnect mode* on page 2-23 for details)
- user-defined disconnect mode specified in the target configuration file (see *User-defined disconnect mode* on page 2-25 for details).

Any user-defined disconnect mode takes precedence over the default disconnect mode chosen by the debugger.

If you click **Cancel**, RealView Debugger disconnects using the disconnect mode shown as the default. Where this mode is not supported by your target, a different disconnect mode is used and you are warned about the substitution.

Note

This selection box is also displayed if you specify a prompt as your user-defined disconnect mode. In this case, there is no default entry (see *Using a prompt* for details).

Using a prompt

You can specify a user-defined disconnect mode to display a prompt to enable you to choose the disconnection mode for each connection. In this case, RealView Debugger displays the Disconnect Mode selection box shown in Figure 2-5 on page 2-26.

However, where the prompt setting is defined, the state options offered are all supported by the target vehicle and so do not include a default option.

Note

If a prompt is specified in your board file, or in any .bcd file linked to the connection, 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.

For more information about using a prompt, see:

- the example *Specifying connect and disconnect mode* on page 3-46 in Chapter 3 *Configuring Custom Targets*
- the description of the Advanced_Information block in Appendix A *Configuration Settings Reference*.

2.5.6 Disconnecting by exiting

By default, exiting the debugger with a connection causes details of the connection to be stored in the current workspace. See the chapter that describes configuring workspaces in *RealView Debugger v1.8 User Guide* for more details.

When you exit, the debugger prompts you to ensure you want to disconnect, as described in *Disconnection confirmation*.

When RealView Debugger starts up with a workspace that includes stored connection information, it tries to reconnect. If this fails, you are prompted for the next action, as described in *Reconnecting stored connections* on page 2-29.

Disconnection confirmation

If you exit the debugger with active connections, the debugger asks whether these connections can be disconnected, shown in Figure 2-6 on page 2-29.

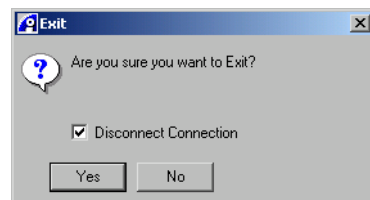


Figure 2-6 Disconnect confirmation

If you do not disconnect, the *Target Vehicle Server* (TVS) maintains the connection until another debugger session requires it. Therefore, if you load and run an image on your target, stop it, exit from the debugger without disconnecting, and then rerun the debugger, it is stopped in the same place when the debugger redisplay the connection.

If you disconnect when exiting the debugger, TVS disconnects that connection, using the disconnection mode defined in the *Advanced_Information* setting *Disconnect_mode* in the connection properties for that connection. If this leaves the TVS with no connections, it exits as well.

Reconnecting stored connections

If, when you restart the debugger, the connection stored in the workspace is no longer available, you are prompted to retry or reconfigure it, shown in Figure 2-7. Select **Configure Device Information...** from the list and click **OK** to reconfigure the connection (see Chapter 5 *Configuring Custom Connections* for more details). Click **Cancel** to abort the connection attempt.

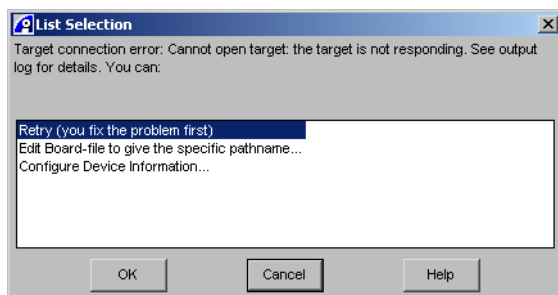


Figure 2-7 Disconnect reconfiguration or retry

2.5.7 Using the CLI

You can disconnect from a debug target using the CLI command `DISCONNECT`. This also enables you to specify the disconnection mode using a dialog box. See *RealView Debugger v1.8 Command Line Reference Guide* for more information.

Chapter 3

Configuring Custom Targets

This chapter describes in detail the debug target configuration model used by RealView® Debugger. Read this chapter to find out how to describe your debug target to the debugger. It contains the following sections:

- *About target configuration* on page 3-2
- *The supplied target description (.bcd) files* on page 3-7
- *Linking a board, chip, or component to a connection* on page 3-11
- *Creating new target descriptions* on page 3-20
- *Using RealMonitor* on page 3-28
- *Example descriptions* on page 3-38.

3.1 About target configuration

This section describes target configuration in more detail in readiness for the examples in the rest of this chapter. This section describes:

- *Target configuration settings*
- *Default configuration files*
- *How configuration files are linked together* on page 3-6
- *Board file contents* on page 3-6.

3.1.1 Target configuration settings

RealView Debugger assembles configuration settings to describe the debug environment and all the debug targets available in the current debugging session. These settings serve two main purposes. They:

- describe your debug targets in a way that enables RealView Debugger to find out all the information it requires to establish a connection
- enable you to configure the *Extended Target Visibility* (ETV) features of your debug targets, and make this information accessible to RealView Debugger.

Using internal configuration settings in this way means that you can change your debug target connection, or connect to multiple debug targets, without leaving your RealView Debugger session.

3.1.2 Default configuration files

As introduced in Chapter 1 *Introduction*, the debug target configuration settings are maintained through the use of a hierarchy of configuration files:

- *Board file* on page 3-3
- *RV-msg configuration files* on page 3-3
- *JTAG configuration files* on page 3-4
- *RDI configuration files* on page 3-4
- *Board/Chip definition files* on page 3-5.

Note

Some default configuration files are supplied as part of the RealView Debugger installation, see *Configuration files* on page 1-12 for details.

Board file

RealView Debugger uses a board file to access information about the debugging environment and the debug targets that are available. You can use RealView Debugger with the default board file that is installed for you. This is called `rvdebug.brd` and is copied into your home directory, from the default settings directory `\etc`, when you first use RealView Debugger after installation. 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.

The board file defines the debug target configuration settings for the current session. For each available target, it describes the type of target, the simulator or emulator being used, and any custom connection information.

RealView Debugger must have a board file to make connections. If you work with a variety of targets and connections, you might set up, and save, several board files so that you can easily switch the debugger from one to another. You can use the default board file as a basis for making many copies, each edited for a particular purpose.

You can use a text editor to display or print the contents of a board file, and all associated configuration files, but it is recommended that you never edit these files with a text editor or word processor. Use only the Connection Properties window to make changes to a board file, or to create a new one.

RV-msg configuration files

These files are used to define the configuration settings for RV-msg connections, such as RealView ICE. You change the contents of one of these files when you modify the configuration of an RV-msg connection using the RVConfig dialog box.

Each interface unit is defined using a `.rvc` file, for example `rvi_940T_tst.rvc`. However, you can specify different `.rvc` files to configure custom targets.

When you install RealView ICE, these XML files are created in your `RVI\RVConfig` directory ready for use. RealView Debugger searches for these files using the default search path, see *The RealView Debugger search path* on page 1-15 for details. However, you can specify a full pathname in your board file to use a different location, for example `\RVI\test_targets\rvi_920T_tst.rvc`.

————— Note —————

Do not edit these files manually. See the chapter that describes configuring a RealView ICE connection in *RealView ICE User Guide* for full details on how to use the RVConfig dialog box.

JTAG configuration files

JTAG (*Joint Test Action Group*) files are used by RealView Debugger to specify the type and order of devices on a JTAG scan chain that are available for connections.

Note

When installing RealView Debugger, you must choose the Custom option and install support for DSP. You must also have a DSP license (see *RealView Debugger v1.8 Extensions User Guide* for more details). Do this to ensure that the required JTAG files are available to enable connection.

JTAG files for supported targets, for example `arm.jtg`, are created in the default settings directory `\etc` at installation.

Whenever RealView Debugger reads a `.brd` file, it also searches for any of these related files and reads them. In this way, the information held in the JTAG files becomes part of the configuration settings for this session. You can add or remove JTAG files if necessary, without having to edit the `.brd` file. By default, new `.jtg` files are stored in `\etc`, but you can specify a different location in your `.brd` file.

RDI configuration files

When you are working with RDI targets, such as Multi-ICE and Remote_A, special configuration files are generated by the RDI configuration utilities. These files make up the RealView Debugger configuration settings specific to RDI targets.

The RDI configuration files consist of:

.rbe These files hold the default RealView Debugger connection properties for an RDI connection. Example 3-1 shows part of a `.rbe` file.

Example 3-1 .rbe example file contents

```
Description="ARM JTAG debug interface (parallel port)"
connect_with.manufacturer=ARM-A-RR
configuration="multiice.cnf"
Advanced_Information.default={\
    Memory_block.default={}:Map_rule.default={}:Register_enum.default={}: \
    Register.default={\ ...
```

There is one `.rbe` file for each RDI target available for connection.

.cnf These files define the processor to which you are connecting. Example 3-2 on page 3-5 shows part of a `.cnf` file.

Example 3-2 .cnf example file contents

```
{ TARGET_PROCESSORS
{ ARM720T_0
MULTI-ICE_CLEANDATAADDR=3758096384
MULTI-ICE_XSCACHEVECTORS[HI][RESRV ...
```

Note

You must not edit these files manually. Instead, use the RDI configuration utilities provided as part of the RealView Debugger base product, as described in Chapter 5 *Configuring Custom Connections*.

Board/Chip definition files

Board/Chip definition files contain ETV information about a particular board or chip as supplied by the manufacturer, including peripheral registers and memory regions. Board/Chip definition files are also supplied as part of support plugins to enable awareness in RealView Debugger, for example RTOS.

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 name, for example CM920T_ETM.bcd
- a board name, for example, AP.bcd
- a peripheral name or other meaningful name, for example, realmonitor.bcd.

By default, .bcd files are stored in \etc, but you can specify a different location in your board file.

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

3.1.3 How configuration files are linked together

The board file might reference several other configuration files, for example the RDI definitions, *.rbe, and the Board/Chip definition files, *.bcd., to form the complete configuration. This relationship is shown in Figure 3-1.

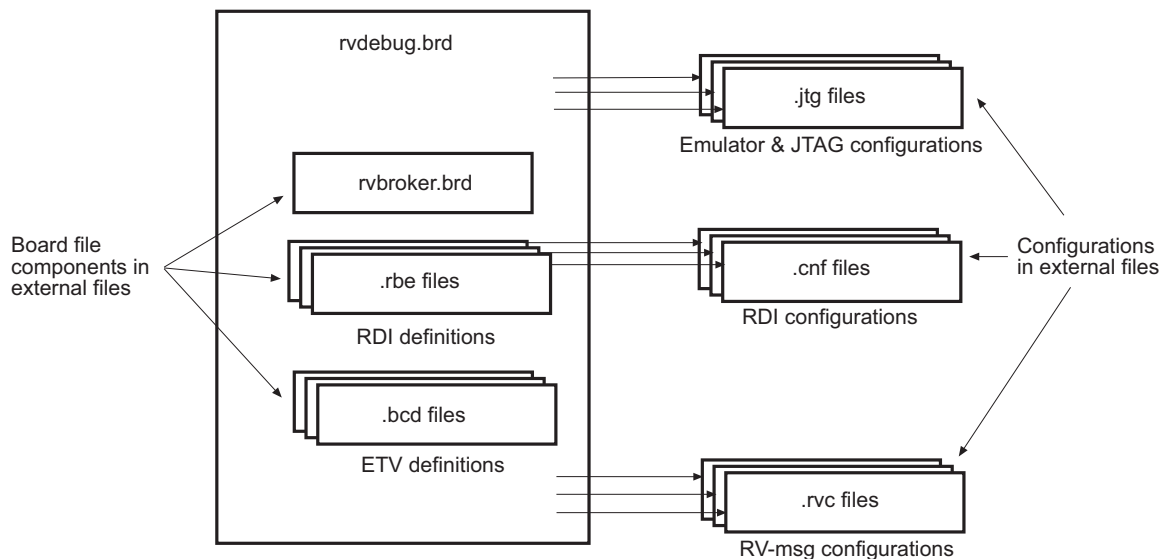


Figure 3-1 Configuration files in the board file

The RV-msg, JTAG, and RDI configuration files contain the remaining information required to configure a specific debug target. These files are not structured in the same way as the board files. They use the format required by the debug target that they are used to configure, for example, the RDI configurations are structured as Toolconf files.

3.1.4 Board file contents

Appendix A *Configuration Settings Reference* provides a detailed reference that describes all the configuration settings available in the board file. It provides examples of when to use settings and describes how to make changes. This might be useful when you are working through the examples in this chapter.

3.2 The supplied target description (.bcd) files

This section details the target board descriptions that are supplied as part of the RealView Debugger base product. These descriptions can be referenced from a specified target without additional modification. You do this by *linking* the file to the connection, see *Linking a board, chip, or component to a connection* on page 3-11 for more details.

The target descriptions are stored in files with the extension .bcd, in the default settings directory \etc. These Board/Chip definition files include details of the location and format of the registers and memory available on the described target boards.

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. See *Creating new target descriptions* on page 3-20 for details of creating your own configurations.

This section includes:

- *Summary of supplied target description files*
- *Board, chip, and component groups* on page 3-9
- *Using the supplied target descriptions* on page 3-9.

3.2.1 Summary of supplied target description files

The supplied target description files include:

AP.bcd	A description of the ARM Integrator™/AP registers and that part of the core module memory map that is decoded by the motherboard.
Eva17T.bcd	A description of the registers and memory map for the ARM Evaluator-7T, including a description of the internal registers for both KS32C50100 and S3C4510B processors.
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 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.
CP.bcd	A description of the ARM Integrator/CP registers and that part of the core module memory map that is decoded by the motherboard. This description is also suitable for use with Integrator core modules.
vpb926ej-s.bcd	A description of the registers and memory map for the Integrator VersaTile platform.

Other target description files are provided, or might be included with RealView Debugger extensions such as OS awareness downloads.

3.2.2 Board, chip, and component groups

Configuring targets within a *.bcd file, 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, for example, the Evaluator-7T (BOARD=Evaluator7T), or the Integrator/AP (BOARD=AP) motherboard, implementing memory maps and including simple peripheral components.
CHIP	Use this to define significant devices on a target board. For example, in the supplied target descriptions, the Evaluator-7T (... \Eval7T.bcd) references the ARM core-based KS32C50100 (CHIP=KS32C50100) to define the processor and ASIC components of that device. It is recommended that you use a CHIP entry where you might use the device on more than one board, or where the device is in itself complex.
COMPONENT	Other components that are not covered by a BOARD or CHIP group.

The name of each group is added to the **BoardChip** context menu (see *Linking a board* on page 3-13), or the associated List Selection dialog. You use this context menu when linking the target description to a connection (see *Linking a board, chip, or component to a connection* on page 3-11).

3.2.3 Using the supplied target descriptions

Use the supplied target descriptions by referencing them from the connection you use to communicate with your target. For example, if you are using an ARM920T processor core module with RealView ICE, modify the setting BoardChip_name in the CONNECTION=RealView-ICE group to reference the CM920T.bcd file. See *Board, chip, and component groups* for details.

If you are using an Integrator/AP or Integrator/CP motherboard with a core module, you can combine the platform and processor core module descriptions by using multiple BoardChip_name settings in the CONNECTION group, shown in the example in Figure 3-2 on page 3-10.

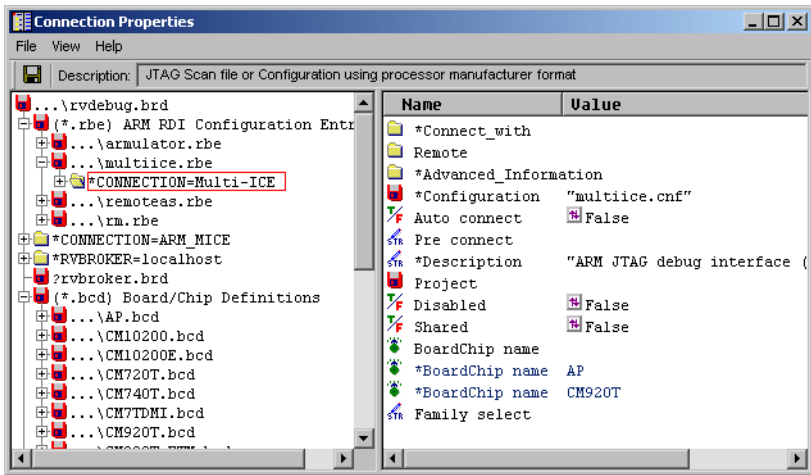


Figure 3-2 Referencing two .bcd files in the Connection Properties window

For more details on doing this, see *Linking a board, chip, or component to a connection* on page 3-11.

3.3 Linking a board, chip, or component to a connection

The target description groups in a *.bcd file are only used if:

- the .bcd file is referenced from a connection
- the identification string, `id_match`, specified in the `Advanced_Information` group, matches the hardware ID string returned by the target.

If more than one linked configuration group matches the target hardware, the group with the longest match is used.

This section describes how you make these links in the board file for a connection. These examples assume that you are linking configuration groups to a connection that uses RealView ICE. To avoid conflict between settings in different configuration groups, see *Avoiding conflicts between linked board groups*.

If this is the first time you have used RealView ICE, ensure that you configure the debug interface before trying to connect (see *Configuring a RealView ICE interface unit* on page 5-10 for details).

There are several cases to consider, presented here in order of increasing complexity:

- *Linking one board group to one connection* on page 3-12
- *Linking several board groups to one connection* on page 3-14
- *Linking one or more board groups to another board group* on page 3-16
- *Linking one or more board groups to multiple processor connections* on page 3-18.

Note

The examples in this section use existing target descriptions (see *The supplied target description (.bcd) files* on page 3-7 for examples). However, you can create and use your own target descriptions. For detailed instructions describing how to create your own target descriptions, see *Creating new target descriptions* on page 3-20.

3.3.1 Avoiding conflicts between linked board groups

When you link target description groups to a connection, make sure that you change settings in one place to avoid conflicts:

- For single processor configurations, it is suggested that you make changes at the CONNECTION level.

- For multi-processor configurations:
 - make changes to settings at the CONNECTION level if they are to apply globally to all targets, for example, enabling or disabling semihosting
 - make changes to settings at the individual target description group level if they are to apply to a specific target board or processor, for example, a memory map configuration.

3.3.2 Linking one board group to one connection

This configuration is shown in Figure 3-3.

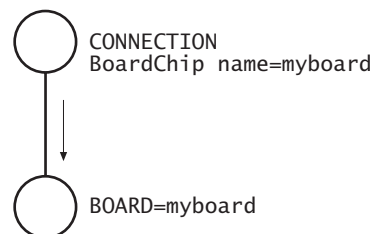


Figure 3-3 Linking one connection to one board

To set up this linking configuration you must first ensure that the *.bcd file exists and contains the required BOARD, CHIP, or COMPONENT groups. Then reference this file from your board file:

1. In the Connection Properties window, click the connection that you are using, for example CONNECTION=RealView-ICE shown in Figure 3-4.

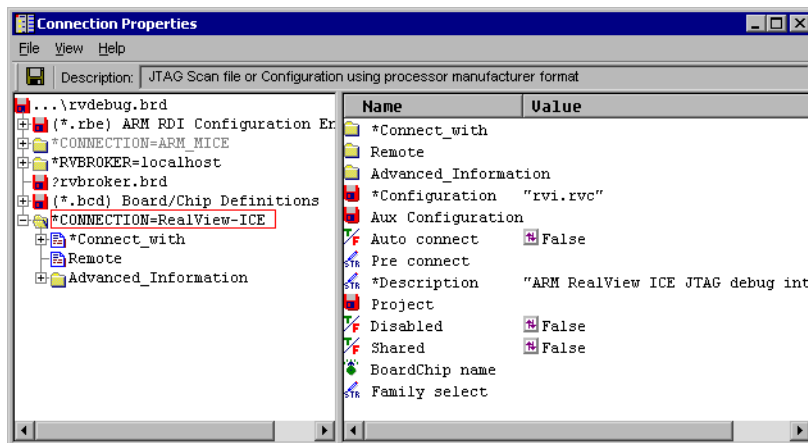


Figure 3-4 RealView-ICE connection properties

- Right-click on the BoardChip_name setting, in the right pane, to display the **BoardChip** context menu shown in Figure 3-5.

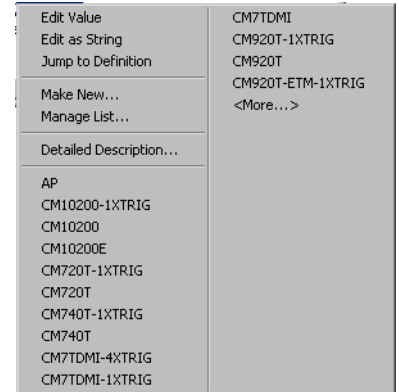


Figure 3-5 Linking a board

This context menu does not show the names of the .bcd files found by RealView Debugger. Rather, the menu includes entries derived from the contents of the .bcd files. It shows all BOARD, CHIP and COMPONENT groups in the order that RealView Debugger locates them (RealView Debugger searches your home directory first, then the ... \etc directory). This illustrates how you can share target descriptions between different connections.

- Select the name of the required group, for example **AP** for the Integrator/AP motherboard. A new entry is displayed in the right pane, which is colored blue, and has an asterisk * beside it, for example *BoardChip_name AP, shown in Figure 3-6.

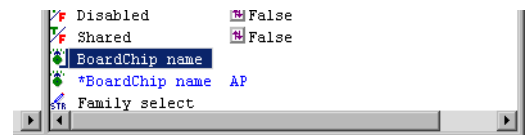


Figure 3-6 New BoardChip_name setting in connection properties

- Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

When you connect to your RealView ICE target, the configuration defined in your board group is applied to the connection. This gives you visibility of your custom hardware.

3.3.3 Linking several board groups to one connection

You might want to link several groups to a single connection if the groups represent different, possibly optional, parts of the same target. For example, the Integrator/AP motherboard definition `BOARD=AP` and an Integrator core module definition such as `BOARD=CM940T`. This kind of layout is shown in tree form in Figure 3-7.

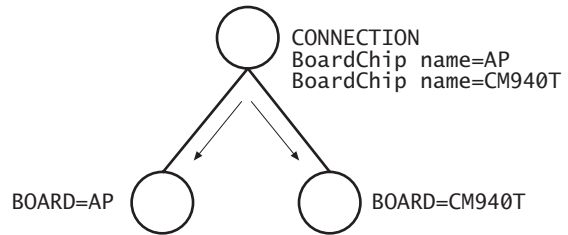


Figure 3-7 Linking one connection to two boards

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 `CONNECTION` is used, for example `BoardChip_name=AP` in Figure 3-7.

To set up this linking configuration you must first ensure that the `*.bcd` files exist and contain the required `BOARD`, `CHIP`, or `COMPONENT` groups. Then reference these files from your board file:

1. In the Connection Properties window, expand the connection that you are using, for example `CONNECTION=RealView-ICE` shown in Figure 3-4 on page 3-12.
2. Link the AP target description as described in *Linking one board group to one connection* on page 3-12, unless you have already done this.
3. Right-click on the `BoardChip_name`, in the right pane, to display the **BoardChip** context menu shown in Figure 3-5 on page 3-13.
4. Select the name of the required group, for example **CM940T**. You might have to select the **<More...>** option to find the group you require. A new entry is displayed in the right pane with an asterisk `*` beside it, shown in Figure 3-8 on page 3-15 (that is `*BoardChip_name CM940T`).

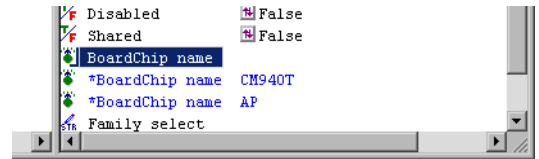


Figure 3-8 Linking a second board

5. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

When you connect to your RealView ICE target, the configuration defined in your board groups is applied to the connection. This gives you visibility of all your custom hardware.

Use the method described here to set up as many configuration groups as you require.

Changing the order of board groups

This procedure describes adding board CM940T after board AP. A board group is always added at the top of the list.

However, you might want to reorder the boards in the BoardChip_name list (to recreate the structure shown in Figure 3-7 on page 3-14). To do this:

1. Right-click on any BoardChip_name entry to display the **BoardChip** context menu and select **Manage List...**. The Settings: List Manager dialog box is displayed.
2. Select the checkbox for the AP entry, so that it is checked.
3. Click **Move Up**. The AP entry is moved above the CM940T entry.
4. Click **OK** to close the dialog box. The BoardChip_name list is reordered.
5. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

3.3.4 Linking 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. This is shown in Figure 3-9.

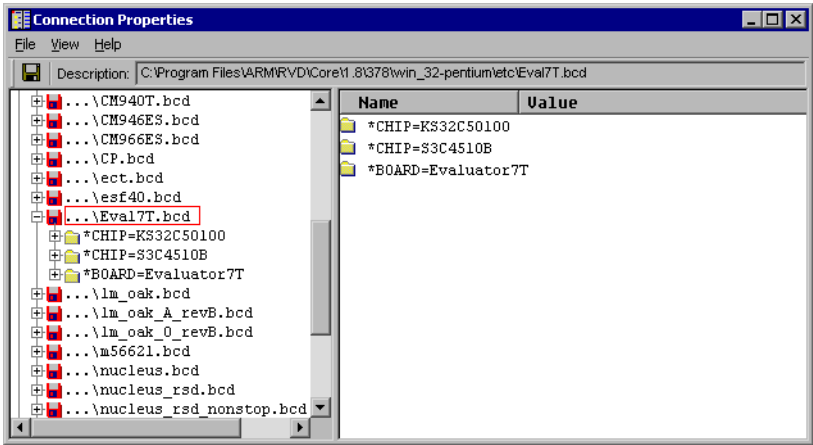


Figure 3-9 Board and chip groups for the Evaluator-7T

This is shown in tree form in Figure 3-10.

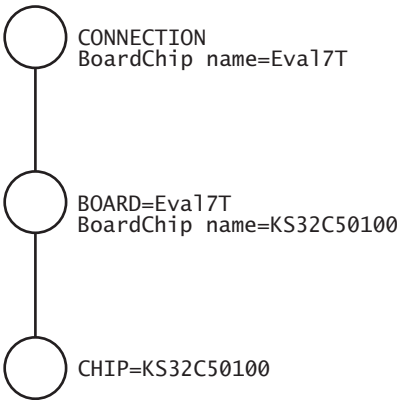


Figure 3-10 Linking one board into another board

Groups can contain BoardChip_name references to other groups, so that you can build multi-layered descriptions. For example, if you were building a simple Ethernet router, you might use the network interface on the KS32C50100 with a second network interface provided by an AMD LANCE. This configuration is shown in Figure 3-11.

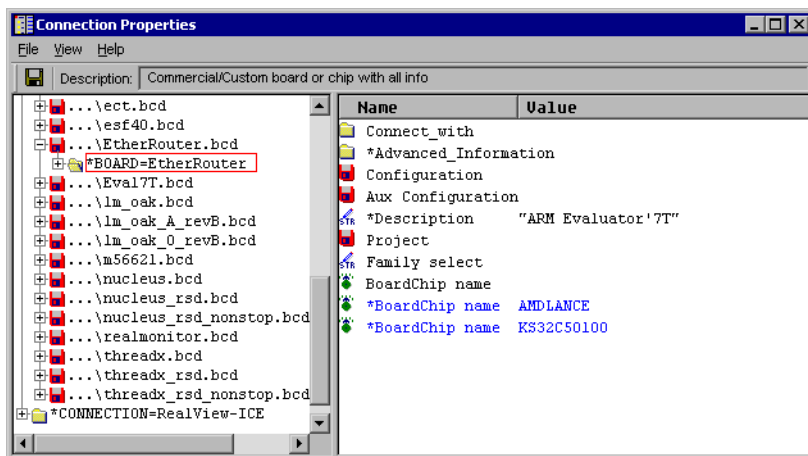


Figure 3-11 Board and chip groups for the EtherRouter

This is shown in tree from in Figure 3-12.

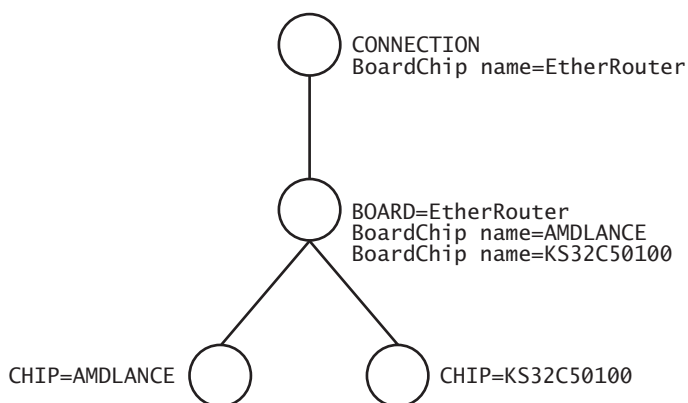


Figure 3-12 Linking one board into other boards

Note

You are not required to split your board up into distinct CHIP descriptions. You could 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.

An example showing how to create this structure is described in *Creating multiple board groups and linking to another board group* on page 3-24.

3.3.5 Linking one or more board groups to multiple processor connections

If you want to use RealView Debugger to debug a multiprocessor target, where some of the processor configurations are different, you do so by defining multiple Advanced_Information groups using names that match the processor name. These then appear in the Connection Control window.

For example, if you have a single Integrator CM920T, RealView ICE names the connection ARM920T_0. The _0 in the name indicates that this processor is on the first TAP position, that is position 0. 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 processor.

If you have two CM920T boards connected to an Integrator motherboard, RealView ICE names them ARM920T_0 and ARM920T_1. If you create two Advanced_Information groups called ARM920T_0 and ARM920T_1, shown in Figure 3-13, you can configure each board independently. Using the Default group, you can also have Advanced_Information that applies to both processors linked to the connection.

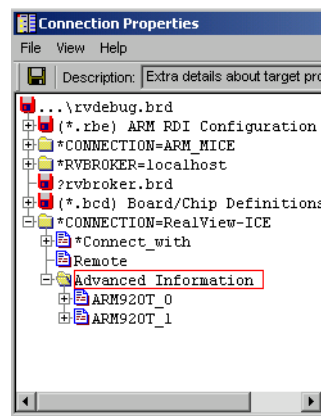


Figure 3-13 Configuring a two processor target

See Chapter 5 *Configuring Custom Connections* for more information about managing connections.

For more information about connecting RealView Debugger to multiprocessor targets, see the multiprocessing chapter in *RealView Debugger v1.8 Extensions User Guide*.

3.4 Creating new target descriptions

This section explains how to create variations of existing configurations and new target descriptions. Creating new target descriptions, rather than modifying existing entries, provides certain advantages, for example:

- if you define the addresses of I/O registers, and the bit fields within them, the debugger can display tabs in the Register pane enabling GUI access to these values
- if you define 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.

If you want to create and use a new target description, you must:

1. Create a new `.bcd` file to store the configuration. The recommended way to do this is to base the new file on an existing file stored in the default settings directory, see *Creating a new .bcd file* on page 3-21 for details.
2. Create and name a BOARD, CHIP, or COMPONENT group for the configuration. Which group you use depends on the type of hardware you are describing.
3. Define the CONNECTION entry to specify the access method used to connect to the new target.

For examples that describe how to create new CONNECTION entries, see *Example descriptions* on page 3-38.

4. Link the new target definition to the CONNECTION group that the target uses.

The rest of this section describes how to configure the new target, that is:

- *Creating a new .bcd file* on page 3-21.
- *Creating and naming a board, chip, or component* on page 3-23.
- *Linking a board, chip, or component to a connection* on page 3-11.

For information about how these different configuration groups interact, that is configuration settings, the board file, and any linked Board/Chip definition files, see *Managing configuration settings* on page 3-26.

3.4.1 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 or `.cnf` files.

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

- *Configuration files* on page 1-12 for instructions on making backups of your configuration.
- *Restoring your .brd file* on page 3-62 for instructions on restoring a default configuration.
- *Troubleshooting* on page 3-62 for instructions on recovering from an incorrectly configured debugger home directory, whether or not you have a backup.

Note

Remember, when you are following these examples, do not configure the board file when the debugger is connected to a target.

Before you start, remove the linked AP and CM940T board descriptions from the connection, if you have been following the instructions in *Linking a board, chip, or component to a connection* on page 3-11. To do this right-click on each *BoardChip_name entry, and select **Delete** from the **BoardChip** context menu. Then select **File** → **Save and Close** to save the changes and close the window.

3.4.2 Creating a new .bcd file

To create a new *.bcd file, copy one of the existing files within RealView Debugger or use the tools provided by your operating system, for example, using Windows Explorer or the cp command on Sun Solaris.

To copy a *.bcd file from RealView Debugger use the Connection Properties window:

1. Select **Target** → **Connection Properties...** to display the Connection Properties window.
2. Expand the group (*.bcd) Board/Chip Definitions by clicking on the plus sign. This displays the current list of target descriptions, in the left pane.
3. Right-click on the name of the *.bcd file to copy. For example, right-click on the entry ... \AP.bcd.
4. Select **Save As...** from the context menu, to display the dialog box shown in Figure 3-14 on page 3-22.

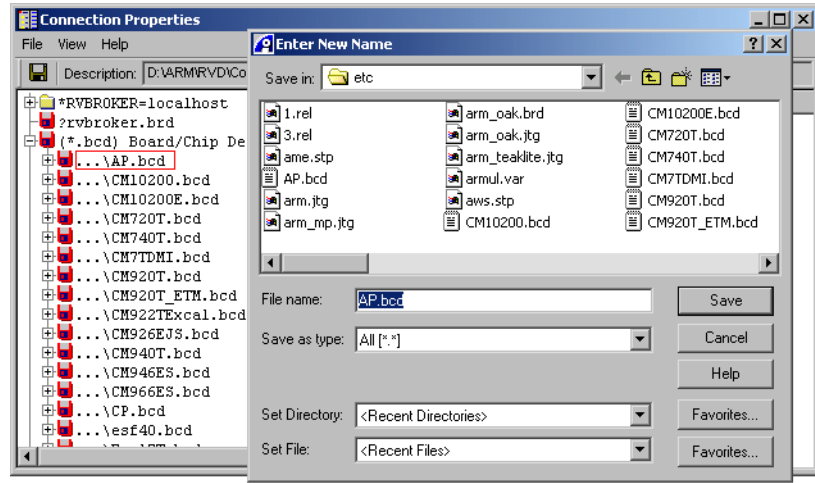


Figure 3-14 Saving an existing .bcd file with a new name

By default, *.bcd files are saved in your default settings directory \etc.

In this example, save the new file in your RealView Debugger home directory.

5. Enter the new filename, for example AM.bcd. You must use the .bcd file extension when saving the file in your home directory.
6. Click **Save**.
The dialog box closes and the new name is displayed in the *.bcd list. It replaces the initial filename.
7. Select **File** → **Save Changes**, then **File** → **Reset** to display the updated list of target description files.
8. Expand the group (*.bcd) Board/Chip Definitions to show the current list of target descriptions. The list includes the new .bcd file, that is AM.bcd.
9. Delete the contents of the new .bcd file group to avoid problems caused by old or inappropriate settings:
 - a. Expand the new .bcd file group.
 - b. Right-click on the contents, for example, *BOARD=AP, to display the context menu.
 - c. Select **Delete** from the context menu to delete the BOARD group in the file. Ensure that you delete all groups, or settings, that appear in the new .bcd file group.

10. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

If you copy the file in Windows Explorer, select **Target** → **Connection Properties...** to display the Connection Properties window and then follow this procedure from step 8 onwards.

Note

The general layout and controls of the RealView Debugger settings windows are described in the online help topic *Changing Settings*.

3.4.3 Creating and naming a board, chip, or component

To configure your target, within the *.bcd file that you created in *Creating a new .bcd file* on page 3-21, you must create a BOARD, a CHIP, or a COMPONENT group (see *Board, chip, and component groups* on page 3-9).

To create a BOARD, a CHIP, or a COMPONENT group in a .bcd file:

1. Right-click on the name of the *.bcd file to display the context menu. For example, right-click on the entry ... \AM.bcd.
2. Select **Make New Group...** from the context menu to display the Group Type/Name selector dialog box.
3. Select the type of group you want to use from BOARD, CHIP, or COMPONENT. For this example, select CHIP.
4. 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.
5. Click **OK** to create the group. This collapses the AM.bcd entry so that the new group is hidden.
6. Expand the .bcd file group to see the new CHIP=S5471KT group, shown in Figure 3-15 on page 3-24.

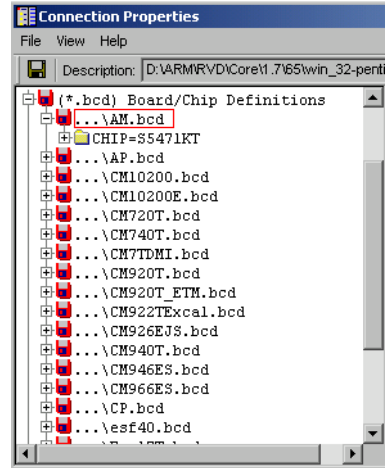


Figure 3-15 Viewing the new group in the .bcd file

7. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

Use the method described here to set up as many configuration groups as you require. It is recommended that you do this before creating, or modifying, any CONNECTION groups, as described in *Example descriptions* on page 3-38.

3.4.4 Creating multiple board groups and linking to another board group

To create the structure shown in Figure 3-12 on page 3-17 you must first ensure that the *.bcd files exist:

1. Create a new .bcd file, called ether.bcd as described in *Creating a new .bcd file* on page 3-21.
2. Create a new BOARD group in ether.bcd file, called EtherRouter, as described in *Creating and naming a board, chip, or component* on page 3-23.
3. Create a new .bcd file, called AMDLANCE.bcd as described in *Creating a new .bcd file* on page 3-21.
4. Create a new CHIP group in the AMDLANCE.bcd file, called AMDLANCE, as described in *Creating and naming a board, chip, or component* on page 3-23.

Splitting out the target descriptions in this way makes it easier to reuse configuration details on different connections. Remember to save your changes and reset the board file contents.

To reference these files from your board file:

1. In the Connection Properties window, expand the connection that you are using, for example CONNECTION=RealView-ICE.
2. Right-click on the BoardChip_name entry, in the right pane, to display the **BoardChip** context menu shown in Figure 3-5 on page 3-13.
In this example, the list includes the BOARD called EtherRouter and the CHIP called AMDLANCE.
3. Select the name of the required group, for example **EtherRouter**. A new entry is displayed in the right pane with an asterisk * beside it, shown in Figure 3-16.

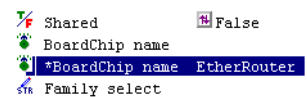


Figure 3-16 Setting up EtherRouter

4. Right-click on the *BoardChip_name EtherRouter entry, in the right pane, and select Jump to Definition from the **BoardChip** context menu.
This expands the group (*.bcd) Board/Chip Definitions to show the current list of target descriptions, which includes the new .bcd files, that is ether.bcd and AMDLANCE.bcd. The ...\ether.bcd group is expanded, and the BOARD=EtherRouter entry is selected.
5. Right-click on the BoardChip_name entry in the right-hand pane to see the **BoardChip** context menu, shown in Figure 3-5 on page 3-13.
6. Select the name of a board group, for example KS32C50100. A new entry is added to the right pane with an asterisk * beside it. For example, *BoardChip_name KS32C50100.
7. Right-click on the BoardChip_name that does not have an asterisk again to see the context menu.
8. Select the name of the other board group, for example AMDLANCE. A new entry is added to the right pane with an asterisk * beside it, shown in Figure 3-17.

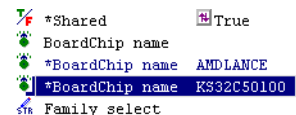


Figure 3-17 Setting up EtherRouter

9. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

When you connect to your RealView ICE target, the configuration defined in your board groups is applied to the connection. This gives you visibility of all your custom hardware.

Use the method described here to set up as many configuration groups as you require.

3.4.5 Managing configuration settings

You configure your debug target by amending board file entries using the Connection Properties window. This enables you to specify connection behavior, target visibility, image loading parameters, project loading and binding behavior, 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 a class of connections, for example CONNECTION=RealView-ICE, or on a per-board (or per-chip) basis using groups in one or more linked Board/Chip definition 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 3-11.

To ensure that settings defined in one or more linked .bcd file are used to assemble the target configuration, do not change the default settings contained in the target connection group. For example, if you specify top_of_memory in a linked .bcd file, you must check that the same entry is blank (the default) in the top-level board file:

1. Select **Target** → **Connection Properties...** to display the Connection Properties window.
2. Expand the following entries in turn:
 - a. CONNECTION=RealView-ICE (*change as required*)
 - b. Advanced_Information
 - c. Default
 - d. ARM_config
3. Ensure that Top_memory is blank.

If the setting contains an entry, right-click to display the context menu, and select **Reset to Empty** to clear the setting.

The interaction between settings in the top-level board file and any linked .bcd files might be important if you are using .bcd files to enable RTOS awareness in RealView Debugger.

———— **Note** —————

The configuration settings Connect_mode and Disconnect_mode are a special case. See *Specifying connect and disconnect mode* on page 3-46 for details on using these entries.

3.5 Using RealMonitor

This section describes how to configure RealView Debugger with RealView ICE and Multi-ICE to run RealMonitor-integrated applications. It includes:

- *Using RealMonitor with RealView ICE*
- *Using RealMonitor with Multi-ICE* on page 3-34
- *Working with RealMonitor* on page 3-37.

3.5.1 Using RealMonitor with RealView ICE

This section describes how to set up RealView Debugger with RealView ICE to run a RealMonitor-integrated application.

Note

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

Restrictions on using RealMonitor with 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 core through RealView ICE) is not possible. In particular, RealView Debugger makes no attempt to recover if the RealMonitor connection is lost during a debug session.

Rules for connecting 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 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 RealView ICE unit when it is connected to a running target
 - connect a running RealView ICE unit to a running target
 - power down the RealView ICE unit when it is connected to a running target.
- Configuring the RealView ICE unit stops the target you are connected to. In particular auto-configuring does this.

- Incorrect RealView ICE settings can result in your target being stopped or the core staying in halt-mode debug. As a result, hardware breakpoints stop the core. RealView Debugger does not warn you if this happens.
- If your target has stopped when the 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.

Procedure summary

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

1. *Making a copy of the original RealView ICE connection on page 3-30.*
2. *Configuring a RealMonitor-compatible RealView ICE connection on page 3-31.*
3. Set up RealMonitor with RealView ICE using one of the following methods:
 - If the RealMonitor image resides in Flash or ROM, you can assign the existing RealMonitor BCD file to your RealView ICE connection (see *Assigning the RealMonitor BCD file to the RealView ICE connection on page 3-32*).
 - If the RealMonitor image resides in Flash or ROM, and you want to manually configure RealMonitor you can:
 - configure the main settings for the connection
 - extend an existing BCD file to use RealMonitor, and then assign it to the connection
 - create a new BCD file to use RealMonitor, and then assign it to the connection.

See *Manually configuring a RealMonitor connection on page 3-32* for details.
 - If you have to upload your image to the target, see *Loading an image before using RealMonitor on page 3-34*.

Making a copy of the original RealView ICE connection

To make a copy of your original RealView-ICE connection:

1. Select **Target** → **Connect to Target...** to display the Connection Control window, shown in Figure 3-18.

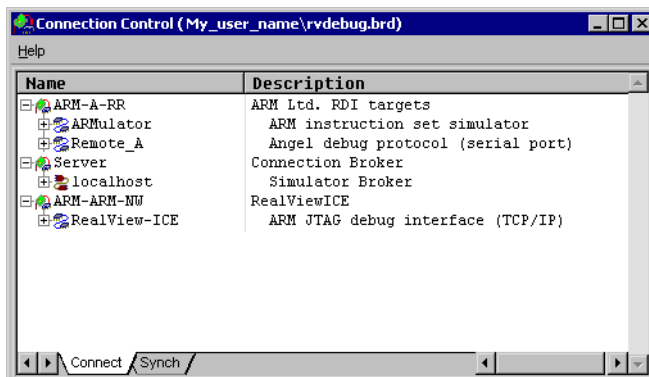


Figure 3-18 RealView ICE in the Connection Control window

2. Right-click on the RealView-ICE entry to display the Connection Properties window, shown in Figure 3-19.

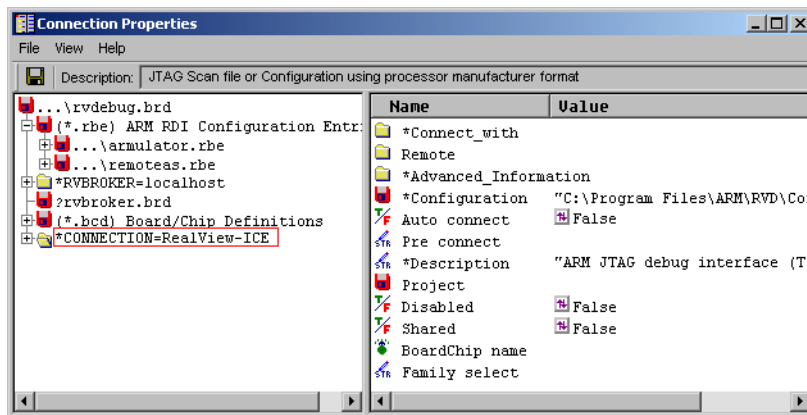


Figure 3-19 RealView ICE connection properties

3. Right-click on the CONNECTION=RealView-ICE entry in the left pane, and select **Make Copy...** from the context menu.
4. Change the name to RealView-ICE_RM.

Note

The name cannot include spaces.

5. Click **OK**.
6. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

Configuring a RealMonitor-compatible RealView ICE connection

To configure the RealView ICE connection for RealMonitor (see *RealView ICE User Guide* for more details):

1. Ensure that your target and RealView ICE unit are connected and switched on.
2. Select **Target** → **Connect to Target...** to display the Connection Control window.
3. Right-click on the RealView-ICE_RM connection you created in *Making a copy of the original RealView ICE connection* on page 3-30 and select **Connection Properties...** from the context menu.
4. Edit the value of *Configuration in the right pane and change it to rvi_rm.rvc.
5. Right-click on the *Configuration entry and select **Edit Configuration-File Contents...** from the context menu. This displays a prompt.
6. Click **Empty** to create an empty configuration. The Select Name of new file dialog box is displayed.
7. Locate your home directory and click **Save**. This saves the RealView ICE configuration file rvi_rm.rvc in your home directory, and displays the RVConfig configuration window.
8. Select your RealView ICE hardware from the list of devices.
9. Click **Connect...** to connect to RealView ICE.
10. Do one of the following:
 - If your target can be halted, click **Auto Configure Scan Chain**.
 - If your target must not be halted, then add the devices manually. See the *RealView ICE User Guide* for more details on how to do this.

11. Click **Advanced**, and change the settings as shown in Table 3-1. Leave all other settings unchanged.

Table 3-1 RealView ICE settings for RealMonitor

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

Note

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

12. Select **File** → **Save** to save the changes.
13. Select **File** → **Exit** to close the configuration window.

Assigning the RealMonitor BCD file to the RealView ICE connection

To assign the existing RealMonitor BCD file to your RealView ICE connection:

1. Select **Target** → **Connect to Target...** to display the Connection Control window.
2. Right-click on the RealView-ICE entry to display the Connection Properties window.
3. Set the BoardChip_name in the right pane to **RealMonitor**.
4. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.
5. You can now connect to the target that is running RealMonitor. No further configuration is required.

Manually configuring a RealMonitor connection

If you want to manually configure a RealMonitor connection, there are some settings for the connection that you must configure. You can configure these settings by:

- configuring the main settings for the connection

- extending an existing BCD file to use RealMonitor, and then assigning it to the connection
- creating a new BCD file to use RealMonitor, and then assigning it to the connection.

To configure the mandatory settings for a RealMonitor connection:

1. Start RealView Debugger without connecting to a target.
2. Select **Target** → **Connect to Target...** to display the Connection Control window.
3. Right-click on the RealView-ICE connection and select **Connection Properties...** from the context menu. This displays the Connection Properties window with the connection expanded.
4. Expand the following entries in turn:
 - a. CONNECTION=RealView-ICE
 - b. Advanced_Information
 - c. Default
5. Click 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**.
6. Click Monitor, and set the Type setting to **RealMonitor**.
7. Click ARM_config, and set the Vector_catch setting to **False**.
8. Expand ARM_config.
9. Click Semihosting, and set the Enabled setting to **False**.
10. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.
11. You can now connect to the target that is running RealMonitor. No further configuration is required.

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:
 - for details on how to create a standard connection, see *Connecting to a target* on page 2-11.
 - for details on how to load an image, see the chapter that describes working with images in the *RealView Debugger v1.8 User Guide*.
- Use the RealView-ICE_RM connection you created in *Making a copy of the original RealView ICE connection* on page 3-30, and that you configured as described in *Configuring a RealMonitor-compatible RealView ICE connection* on page 3-31.

3.5.2 Using RealMonitor with Multi-ICE

This section describes how to set up RealView Debugger with Multi-ICE to run a RealMonitor-integrated application.

Procedure summary

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

1. *Configuring RealMonitor to work with Multi-ICE.*
2. *Configuring the connection that is to use RealMonitor* on page 3-35.
3. *Loading an image before using RealMonitor* on page 3-36.

Configuring RealMonitor to work with Multi-ICE

Configure RealMonitor to use Multi-ICE:

1. If you are using the Multi-ICE server with RMHost, make sure that it is not autoconfigured because this causes the target to be reset and disrupts any running program. See *ARM RMHost User Guide* for more details on configuring Multi-ICE server with RMHost.
2. Select **Target** → **Connect to Target...** to display the Connection Control window, shown in Figure 3-20 on page 3-35.

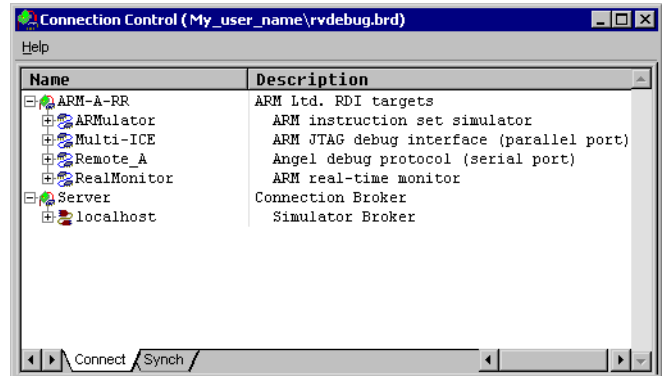


Figure 3-20 RealMonitor in the Connection Control window

Where you have installed RealMonitor, the RealMonitor.dll is autodetected by RealView Debugger and appears as a target in the Connection Control window. If it is not visible, see *Adding and configuring RDI targets* on page 5-17 for details on how to add this DLL to your list of RDI targets.

3. Expand the RealMonitor entry.
The entry expands and the relevant processor name entry is displayed.
4. Right-click on the RealMonitor entry and select **Configure Device Info...** from the context menu. Use the configuration dialog box to configure RealMonitor to use Multi-ICE. Ensure that the JTAG Controller settings points to the Multi-ICE DLL. See *Configuring ARM RealMonitor* on page 5-26 for details.
5. Click **OK** to close the RealMonitor configuration dialog box.

Configuring the connection that is to use RealMonitor

To use RealMonitor, you must modify the connection details. To do this:

1. Select **Target** → **Connect to Target...** to display the Connection Control window.
2. Right-click on the RealMonitor entry to display the Connection Properties window, shown in Figure 3-21 on page 3-36.

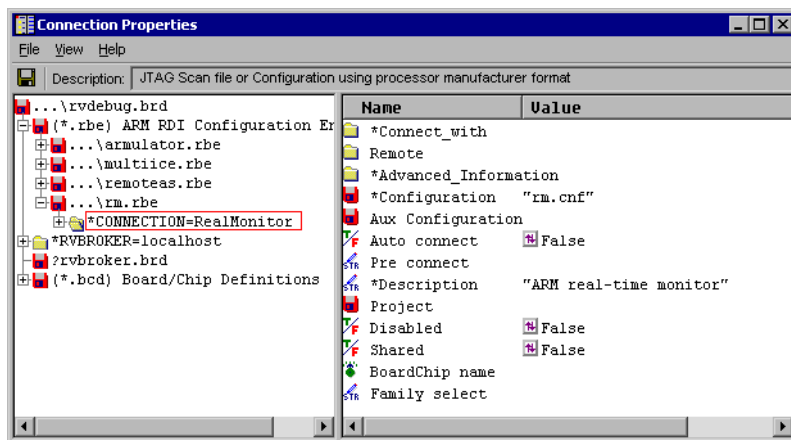


Figure 3-21 RealMonitor connection properties

3. Expand the following entries in turn:
 - a. CONNECTION=RealMonitor
 - b. Advanced_Information
 - c. Default
 - d. ARM_config
4. Click Default in the left pane, and set Connect_mode to **no_reset_and_no_stop**.
5. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

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:
 - for details on how to create a standard connection, see *Connecting to a target* on page 2-11.
 - for details on how to load an image, see the chapter that describes working with images in the *RealView Debugger v1.8 User Guide*.
- Use the RealMonitor connection you configured in *Configuring the connection that is to use RealMonitor* on page 3-35.

3.5.3 Working with RealMonitor

You are now ready to start debugging the RealMonitor image using RealView Debugger. 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 core stopping the servicing of interrupts.

For detailed information on using RealMonitor, see:

- *ARM RMTarget Integration Guide*
- *ARM RMHost User Guide*.

Loading the symbols for the image

If available, you can load the symbols of the image that is running. To do this:

1. Connect to the target with the RealMonitor connection.
2. Select **Target** → **Load Image...** to display the Load File to Target dialog box where you can locate your image.
3. Make sure that you select the **Symbols Only** check box and unselect all other check boxes.
4. Click **Open** to load the symbols.
5. To get the context, you must stop and restart the image. Specify the location of the source when requested.

Periodic update of panes

In addition, the Call-Stack, Watch, and Memory pane views have the following **Pane** menu options enabled:

- **Timed Update when Running**
- **Timed Update Period...**

These options enable the pane contents to be updated periodically with the foreground application running. See the *RealView Debugger v1.8 User Guide* for more details on how to use these panes.

3.6 Example descriptions

These examples describe how to amend board file entries, using the Connection Properties window, to configure your debug target. They assume you are familiar with the procedures described in the online help topic *Changing Settings*.

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.

In these examples, 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

- Do not configure the board file when the debugger is connected to a target.
 - See *Configuration files* on page 1-12 for instructions on making backups of your configuration before you start.
-

The examples are described in the following sections:

- *Setting up an Integrator board and core module* on page 3-39
- *Specifying connect and disconnect mode* on page 3-46
- *Configuring a memory map* on page 3-49
- *Setting up a custom register* on page 3-51
- *Setting up memory blocks* on page 3-56
- *Setting top of memory* on page 3-59
- *Flash programming* on page 3-62.

This section also includes:

- *Restoring your .brd file* on page 3-62 for instructions on restoring your factory settings
- *Troubleshooting* on page 3-62 for instructions on recovering from an incorrectly configured debugger home directory, whether or not you have a backup.

Before you start, remove the linked EtherRouter board description from the connection, if you have been following the instructions in *Creating new target descriptions* on page 3-20. To do this right-click on each *BoardChip_name entry, and select **Delete** from the **BoardChip** context menu. Then select **File** → **Save and Close** to save the changes and close the window.

3.6.1 Setting up an Integrator board and core module

This example demonstrates how to use the Connection Properties window to create a specific Integrator/AP and 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 RealView ICE with the Connection Control 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. *Configuring the new target* on page 3-40
3. *Connecting to the new target* on page 3-43
4. *Viewing the new target definition* on page 3-44.

Setting up the hardware and debug interface

The first stage is to set up the hardware and configure the RealView ICE unit:

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

Note

- RealView ICE version 1.0.1 works with RealView Debugger with some limitations. The examples assume that you are using RealView ICE version 1.2. See the release notes accompanying the software for the latest details.
- See *RealView ICE User Guide* for more details on configuring RealView ICE.

Using Multi-ICE

If you are using Multi-ICE to complete the examples, ensure that you have Multi-ICE installed, and that the Multi-ICE server is running on the workstation connected to the target. If you have not yet configured the target with Multi-ICE, do so now.

See *Multi-ICE User Guide* for more details on configuring Multi-ICE.

Configuring the new target

The next stage is to configure the new target:

1. Start RealView Debugger without connecting to a target.
2. Select **Target** → **Connect to Target...** to display the Connection Control window.
3. Right-click on the RealView-ICE entry and select **Connection Properties...** from the context menu to display the Connection Properties window.
4. Right-click on the CONNECTION=RealView-ICE entry and select **Make Copy...** from the context menu.

If the RealView-ICE target vehicle is not visible in the Connection Properties window, you must add it before continuing with this example. See the chapter that describes configuring a RealView ICE connection in *RealView ICE User Guide* for full details on how to do this.

5. Use the Group Type/Name selector dialog box, shown in Figure 3-22, to name the new connection. For this example, enter MP3Player.

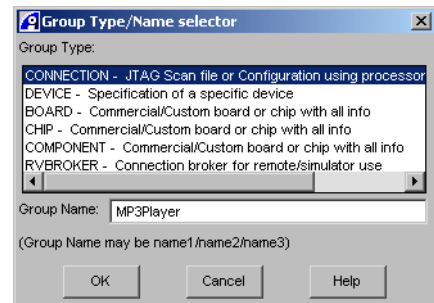


Figure 3-22 Creating a new connection

6. Click **OK**. The new entry, CONNECTION=MP3Player, is added to the List of Entries in the Connection Properties window.
7. Right-click on the Configuration setting for the new connection to display the context menu.
8. Select **Reset to Empty** to delete the current value.
9. Right-click on the Configuration setting again and select **Edit Configuration-File Contents...** the context menu. You are prompted to create a new configuration file either from an existing one, or from an empty one.

10. From the prompt dialog, select **Copy** to copy an existing configuration file. The Select file to copy from dialog box is displayed.
11. Set the Files of type field to **All [*.*)]**, and locate the existing `rvi.rvc` in the directory `install_directory\RVD\Core\...\etc\`.
12. Click **Open**. A Select Name of new file dialog box is displayed. The filename is automatically set to the connection name you specified in step 6. In this example, `MP3Player.rvi`.
13. Locate your home directory, and click **Save** to save the file. The RVConfig dialog box is displayed.
14. Click **File** → **Save** to save the configuration file.
15. Click **File** → **Exit** to exit the RVConfig dialog box. The location of the `MP3Player.rvi` file is inserted into the Configuration setting.
16. In the Connection Properties window, select **File** → **Save and Close** to save your changes. The new target is added to the Connection Control window.
17. Right-click on the new `MP3Player` connection and select **Connection Properties...** from the context menu. This displays the Connection Properties window showing the contents of the new connection, shown in Figure 3-23.

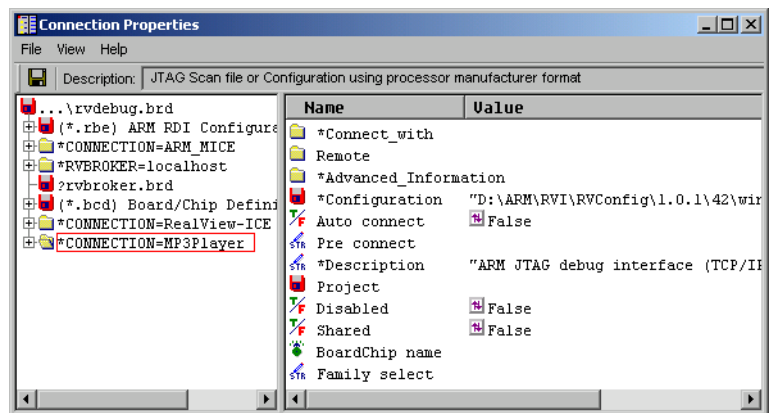


Figure 3-23 Displaying the new MP3Player connection properties

18. Right-click on Description in the right pane to display the context menu. Select **Edit Value** and 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.

19. Select **File** → **Save and Close** to save your changes. The new target description is added to the Connection Control window.

Linking board groups to the connection

The next stage is to link board groups to the new connection. This is not necessary but 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 Register pane differ from those shown here.

To link board groups:

1. Right-click on the MP3Player entry and select **Connection Properties...** from the context menu to display the Connection Properties window.
2. Right-click on BoardChip_name in the right pane to display the **BoardChip** context menu.
3. Select **AP** from the context menu to select the Integrator/AP description. A new entry *BoardChip_name AP is added to the pane.
4. Right-click on BoardChip_name (not on *BoardChip_name). The context menu is displayed again.
5. Select **<More...>** to display the List Selection dialog box.
6. Select **CM940T** from the list to select the ARM940T core module description, then click **OK**. The Connection Properties window now shows two BoardChip_name settings, shown in Figure 3-24 on page 3-43.

Name	Value
*Connect_with	
Remote	
*Advanced_Information	
*Configuration	"D:\ARM\RWI\RVConfig\1.0.1"
Auto connect	False
Pre connect	
*Description	"Integrator/AP with ARM940"
Project	
Disabled	False
Shared	False
BoardChip name	
*BoardChip name	CM940T
*BoardChip name	AP
Family select	

Figure 3-24 Board groups linked to the new connection

Target configuration settings were copied from the source connection. However, depending on the target hardware, you might have to configure other settings, for example to enable semihosting or to set stack size. See *Configuring a memory map* on page 3-49 for details. If required, you can do this now for the new connection.

7. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

Connecting to the new target

The next stage is to connect to the new target board and core module:

1. Select **Target** → **Connect to Target...** to display the Connection Control window.
2. Expand the new MP3Player entry.

The relevant processor name entry is displayed, shown in Figure 3-25 on page 3-44.

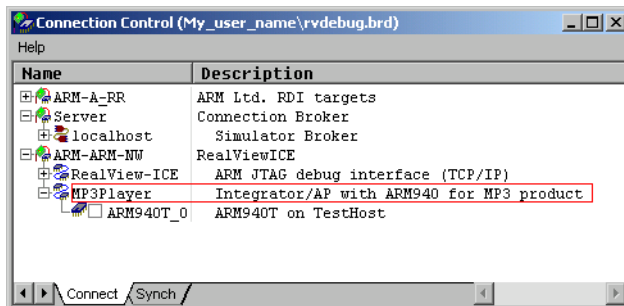


Figure 3-25 Connecting to the new target

- Click the check box for the processor entry under the new MP3Player entry to connect to the target. RealView Debugger retrieves information specific to the target.

Viewing the new target definition

To view details about the new target hardware:

- In the Code window, select **View** → **Registers** to display the Register pane. Two new tabs are included at the bottom of the pane, **AP** and **CM940T**.
- Click on the **AP** tab. RealView Debugger shows the abstraction of the hardware information specific to the Integrator/AP board, shown in Figure 3-26 on page 3-45.

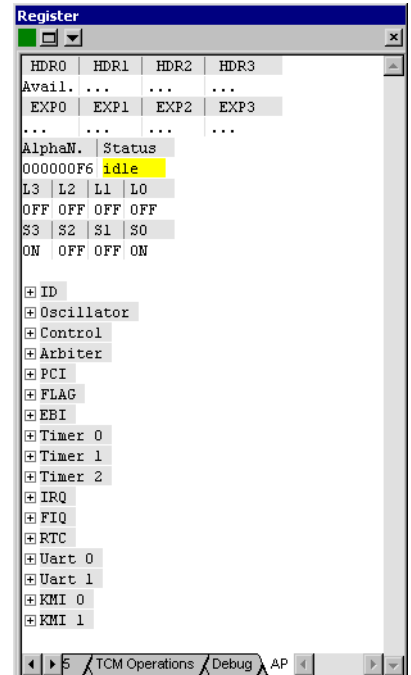


Figure 3-26 AP tab in the Register pane

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 your Integrator/AP board, right-click on the text OFF directly beneath the L2 entry in the Register pane, and select **ON** from the context menu. The relevant LED display on your Integrator/AP board is turned on.
4. Select the **CM940T** tab to see the abstraction of the hardware specific to the core module. The PRESENT status of the Motherboard indicates that the core module is connected to the Integrator/AP board.

For more details on the Register pane, see the section on working with registers in the chapter that describes debug views in *RealView Debugger v1.8 User Guide*.

5. In the Output pane at the bottom of the Code window, click on the **Cmd** tab. The display includes the line Advanced_info searched...BOARD=AP, BOARD=CM940T indicating that RealView Debugger is using the Integrator/AP and the CM940T board files.

Note

This information is only displayed when you first connect to the target after expanding the access-provider entry (MP3Player in this example).

As a result, the memory map now contains the definitions required to use the Flash memory on the Integrator. See *Flash programming* on page 3-62 for details.

3.6.2 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 in your board file. These definitions are contained in the Advanced_Information group for the target processor.

Note

This example uses the connection created in *Setting up an Integrator board and core module* on page 3-39. This example also demonstrates how to manage conflicting settings (see *Managing configuration settings* on page 3-26).

The configuration settings Connect_mode and Disconnect_mode are a special case when used to configure a debug target:

- If a prompt is specified in your board file, or in any .bcd file linked to the connection, it takes priority over any other user-defined setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.
- If a (non-prompt) user-defined setting is specified in your board file and in any .bcd file linked to the connection, the board file setting takes priority.
- A blank entry in the top-level Advanced_Information block ensures that any setting in a linked Board/Chip definition file is used instead. This might be important if you are using .bcd files to enable RTOS awareness in RealView Debugger.

The tasks described in this example illustrate both rules.

Note

The connect (or disconnect) mode that is actually used depends on the target hardware, the target vehicle, and the associated interface software that manages the connection. If you are using RealView ICE, the unit configuration determines the connect mode and makes the connection. Therefore, the unit configuration might override any settings that you specify in your board file. See *Setting connect mode* on page 2-12 for more details.

To configure connect mode for any hardware debug target:

1. Ensure that RealView Debugger is not connected to a target.
2. Select **Target** → **Connect to Target...** to display the Connection Control window.
3. Right-click on the MP3Player entry and select **Connection Properties...** from the context menu.
4. Expand the following entries in turn:
 - a. CONNECTION=MP3Player (*change as required*)
 - b. Advanced_Information
5. Select the Default entry.
6. Right-click on the Connect_mode entry in the right-hand 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 target vehicle. If you specify such an option, the debugger prompts you to select an appropriate connection mode when you try to connect. See *Setting connect mode* on page 2-12 for details on how RealView Debugger connects to a target.

7. 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.
8. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

To configure disconnect mode for any hardware debug target:

1. Ensure that RealView Debugger is not connected to a target.
2. Select **Target** → **Connect to Target...** to display the Connection Control window.
3. Right-click on the MP3Player entry and select **Connection Properties...** from the context menu.
4. Expand the following entries in turn:
 - a. (*.bcd) Board/Chip Definitions
 - b. ... \CM940T.bcd
 - c. BOARD=CM940T
 - d. Advanced_Information

5. Select the ARM940T entry.
6. Right-click on the Disconnect_mode entry in the right-hand 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 target vehicle. If you specify such an option, the debugger prompts you to select an appropriate disconnection mode when you try to disconnect. See *Setting disconnect mode* on page 2-23 for details on how RealView Debugger disconnects from a target
7. 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. See *Using a prompt* on page 2-28 for details on using a disconnect prompt in this way.

Note

This setting is stored in the Board/Chip definition file and so is used if you link this file to any other connection.

8. You can override the Disconnect_mode setting by specifying the same setting higher in the configuration hierarchy (see *Managing configuration settings* on page 3-26). To ensure that the setting in the BOARD=CM940T group is used, check for a blank entry. Expand the following entries in turn:
 - a. CONNECTION=MP3Player (*change as required*)
 - b. Advanced_Information
9. Select the Default entry.
10. Ensure that Disconnect_mode is blank.
If the setting contains an entry, right-click to display the context menu, and select **Reset to Empty** to clear the setting.

Note

If a prompt is specified in your board file, or in any .bcd file linked to the connection, 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.

11. Select **File → Save and Close** to save your changes and close the Connection Properties window.
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.
See *Disconnect (Defining Mode)*... on page 2-25 for details on using these options.

Note

This behavior also applies to connect mode, that is, a blank entry in the top-level Advanced_Information block ensures that any setting in a linked Board/Chip definition file is used instead. This might be important if you are using .bcd files to enable RTOS awareness in RealView Debugger.

3.6.3 Configuring a memory map

If you want to set up a memory map that is used automatically when you connect to a target processor, you must configure this in your board file. The memory definition is contained in the Advanced_Information group for the target processor.

Note

This example uses the connection created in *Setting up an Integrator board and core module* on page 3-39. However, in this example, the linked AP and CM940T board groups are removed to ensure a default memory mapping.

To configure a memory map for any hardware debug target:

1. Ensure that RealView Debugger is not connected to a target.
2. Select **Target** → **Connect to Target...** to display the Connection Control window.
3. Right-click on the MP3P1ayer entry and select **Connection Properties...** from the context menu.
4. Right-click on the *BoardChip_name AP setting in the right pane, and select **Delete** from the context menu.
Repeat this for the *BoardChip_name CM940T setting.
5. Expand the following entries in turn:
 - a. CONNECTION=MP3P1ayer (*change as required*)
 - b. Advanced_Information
 - c. Default
 - d. Memory_block

6. Right-click on the Default entry, under Memory_block, to display the context menu.
7. Select **Make Copy...** from the context menu to describe the memory map for the chosen target. Give this entry a suitable name, for example SSRAM, and click **Create**.
8. Click on the new SSRAM entry in the left pane to display it in the right pane.
9. Set the value of Start, in the right pane, to 0x0.
10. Set the value of Length to 0x20000.
11. Set the value of Description to Static RAM as shown in Figure 3-27.

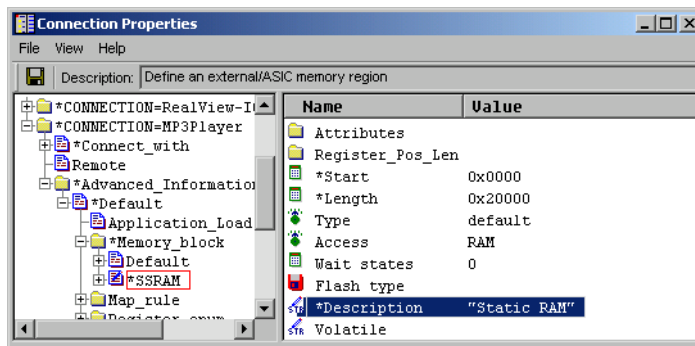


Figure 3-27 Viewing the contents of the new group

12. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.
13. Connect to your target and load an image, for example dhrystone.axf.
14. Select **View** → **Memory Map** tab to view the new memory map, shown in Figure 3-28, before loading an image.

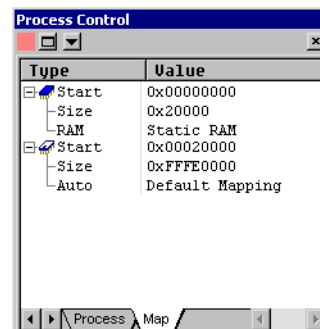


Figure 3-28 New memory map in the Process Control pane

3.6.4 Setting up a custom register

This example describes the steps to follow to specify a register MYREG, that appears as a new tab in the Register pane. It also describes how to set up named bit fields in this register. To set up the custom register, you must make changes in the Memory_block, Register, Register_enum, and Register_Window groups.

————— Note —————

This example uses the connection created in *Setting up an Integrator board and core module* on page 3-39. However, in this example, the AP and CM940T board groups are removed to ensure a default registers view.

If you have set up a memory map as described in *Configuring a memory map* on page 3-49, then delete the SSRAM entry before you begin.

In this example:

- The memory region used for the new register is called REGS and is addressed from 0x10000000-0x107FFFFF.
- The custom register, named MYREG, has an offset of 0x20 from the base of the memory block.
- The custom register, MYREG, has four bit fields. These are used as indicators of the state of the register and are named INDICATORS. They are labeled IND1, IND2, IND3, and IND4.

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

1. *Setting up the configuration*
2. *Creating enumerations for the register values* on page 3-53
3. *Creating the register descriptions* on page 3-53
4. *Creating the register tab* on page 3-55
5. *Displaying the register* on page 3-55.

Setting up the configuration

In this stage, you set up a memory group that provides the base address for the new registers:

1. Ensure that RealView Debugger is not connected to a target.
2. Select **Target** → **Connect to Target...** to display the Connection Control window.

3. Right-click on the MP3Player entry and select **Connection Properties...** from the context menu.
4. If the connection has the linked board files from *Setting up an Integrator board and core module* on page 3-39, right-click on the *BoardChip_name AP setting in the right pane, and select **Delete** from the context menu.
Repeat this for the *BoardChip_name CM940T setting.
5. Expand the following entries in turn:
 - a. CONNECTION=MP3Player (*change as required*)
 - b. Advanced_Information
 - c. Default
6. Expand the Memory_block group.
7. If the connection has the SSRAM memory block defined from *Configuring a memory map* on page 3-49, right-click on the SSRAM entry in the left pane, and select **Delete** from the context menu.
8. Rename the Default entry under Memory_block to REGS.
9. Click on the REGS entry, in the left pane, to display the group contents.
10. Set the value of Start, in the right pane, to 0x10000000.
11. Set the value of Length to 0x800000.
12. Set the value of Description to I/O Registers, (shown in Figure 3-29).

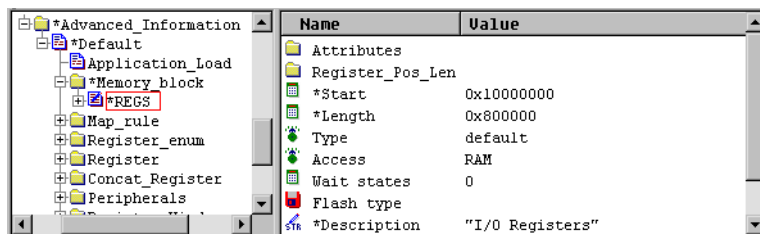


Figure 3-29 Configuring REGS

13. Select **File** → **Save Changes** to save these changes.

Creating enumerations for the register values

In this stage you set up enumerations, or names for specific values, that are used when the register value is displayed:

1. Expand Register_enum in the left pane.
2. Right-click on Register_enum and select **Make New...** to create a new group. Name this E_SWITCH.
3. Click on the E_SWITCH entry, in the left pane, to display the group contents.
4. Set the value of Names, in the right pane, to On,Off.

Note

By default, enumerations are numbered 0,1,2,... However, if you want to give your enumerations specific values, specify them as "enum1=val1,enum2=val2,...". For this example, if you want On to equal 7 and Off to equal 0 (zero), then enter "On=7,Off=0".

5. Rename the Default entry under Register_enum to E_ENABLE.
6. Click on the E_ENABLE entry, in the left pane, to display the group contents.
7. Set the value of the Names entry, in the right pane, to Disable,Enable, shown in Figure 3-30.

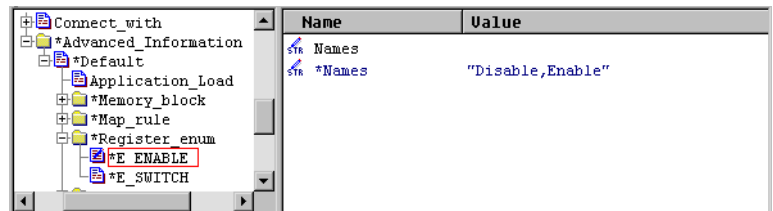


Figure 3-30 Creating enumerations

8. Select **File** → **Save Changes** to save the these changes.

Creating the register descriptions

In this stage you create descriptions of each register:

1. Expand the Register group.
2. Rename the Default entry under Register to Newreg.
3. Select the Newreg group.

4. Set the value of Base to REGS.
5. Set the value of Start to 0x20, that is the offset from the base address of the REGS memory block.
6. Expand the Newreg group.
7. Expand the Bit_fields group. You are now going to set up four bit fields.
8. Rename the Default entry under Bit_fields to IND1.
9. Use **Make Copy...** on IND1. The dialog suggests the name IND2. Click **Create**.
10. Use **Make Copy...** on IND2. The dialog suggests the name IND3. Click **Create**.
11. Use **Make Copy...** on IND3. The dialog suggests the name IND4. Click **Create**.
12. Click IND1, in the left pane and set these values, shown in Figure 3-31:
 - Position=0 (this is the default)
 - Size=1
 - Enum=E_ENABLE.

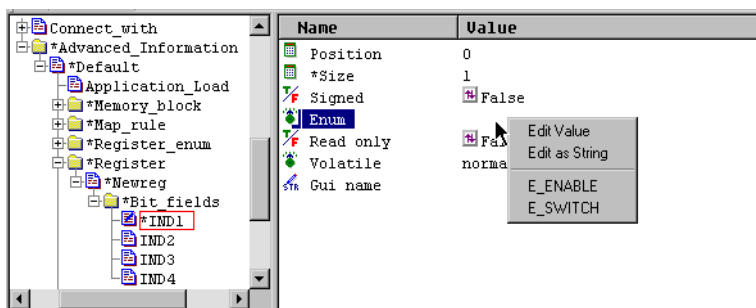


Figure 3-31 Creating bit field descriptions

13. Click on the IND2 entry and set these values:
 - Position=1
 - Size=1
 - Enum=E_SWITCH.
14. Click on the IND3 entry and set these values:
 - Position=2
 - Size=4.
15. Click on the IND4 entry and set these values:
 - Position=6
 - Size=4.

16. Select **File** → **Save Changes** to save these changes.

Creating the register tab

In this stage you create a Register_Window group to display the new register in the Register pane:

1. Expand the Register_Window group.
2. Rename the Default entry under Register_Window to MYREG. This is the name of the new tab in the Register pane.
3. Click on MYREG, in the left pane.
4. Click on MYREG, in the left pane.
5. Set the Line entry, to Newreg.
6. Use **Make New...** on *Line to create a new *Line entry.
7. Set the *Line entry, to _INDICATORS. Literals entered in *Line (or Line) must be preceded by an underscore.
8. Use **Make New...** on *Line "_INDICATORS" to create a new *Line entry.
9. Set the new *Line to IND1,IND2,IND3,IND4.

The Connection Properties window looks like Figure 3-32.

Name	Value
Line	
*Line	"Newreg"
*Line	"_INDICATORS"
*Line	"IND1,IND2,IND3,IND4"

Figure 3-32 The MYREG group

All board file entries are now complete.

10. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

Displaying the register

In the last stage, display the new register in the Register pane:

1. Connect to your target.
2. Select **View** → **Registers** to view the new tab, **MYREG**, shown in Figure 3-33 on page 3-56.

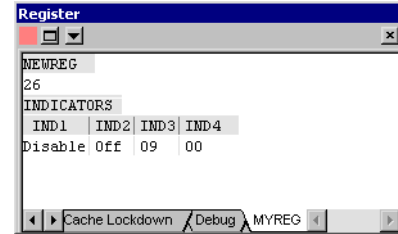


Figure 3-33 MYREG in the Register pane

3.6.5 Setting up memory blocks

This example uses the connection created in *Setting up an Integrator board and core module* on page 3-39. However, the AP and CM940T board groups are removed to ensure a default memory mapping.

This example assumes that you have worked through *Setting up a custom register* on page 3-51, because it uses some of the configuration details set up in that example and saved in the board file.

This example describes how to set up two memory blocks that are activated at different times according to the value of a register. It uses the Newreg register created in *Setting up a custom register* on page 3-51. This is displayed in the **MYREG** tab in the Register pane.

This example also describes setting a memory rule to specify how the memory is used. When IND1 is set to Enabled, MEM2 is activated. Otherwise, MEM1 is used. The example is split into these sections, which must be executed in this sequence:

1. *Defining the memory blocks*
2. *Defining the memory rules* on page 3-58.

Defining the memory blocks

The first stage is to define the two Memory_blocks named MEM1 and MEM2:

1. Select **Target** → **Connect to Target...** to display the Connection Control window.
2. Ensure that RealView Debugger is not connected to a target.
3. Right-click on the MP3Player entry and select **Connection Properties...** from the context menu.
4. Expand the following entries in turn:
 - a. CONNECTION=MP3Player (*change as required*)

- b. Advanced_Information
 - c. Default
 - d. Memory_Block
5. Right-click on the REGS entry, in the left pane, and select **Make New...** from the context menu to see the prompt shown in Figure 3-34.

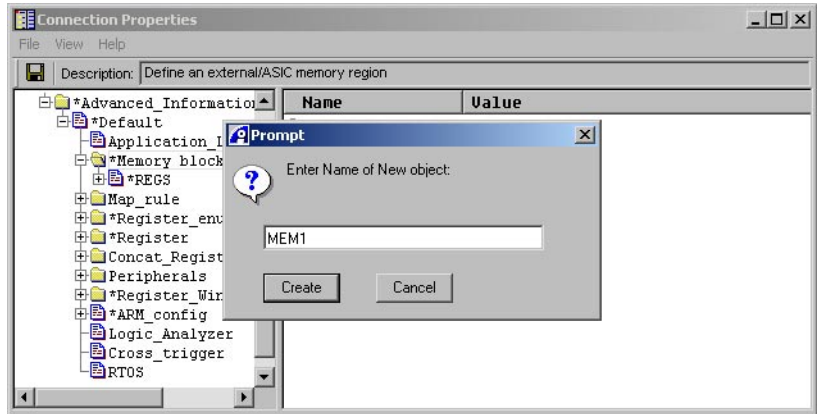


Figure 3-34 Creating a new memory block

6. Enter a new name for this entry, for example MEM1, and click **Create**.
7. Click on the MEM1 entry, in the left pane.
8. Set the value of Start to 0x0 (this is the default).
9. Set the value of Length to 0x80000.
10. Set the value of Description to Fast Static RAM.
11. Set the value of Access entry to RAM (this is the default).
12. Use **Make Copy...** on MEM1 to create a new group, MEM2.
13. Click on the MEM2 entry, in the left pane, to display the settings values.
14. Set the value of Access to ROM.
15. Set the value of *Description to Slow Boot ROM.
16. Select **File** → **Save Changes** to save the these changes.

Defining the memory rules

The second stage is to define the rules that control which memory block is used:

1. Expand the Map_rule group.
The map rule defines which memory block to use. In this example, MEM2 is activated if IND1 is set to Enabled (one). Otherwise MEM1 is used.
2. Click on the Default entry.
3. Set the following values, shown in Figure 3-35:
 - Register to Newreg (use the context menu).
 - Mask to 0x0001 (to check IND1 value only)
 - Value to 0 (IND1 set to Disabled)
 - On_equal to MEM1.

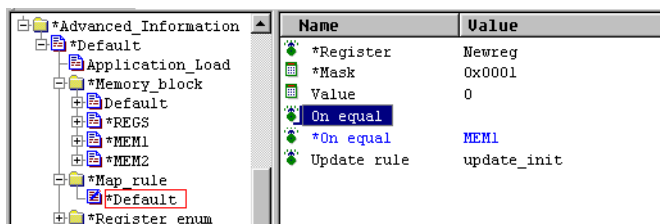


Figure 3-35 Creating a map rule

4. Rename the Default entry of Map_rule to RULE1.
5. Use **Make Copy...** on RULE1, to create a new group, RULE2.
6. Click on RULE2 and set the value of Value to 1 (IND1 set to Enabled).
7. Set the value of *On_equal to MEM2, shown in Figure 3-36.

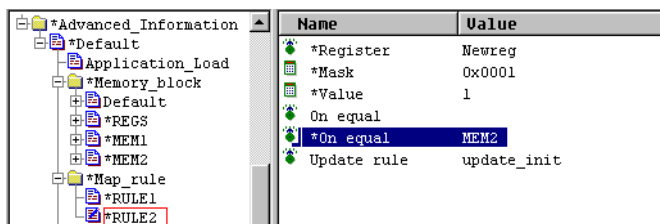


Figure 3-36 Settings for the second map rule

All board file entries are now complete.

8. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.
9. Connect to your target.
10. In the Code window, select **View** → **Registers** to view the **MYREG** tab.
11. Select **View** → **Memory Map tab** to view the **Map** tab, shown in Figure 3-37.

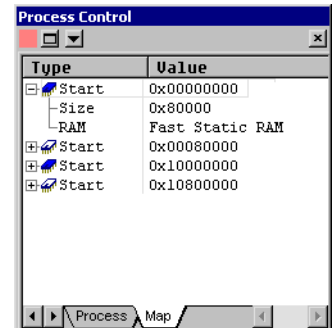


Figure 3-37 New memory block in the Map tab

12. Set the IND1 value to Enable to activate the memory rule (RULE2) and so change the memory map. The Start entry in the **Map** tab changes to a ROM entry, indicated by a yellow icon.

3.6.6 Setting top of memory

This example demonstrates how you can permanently set `top_of_memory` values for a given target using your board file. It shows how to do this by updating the information in the `ARM_config` group of the `Advanced_Information` block for the connection. After you have defined the setting, it is used whenever you connect to the target with RealView Debugger.

See the chapter that describes embedded software development in the *RealView Compilation Tools Developer Guide* for more details on using top of memory and stack heap values.

About top of memory, stack and heap

The `Top_memory` variable is used to enable the semihosting mechanism to determine the stack base address. RealView Debugger uses target-dependent defaults for the stack and heap limits. If your application is scatterloaded, you must define the stack and heap limits in `__user_initial_stack_heap` (see the *RealView Compilation Tools Developer Guide* for details).

The default value of `Top_memory` for ARM processors is `0x80000`. For details on how this variable is relevant to ARM targets, see the internal variable descriptions sections in the user guide that accompanies your interface, for example *RealView ICE User Guide*.

The stack and heap are immediately below the top of memory. For SWIs, RealView Debugger uses default values for the stack (`0x2000`) and heap (`0x4000`). Otherwise, it gets the values from the image. The relationship between top of memory, stack and heap is shown in Figure 3-38.

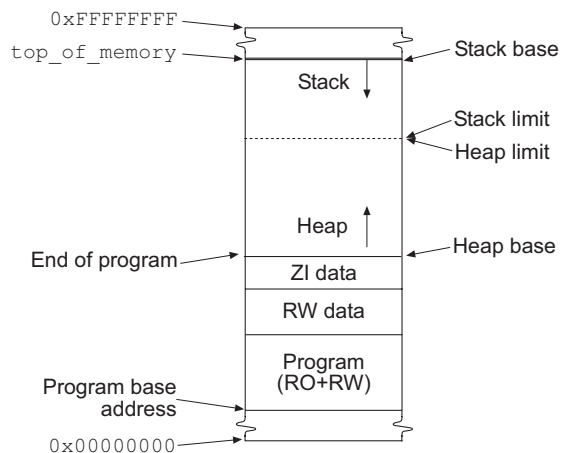


Figure 3-38 Relating `top_of_memory` to single section program layout

Note

The value of `Top_memory` 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 higher addresses.

Setting top of memory

You can set the value of `Top_memory` in a `BOARD` group used to configure the target or in the `CONNECTION` entry you use to connect to the target. The method described here applies to both. This example uses an Integrator/AP board with an ARM940T core module, but the procedure for amending the settings is the same for any target.

Note

This example uses the connection created in *Setting up an Integrator board and core module* on page 3-39. However, the AP, AM, and CM940T board groups are not linked to ensure a default memory mapping. If you do link the board groups, the contents of the Register pane differ from those shown here because the value of `top_of_memory` is defined in the linked CM940T board group. This value overrides any value that you define in the CONNECTION entry.

To set the `top_of_memory` in the CONNECTION:

1. Ensure that RealView Debugger is not connected to a target.
2. Select **Target** → **Connect to Target...** to display the Connection Control window.
3. Right-click on the MP3Player entry and select **Connection Properties...** from the context menu.
4. Expand the following entries in turn:
 - a. CONNECTION=MP3Player (*change as required*)
 - b. Advanced_Information
 - c. Default
5. Click ARM_config in the left pane.
6. 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 debug target.

When you load a program, the debugger sanity-checks Top_memory by checking that the words below Top_memory are writable. It issues a warning if they are not. However, your program might require much more RAM than the debugger checks for.

7. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

Whenever you load a program compiled with the standard ARM C library to this target, the `top_of_memory` value you have set is used.

3.6.7 Flash programming

Programming the Flash on the Integrator/AP involves:

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

Chapter 4 *Programming Flash with RealView Debugger* gives detailed information and instructions on how to 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.

For an example of how to program Flash on the Integrator/AP, see *Programming Flash on the ARM boards* on page 4-11.

3.6.8 Restoring your .brd file

If you have completed these examples and you want to return to the factory settings:

1. Exit RealView Debugger.
2. Delete your RealView Debugger home directory.

When you restart RealView Debugger it creates a new default configuration for you.

3.6.9 Troubleshooting

If your working versions of configuration files are accidentally erased, or become corrupted, RealView Debugger might be unable to use them. 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 1-16.
- If it is acceptable to lose all of the configuration settings, program preferences, workspaces and other information that is stored in the debugger home directory, you can delete it:
 1. Exit RealView Debugger.
 2. Locate the home directory the debugger is using.
See *The home directory* on page 1-14 for more details.
 3. Use Windows Explorer to rename or delete the home directory.

You might want to move or rename it before deleting so that if you make a mistake you can recover selected files.

4. Restart RealView Debugger. It creates a new, default copy of the debugger home directory as it starts up.
- If there are configuration items that you want to try to keep:
 1. Exit RealView Debugger.
 2. Using Windows Explorer, display the home directory the debugger is using. See *The home directory* on page 1-14 for more details.
 3. Using a second Windows Explorer window, locate the RealView Debugger installation directory. See *The install directory* on page 1-13 for more details.
 4. Use the hints given in *Using manual file or directory backups* on page 1-16 to copy files from the default settings directory \etc to your debugger home directory. Some of the *.cnf files have no default in \etc, and are recreated as required. If you believe it is causing problems, delete the version in your home directory and let the debugger recreate it when you next connect.
 5. Restart RealView Debugger.

Chapter 4

Programming Flash with RealView Debugger

This chapter describes how to use RealView® Debugger to program Flash memory on your target hardware. It contains:

- *Introduction to Flash programming with RealView Debugger* on page 4-2
- *RealView Debugger files used for Flash programming* on page 4-7
- *Using the pakflash utility* on page 4-10
- *Programming Flash on the ARM boards* on page 4-11
- *Programming Flash for a custom development platform* on page 4-16
- *Gathering information about your target* on page 4-19
- *Creating algorithms for a Flash type supported by RealView Debugger* on page 4-20
- *Creating algorithms for a Flash type not provided with RealView Debugger* on page 4-24
- *Creating the Flash-level and board-level AME files* on page 4-30
- *Generating the FME file* on page 4-36
- *Checking the FME file with the dispflash utility* on page 4-41
- *Creating a BCD file* on page 4-42
- *Programming an image into Flash* on page 4-46
- *Troubleshooting* on page 4-48.

4.1 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 MMethod file
- Board/Chip Definition file.

RealView Debugger provides files for some 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 platform uses a Flash type that is already supported by RealView Debugger.

The following sections give a summary of the files required for programming Flash:

- *Summary of files used to program Flash for the supported platforms* on page 4-3
- *Summary of files used to program supported Flash types on custom platforms* on page 4-5
- *Summary of files used to program custom Flash types on custom platforms* on page 4-6.

4.1.1 Summary of files used to program Flash for the supported platforms

Table 4-1 shows the files used to program Flash for the platforms that are provided with RealView Debugger.

Table 4-1 Files used to program Flash for the supported platforms

Platform	Flash type supported	Files used to create the FME file ^a	Files used to program Flash ^b
ARM Evaluator-7T	AMD	f_amd_sst_arm.s b_evaluator7t.s flash_amd.ame board_amd_eval7T.ame	flash_amd_eval7t.fme Eval7T.bcd
	SST	f_amd_sst_arm.s b_evaluator7t.s flash_sst.ame board_sst_eval7T.ame	flash_sst_eval7t.fme Eval7T.bcd
ARM Integrator™/AP	Intel	f_intel_arm.s b_IntegratorAP.s flash_intel.ame board_intel_arm.ame	flash_IntegratorAP.fme AP.bcd
Atmel AT91EB01, AT91EB01_UB, and AT91EB63	Atmel	f_atmel_arm.s b_at91eb01.s flash_atmel.ame board_atmel_arm.ame	flash_AT91EB01_UB.fme
Motorola NEP29301EVB	Intel	f_intel_arm.s b_m56621evb.s flash_intel.ame board_intel_mot.ame	flash_M56621evb.fme m56621.bcd
Oki ML674000	Toshiba	f_toshiba_arm.s b_ml674000.s flash_toshiba.ame board_toshiba_arm.ame	flash_toshiba.fme
STMicroelectronics ST30	ST30	f_st30_arm.s b_st30.s flash_st30.ame board_st30_arm.ame	flash_st30.fme
Unique ML674K	SST	f_amd_sst_arm.s b_unique_ml674k.s flash_sst.ame board_unique_ml674k.ame	flash_unique_ml674k.fme

Table 4-1 Files used to program Flash for the supported platforms (continued)

Platform	Flash type supported	Files used to create the FME file ^a	Files used to program Flash ^b
Versatile VPB926EJ-S	AMD	f_amd_arm.s b_VPB926EJ-S_amd.s flash_amd.ame board_amd_VPB926EJ-S.ame	VPB926EJ-S_AM29LV256M.fme
	AMD (with locks)	f_amd_lock_arm.s b_VPB926EJ-S_amd_lock.s flash_amd.ame board_amd_lock_VPB926EJ-S.ame	VPB926EJ-S_AM29BDS640G.fme
	SST	f_intel_arm.s b_VPB926EJ-S_intel.s flash_intel.ame board_intel_VPB926EJ-S.ame	VPB926EJ-S_28F256L30B90.fme

a. These files are in the *install_directory\RVD\Core\...\flash\examples\platform* directory.

b. The specified .bcd files are in the *install_directory\RVD\Core\...\etc* directory.

The files provided are:

- assembly files (.s) containing the Flash algorithms and board-level routines required to program the Flash
- ASCII MMethod files (.ame) that describe the Flash memory for the target
- Flash MMethod files (.fme) generated with the pakflash utility from assembly code and ASCII MMethod files
- A Board/Chip Definition file (.bcd) that describes the memory map for your custom platform, and specifies the Flash MMethod file that RealView Debugger is to use to program the Flash device.

RealView Debugger projects to create FME files from these sources are also provided, for example in:

install_directory\RVD\Core\...\flash\examples\IntegratorAP\IntegratorAP.prj.

4.1.2 Summary of files used to program supported Flash types on custom platforms

Table 4-2 shows the files required to program Flash when you have a custom platform that supports a Flash type that is already supported by RealView Debugger. These files are in the *install_directory\RVD\Core\...\flash\examples\platform* directory.

Table 4-2 Files used to program Flash types already supported

Flash type supported	Files used to create the FME file
AMD	f_amd_sst_arm.s flash_amd.ame board_amd_eval7T.ame
ATMEL	f_atmel_arm.s flash_atmel.ame board_atmel_arm.ame
Intel	f_intel_arm.s flash_intel.ame board_intel_arm.ame
SST	f_amd_sst_arm.s flash_sst.ame board_sst_eval7T.ame

In addition to these files, you must create the following files:

b_flashwrapper.s

This defines the board-level operations required for programming the Flash device. It must include the required Flash-level assembly file for the Flash device (for example, *f_amd_sst_arm.s* for SST Flash).

board.bcd

A Board/Chip Definition file that describes the memory map for your custom platform, and specifies the Flash MMethod file that RealView Debugger is to use to program the Flash device.

The assembler and ASCII MMethod files are required to generate the Flash MMethod file.

4.1.3 Summary of files used to program custom Flash types on custom platforms

You must create the following files if you want to program Flash for a custom platform and a custom Flash type:

rvd2apcs.s

This is an assembly wrapper that acts as an interface between the RealView Debugger API your Flash programming code.

flash.h

This is a C header file containing prototypes for the functions called by RealView Debugger.

flash_algorithm.c

This contains your Flash programming routines, which are called by the related routines in *rvd2apcs.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 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.

4.2 RealView Debugger files used for Flash programming

RealView Debugger uses information provided in various target-specific files when programming Flash. These files include:

- *Flash-level code*
- *Board-level code* on page 4-8
- *ASCII MMethod file* on page 4-8
- *Flash MMethod file* on page 4-9
- *Board Chip Definition file* on page 4-9.

RealView Debugger provides source code, AME, FME, and project files for supported targets in `install_directory\RVD\Core\...\flash\examples\target`.

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 4-16.

4.2.1 Evaluator-7T example

The files for the ARM Evaluator-7T platform are in:

`install_directory\RVD\Core\...\flash\examples\eval7t`.

4.2.2 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`.

4.2.3 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`. For a description of the contents of a board-level assembly code file, see *Creating algorithms for a Flash type supported by RealView Debugger* on page 4-20.

Evaluator-7T example

The board-level file provided for the ARM Evaluator-7T is named `b_evaluator7t.s`.

4.2.4 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`. See *Flash-level AME file format* on page 4-30 for a description of the format of a Flash-level AME file.
- a board-level AME file. The files provided with RealView Debugger are named `board_flashtype.ame`. See *Board-level AME file format* on page 4-33 for a description of the format of a board-level AME file.

Evaluator-7T example

The AME files provided for the Evaluator-7T are named:

`board_amd_eval7T.ame`
`board_sst_eval7T.ame`

———— Note —————

Two different files are provided because there are two different versions of this board, each with a different flash device.

4.2.5 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 (see *Flash-level code* on page 4-7) and the AME files (see *ASCII MEthod file* on page 4-8).

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
```

4.2.6 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 (see *Flash MEthod file*).

BCD files have the extension `.bcd`. The files provided with RealView Debugger are named `boardname.bcd`, and are located in `install_directory\RVD\Core\...\etc`.

See *Creating a BCD file* on page 4-42 for instructions on how to create a BCD file.

Evaluator-7T example

The BCD file provided for the ARM Evaluator-7T is named `Eval7T.bcd`.

4.3 Using the pakflash utility

The pakflash utility packs an Flash METHOD file for RealView Debugger to use when programming Flash.

4.3.1 Command syntax

```
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 Method file.
If you do not specify this option, *name* is assumed to be an entry [FLASH=DEFAULT] in the ASCII Method file (see *Creating the Flash-level and board-level AME files* on page 4-30).
- axf-file* The name of the ELF/DWARF (.axf) image containing the Flash programming routines.
- FME-filename* The generated Flash Method 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 (see *Board-level AME file format* on page 4-33), and the Link_advanced.Ro_base setting in the BUILD properties of the project (see *Specifying the linker settings* on page 4-39)
- 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 Method file (see *Creating the Flash-level and board-level AME files* on page 4-30).
If you do not specify an ASCII Method file, the method file name is assumed to be the same as the .axf file name, but with the .ame extension.

4.4 Programming Flash on the ARM boards

RealView Debugger provides Flash programming files for the following ARM boards:

- ARM Evaluator-7T
- ARM Integrator/AP.

All files required to program Flash on the ARM boards are provided with RealView Debugger (see *Summary of files used to program Flash for the supported platforms* on page 4-3).

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 4-16 for more details.

This section describes how to program Flash on the ARM boards, using the Integrator/AP board as an example. It includes:

- *About the Integrator/AP board*
- *Assigning a BCD file to your connection* on page 4-12
- *Reviewing the information contained in the Integrator/AP BCD file* on page 4-12
- *Displaying the memory map in the Process Control pane* on page 4-13
- *Programming an image into Flash on the Integrator/AP* on page 4-14.

4.4.1 About the 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.

4.4.2 Assigning a BCD file to your connection

To assign the BCD file for the Integrator/AP board to a connection:

1. Ensure that RealView Debugger is not connected to a target.
2. Select **Target** → **Connection Properties...** to display the Connection Properties window.
3. Select the connection to which you want to assign the BCD file.
For example, if you are using RealView ICE, select the CONNECTION=RealView-ICE entry in the left-hand pane.

———— **Note** ————

If you are following the example in *Flash programming* on page 3-62, then select the CONNECTION=MP3Player in the left-hand pane.

4. In the right-hand pane, right-click on BoardChip_name, and select **AP** from the context menu.
5. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

4.4.3 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 **Target** → **Connection Properties...** to display the Connection Properties window.
2. Expand the (*.bcd) Board/Chip Definitions entry.
3. Right-click on the ... \AP.brd entry, and select **Expand whole Tree** from the context menu.

The Memory_block entry has many memory areas defined. Each entry corresponds to an entry in the memory map (see *Displaying the memory map in the Process Control pane* on page 4-13).

4. Select the M_FLASH entry.
The Start, Length, and Flash_type entries are those used by RealView Debugger when you program Flash memory.
5. Select **File** → **Close Window** to close the Connection Properties window without making any changes.

4.4.4 Displaying the memory map in the Process Control pane

When you connect to the target to which you have assigned a BCD file, you can display the memory map in the **Map** tab of the Process Control pane.

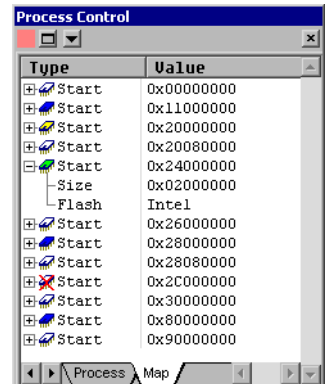
To display the memory map:

1. Connect to your target.
2. Click on the **Cmd** tab in the Output pane.

The information ...Local Advanced_info, BOARD=AP in this pane shows that RealView Debugger is using the Integrator/AP BCD file (AP.bcd). As a result, the memory map now contains the definitions required to use the Flash memory on the Integrator/AP board.

3. Select **View → Memory Map tab** to view the new memory map before loading an image.

Figure 4-1 shows the Flash memory as defined on the Integrator/AP board, and that it is using an Intel flash device.



Type	Value
Start	0x00000000
Start	0x11000000
Start	0x20000000
Start	0x20080000
Start	0x24000000
Size	0x02000000
Flash	Intel
Start	0x26000000
Start	0x28000000
Start	0x28080000
Start	0x2C000000
Start	0x30000000
Start	0x80000000
Start	0x90000000

Figure 4-1 New memory map in the Process Control pane

Note

The Flash_type setting in the connection properties for the Flash memory area definition determines whether or not the Flash Memory Control dialog is displayed (see *Describing the memory map* on page 4-44 for more details).

See the chapter that describes memory mapping in the *RealView Debugger v1.8 User Guide* for more details.

4.4.5 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 necessary, create an image file compiled to run with code at 0x24000000 and that has data in RAM.
In this example, use the dhrystone project, located in your examples directory:
`install_directory\RVDS\Examples\...\dhrystone`
- 2. Open the project and modify the linker settings in the BUILD group as shown in Table 4-3.

Table 4-3 Linker settings for Integrator/AP

Setting	Value
Link_advanced.Ro_base ^a	0x24000000
Link_advanced.Rw_base	0x8000

a. The period indicates that the following item is a sub-option of the preceding item.

- 3. Save your changes and rebuild the project.
- 4. Click **Target** → **Load Image...** and select the image file.
- 5. Click **Open** in the Load File to Target dialog box. The Flash Memory Control dialog box appears, shown in Figure 4-2 on page 4-15. RealView Debugger queues the image in preparation for writing to Flash.

Note

Use the option **Use Current values for Unspecified data in block** only if you are updating part of the Flash block and you want to retain the current values in the rest of the block. This check box is unselected by default. If you select this option, RealView Debugger reads and then writes the entire block. This might take some time to complete.

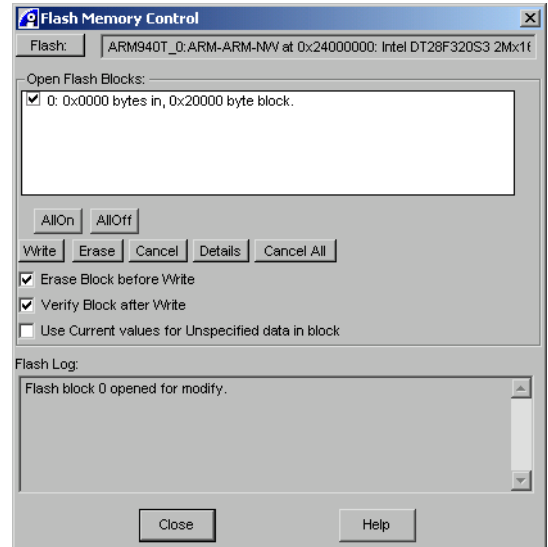


Figure 4-2 The Flash Memory Control dialog box

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 4-3).

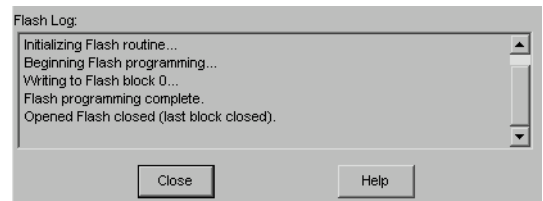


Figure 4-3 Flash programming complete

7. Click **Close** to close the Flash Memory Control dialog box.

4.5 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.

This section includes:

- *Third-party support for ARM core-based development platforms*
- *Programming a Flash type supported by RealView Debugger*
- *Programming a custom Flash type on page 4-17.*

4.5.1 Third-party support for ARM core-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.

4.5.2 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 (see *Summary of files used to program supported Flash types on custom platforms* on page 4-5).

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* on page 4-17, do the following:

1. Create a new directory called eval7t_support.
2. Copy the following files from the ...\\flash\\examples\\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.
4. Where the procedure directs you to create a RealView Debugger project, use the eval7t_support directory as the base for the project.

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 (see *Gathering information about your target* on page 4-19).
2. Create the Flash-level and board-level AME files (see *Creating the Flash-level and board-level AME files* on page 4-30).
3. Generate the FME file. To do this, create a Standard project in RealView Debugger 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.

See *Generating the FME file* on page 4-36 for details.
4. Check the FME file with the dispflash utility (see *Checking the FME file with the dispflash utility* on page 4-41).
5. Create a BCD file (see *Creating a BCD file* on page 4-42).
6. Program your Flash device (see *Programming an image into Flash* on page 4-46).

4.5.3 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 (see *Summary of files used to program custom Flash types on custom platforms* on page 4-6).

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* on page 4-18. 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 ...\\flash\\examples\\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.
4. Where the procedure directs you to create a RealView Debugger project, use the eval7t_custom directory as the base for the project.

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 target (see *Gathering information about your target* on page 4-19).
2. Create the Flash algorithms for the required Flash type, see *Creating algorithms for a Flash type not provided with RealView Debugger* on page 4-24.
3. Create the Flash-level and board-level AME files (see *Creating the Flash-level and board-level AME files* on page 4-30).
4. Generate the FME file. To do this, create a Standard project in RealView Debugger 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.See *Generating the FME file* on page 4-36 for details.
5. Check the FME file with the dispflash utility (see *Checking the FME file with the dispflash utility* on page 4-41).
6. Create a BCD file (see *Creating a BCD file* on page 4-42).
7. Program your Flash device (see *Programming an image into Flash* on page 4-46).

4.6 Gathering information about your target

To provide Flash support for your target, you must have the following information:

- a copy of the datasheet for the Flash device used on your target
- 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.

4.6.1 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 4-4.

Table 4-4 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:

- 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.

4.7 Creating algorithms for a Flash type supported by RealView Debugger

For targets that use one of the Flash types supported by RealView Debugger (see *Introduction to Flash programming with RealView Debugger* on page 4-2) you must create the board-level assembly code. A template board-level assembly code (`b_flashwrapper.s`), complete with comments, is shown in Example 4-1. Also, see *Editing b_flashwrapper.s* on page 4-22 for details on how to edit the file.

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 4-1 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 ; 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
; ----- */
        b1      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

```

4.7.1 Editing b_flashwrapper.s

The areas of `b_flashwrapper.s` that you can edit are (see Example 4-1 on page 4-20):

`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.

4.7.2 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 4-5.

Table 4-5 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>	f_amd_sst_arm.s

4.8 Creating algorithms for a Flash type not provided with RealView Debugger

If your target uses a Flash device that cannot be programmed using the Flash algorithms provided with RealView Debugger, you must provide your own Flash programming code. To do this:

1. Create a C source file containing the Flash-specific C functions (see Example 4-2).
For example, you might provide these functions in a file `myflash_sst.c`.
2. Create the following files:
 - `flash.h` (see Example 4-3 on page 4-26).
This is a C header file containing prototypes for the functions called by RealView Debugger.
 - `rvd2apcs.s` (see Example 4-4 on page 4-27).
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 target. Therefore, you must only specify an RO base in the linker options. See the *RealView Compilation Tools Linker and Utilities Guide* for more details on the RO base option.

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 4-2 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
    }

```

Example 4-3 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.
*****/

```

Example 4-4 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

```

4.9 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. It includes:

- *Flash-level AME file format*
- *Board-level AME file format* on page 4-33.

4.9.1 Flash-level AME file format

Example 4-5 shows the format of a Flash-level AME file. The settings are described in:

- *Mandatory settings* on page 4-31
- *Optional settings* on page 4-31.

You can specify multiple FLASH entries in the same AME file, but each entry must have a unique device name (see ...flash\examples\IntegratorAP\flash_intel.ame for an example).

Example 4-5 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 4-7 on page 4-33). 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` Always set to "ARM", which is the default.

`width=FlashWidth`

The width of the Flash device:

- | | |
|----------|----------------|
| 1 | 8bit (default) |
| 2 | 16bit |
| 4 | 32bit |

———— **Note** —————

The `width=` setting in the board-level AME file overrides this setting.

Evaluator-7T example

Example 4-6 shows a sample Flash-level AME file for the Evaluator-7T. This is the file `flash_sst.ame` in the directory you created (see *Preparing to follow the procedure for a supported Flash type* on page 4-16 or *Preparing to follow the procedure for a custom Flash type* on page 4-17).

Example 4-6 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"
```

4.9.2 Board-level AME file format

Example 4-7 shows the format of a board-level AME file. The settings are described in:

- *Mandatory Settings*
- *Optional Settings* on page 4-34.

You can specify multiple BOARD entries in the same AME file, but each entry must have a unique device name (see ...flash\examples\at91eb01\board_atmel_arm.ame for an example).

Example 4-7 Board-level AME file format

```
[BOARD=BoardName]
proc_name=ARM
from_flash=DeviceName
width=DataWidth
reloc.start_addr=R0baseAddress
reloc.pc_rel=True

[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.

This is the name you use with the pakflash -f *name* option (see *Using the pakflash utility* on page 4-10).

proc_name Always set to ARM.

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 4-5 on page 4-30). 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:

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 *Link_advanced.Ro_base* setting in the BUILD properties of the project (see *Specifying the linker settings* on page 4-39).

The *pakFlash -s addr* option overrides this setting (see *Using the pakflash utility* on page 4-10).

————— 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:

- | | |
|----------|----------------|
| 1 | 8bit (default) |
| 2 | 16bit |
| 4 | 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 (see *Creating algorithms for a Flash type supported by RealView Debugger* on page 4-20).

————— Note —————

This setting overrides the *width=* setting in the Flash-level AME file.

Evaluator-7T example

Example 4-8 shows a sample board-level AME file for the Evaluator-7T. This is the file `board_sst_eval7T.ame` in the directory you created (see *Preparing to follow the procedure for a supported Flash type* on page 4-16 or *Preparing to follow the procedure for a custom Flash type* on page 4-17).

In your copy of `board_sst_eval7T.ame`, change the `BOARD=` value to that shown in Example 4-8.

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 4-8 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
```

4.10 Generating the FME file

Before you can generate the FME file, you must:

1. Create Flash algorithm code (see *Creating algorithms for a Flash type supported by RealView Debugger* on page 4-20 and *Creating algorithms for a Flash type not provided with RealView Debugger* on page 4-24).
2. Create Flash-level and board-level AME files (see *Creating the Flash-level and board-level AME files* on page 4-30).

This section describes how to create a Standard RealView Debugger project to generate the FME file for the Evaluator-7T Flash example. It includes:

- *Locating the source and AME files required*
- *Creating a Standard RealView Debugger project* on page 4-37
- *Specifying the compiler settings* on page 4-38
- *Specifying the assembler settings* on page 4-38
- *Specifying the linker settings* on page 4-39
- *Running the pakflash utility* on page 4-39.

4.10.1 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`).

See *Creating algorithms for a Flash type supported by RealView Debugger* on page 4-20.

- 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`.

See *Creating algorithms for a Flash type not provided with RealView Debugger* on page 4-24.

You must also locate the AME files you want to use (see *Creating the Flash-level and board-level AME files* on page 4-30).

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).

4.10.2 Creating a Standard RealView Debugger project

This section assumes that you are familiar with the procedure for creating projects with RealView Debugger. For more details, see the chapter that describes creating projects in the *RealView Debugger v1.8 Project Management User Guide*.

To create a Standard project:

1. Select **Project** → **New Project...** from the default Code window main menu to display the Create New Project dialog box.

2. Complete the entries in the Create New Project dialog box:

Project Name

Enter the name of your project.

For the Evaluator-7T example, enter the text `Flash_Eval7T`.

Project Base

Locate the directory containing the source and AME files you want to use (see *Locating the source and AME files required* on page 4-36).

Select Type of Project

Select **Standard project (Compile/Assemble/Link)**.

3. Click **OK**.
4. RealView Debugger displays the Create Standard Project dialog box. Select the toolchain you are using.
For example, select ARM-C22: ARM RVCT 2.2 if you are using RealView Compilation Tools v2.2.
5. Select the source files you are using:

- If you are using Flash algorithms provided with RealView Debugger, then select your version of `b_flashwrapper.s`.

————— Note —————

Do not select the Flash algorithm file, because this is included by `b_flashwrapper.s`.

For the Evaluator-7T example, select `b_flashwrapper_eval7t.s`.

- If you are using Flash algorithms that you have created, then select the following source files:
 - `rvd2apcs.s`
 - `flash_algorithm.c`.

For the Evaluator-7T example, select `rvd2apcs.s` and `myflash_sst.c`.

See *Locating the source and AME files required* on page 4-36.

6. Click **OK** to confirm the project details and close the dialog box.
RealView Debugger creates a project file with the project name you specified in step 2, and displays the Project Properties window.
7. Make the changes to the project properties described in the following sections:
 - *Specifying the compiler settings*
 - *Specifying the assembler settings*
 - *Specifying the linker settings* on page 4-39
 - *Running the pakflash utility* on page 4-39.
8. Build the project to generate the FME file. See *Building the project* on page 4-40.

4.10.3 Specifying the compiler settings

If you have C code, then set up any compiler settings in the Project Properties `COMPILE=arm` group. For example, make sure that the endianness is the same as the endianness for your debug target.

Evaluator-7T example

Set the `Compilation.Endianness` setting to `little`.

4.10.4 Specifying the assembler settings

If you have assembly code, then set up any assembler settings in the Project Properties `ASSEMBLE=arm` group. For example, make sure that the endianness is the same as the endianness for your debug target.

Evaluator-7T example

Set the `Assembly.Endianness` setting to `little`.

4.10.5 Specifying the linker settings

Set up the linker settings in the Project Properties BUILD group.

The Link_advanced.Ro_base setting is overridden by the reloc.start_addr= setting in the build-level AME file (see *Board-level AME file format* on page 4-33), and by the pakflash -s addr option (see *Using the pakflash utility* on page 4-10).

Note

You must always set the Link_advanced.Remove_unused setting to disabled.

Evaluator-7T example

Set the linker settings as shown in Table 4-6.

Table 4-6 Linker settings

Setting	Value
Debug_info (optional)	enabled
Link_advanced.Ro_base ^a	0x40000
Link_advanced.Remove_unused	disabled

a. The period indicates that the following item is a sub-option of the preceding item.

4.10.6 Running the pakflash utility

To make the RealView Debugger project run the pakflash utility automatically after the link stage is complete, set the Pre_Post_Link.post_link setting in the Project Properties BUILD group to:

```
'$RVDEBUG_BASE\bin\pakflash' -f BoardName debug\project_name.axf
board-level_AME_file_name.ame -o project_name.fme
```

BoardName The name assigned to the BOARD= in the board-level AME file. See *Board-level AME file format* on page 4-33 for details.

project_name The name of the project that you specified in *Creating a Standard RealView Debugger project* on page 4-37.

board-level_AME_file_name

The name of the board-level AME file that you are using (see *Creating the Flash-level and board-level AME files* on page 4-30).

See *Using the pakflash utility* on page 4-10 for more details on using the pakflash utility.

Evaluator-7T example

Set the Pre_Post_Link.post_link setting to:

```
'$RVDEBUG_BASE\bin\pakflash' -f ARM_Eval7T debug\Flash_Eval7T.axf  
board_sst_eval7T.ame -o Flash_Eval7T.fme
```

4.10.7 Building the project

After you have set up the Project Properties, do the following:

1. In the Project Properties window, select **File** → **Save and Close**.
RealView Debugger generates the makefiles for the project, and displays messages in the **Build** tab of the Output pane.
2. Select **Build** → **Rebuild All** to build the project.
RealView Debugger displays the progress of the build in the **Build** tab of the Output pane.

4.11 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 pane.
2. Enter the following command:

```
host dispflash path\filename.fme
```

4.11.1 Evaluator-7T example

For example, to check the contents of the file created in *Generating the FME file* on page 4-36, 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. For more details, see the chapter that describes reading and writing memory, registers and Flash in the *RealView Debugger v1.8 User Guide*.

4.12 Creating a BCD file

By default RealView Debugger looks for BCD files in the *install_directory\RVD\Core\...\etc* directory. BCD files are provided for the various ARM development boards.

You can create a new BCD from within RealView Debugger by copying an existing BCD. For more details on board files see *Creating a new .bcd file* on page 3-21.

This section does not describe all aspects of *Extended Target Visibility* (ETV), for example, enumeration of registers. It describes only those aspects required to program the Flash.

To create a new minimal BCD for the Evaluator-7T follow these steps:

1. Copy and rename an existing BCD file (see *Copying and renaming an existing BCD file*).
2. Delete any existing Board/Chip definitions in the BCD file (see *Deleting any existing Board/Chip definitions* on page 4-43).
3. Create a new board group in the BCD file (see *Creating a new board group in the BCD file* on page 4-43).
4. Describe the memory map for your hardware (see *Describing the memory map* on page 4-44).
5. Assign the new BCD file to the required target connection (see *Assigning your BCD file to a target connection* on page 4-45).

4.12.1 Copying and renaming an existing BCD file

Do the following:

1. Locate an existing BCD file in the *...\etc* directory.
2. Make a copy of this file, rename it, and then copy the new file back to the *...\etc* directory.

For the Evaluator-7T example, copy the file *...\etc\Eval7T.bcd*, and rename it to *ARM_Eval7T.bcd*.

4.12.2 Deleting any existing Board/Chip definitions

When you copy an existing BCD file to create a new BCD file, you must delete any existing Board/Chip definitions:

1. Select **Target** → **Connection Properties...** to display the Connection Properties window.
2. Expand the (*.bcd) Board/Chip Definitions entry.
3. Select the entry for your board file.
For the Evaluator-7T example, select the ... \ARM_Eval7T.bcd entry.
4. For each BOARD and CHIP group in the right-hand pane:
 - a. Right-click on the group.
 - b. Select **Delete** from the context menu to delete it.

4.12.3 Creating a new board group in the BCD file

To create a new board group in your BCD file:

1. Right-click on the entry for your BCD file, and select **Make New Group...** from the context menu.
The Group Type/Name selector dialog is displayed.
For the Evaluator-7T example, right-click on the ... \ARM_Eval7T.bcd entry.
2. Select the BOARD group type.
3. Enter a name for the group.
This name must match the BOARD= entry in your board-level AME file (see *Board-level AME file format* on page 4-33).
For the Evaluator-7T example, enter the text enter ARM_Eval7T for the group name.
4. Click **OK**.

4.12.4 Describing the memory map

- To describe the memory map for your hardware:
1. In the left-hand pane, right-click on the entry for your board/chip file, and select **Expand whole Tree** from the context menu.
For the Evaluator-7T example, right-click on the ... \ARM_Eval7T.bcd entry.
 2. Select the Memory_block entry.
 3. Create an entry for each area of memory for your board (see *Gathering information about your target* on page 4-19). To do this:
 - a. In the right-hand pane, right-click on the Default entry, and select **Make Copy...** from the context menu.
 - b. In the dialog box, enter a name for the area of memory you are defining.
For the Evaluator-7T example, enter Eval_App_Flash.
 - c. Repeat steps a and b to create the entries for other memory areas.
For the Evaluator-7T example, create entries with the following names:
 - SRAM_bank1
 - SRAM_bank2.
 4. For each memory area entry you created in step 3, set the attributes of each memory area.
For the Evaluator-7T example, set the attributes as shown in Table 4-7.

Table 4-7 Memory map attributes

Attribute	Eval_App_Flash	SRAM_bank1	SRAM_bank2
Start	0x01820000	0x0	0x40000
Length	0x60000	0x40000	0x40000
Access ^a	Flash	RAM	RAM
Flash_type ^b	Location of the Flash_Eval7T.fme file ^c		
Description	Application Flash 384K	SRAM bank1 256K	SRAM bank2 256K

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. See *Generating the FME file* on page 4-36.

5. Right-click on the Default entry, and select **Delete** from the context menu.

6. Select **File** → **Save and Close** to close the Connection Properties window and save your changes.

4.12.5 Assigning your BCD file to a target connection

When you have modified your new BCD file, you must assign it to the connection for your target hardware.

Note

This section assumes that you have access to a compatible JTAG debug interface, such as RealView ICE.

To assign a BCD file to a connection:

1. Ensure that RealView Debugger is not connected to a target.
2. Select **Target** → **Connection Properties...** to display the Connection Properties window.
3. Select the connection to which you want to assign the BCD file.
For example, if you are using RealView ICE, select the CONNECTION=RealView-ICE entry in the left-hand pane.
4. In the right-hand pane, right-click on BoardChip_name, and select your board/chip file from the context menu.
For the Evaluator-7T example, select **ARM_Eval7T** from the context menu.
5. Select **File** → **Save and Close** to close the Connection Properties window and save your changes.

4.13 Programming an image into Flash

This section describes how to program an image to Flash. It includes:

- *Writing the image to Flash*
- *Using CLI commands to program Flash* on page 4-47
- *Checking the contents of Flash* on page 4-47.

4.13.1 Writing the image to Flash

To program an image into Flash using RealView Debugger:

1. Assign the BCD file that describes the memory map and specifies the FME file to the target connection (see *Assigning your BCD file to a target connection* on page 4-45).
2. Build your Flash image using the linker setting `Link_advanced.Ro_base`. Make sure that this points to a valid Flash address.
3. Connect to the target. See Chapter 2 *Connecting to Targets* for more details.
4. Queue your Flash image for programming into the Flash device. You can do this using one of the following methods:
 - Load the Flash image using the Load File to Target dialog box, or select an image from the **Target** → **Recent Images** list. See the chapter that describes working with images in the *RealView Debugger v1.8 User Guide* for instructions on how to load an image.
 - Load the Flash image using the Upload/Download file from/to Memory dialog box.

In both cases, the Flash Memory Control dialog box is displayed.
5. 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 BCD file.

See the chapter that describes reading and writing memory, registers, and Flash in the *RealView Debugger v1.8 User Guide* for instructions on how to use the following dialog boxes:

- Upload/Download file from/to Memory dialog box
- Flash Memory Control dialog box.

4.13.2 Using CLI commands to program Flash

You can use the following CLI commands to program an image to Flash:

- LOAD, to load the Flash image, for example:
LOAD 'C:\my_projects\flash\my_flash_image.axf'
- FLASH, to perform operations on Flash. For example, to write the image to Flash enter:
FLASH,write

For more details on how to use these commands, see the *RealView Debugger v1.8 Command Line Reference Guide*.

4.13.3 Checking the contents of Flash

You can check the contents of Flash as follows:

1. Make sure that the Memory pane is visible. If it is not, then select **View** → **Memory** from the Code window main menu.
2. In the Memory pane, right-click on an address in the left-hand column.
3. Select **Set Start Address...** from the context menu.
4. Enter the start address of your Flash memory.
 - for the Evaluator-7T example, enter 0x18000000
 - for the Integrator/AP, enter 0x24000000.

The contents of memory starting at this address are displayed in the Memory pane. The contents of Flash are shown in green.

For more details on using the Memory pane, see the chapter that describes reading and writing memory, registers, and Flash in the *RealView Debugger v1.8 User Guide*.

4.14 Troubleshooting

If you are having problems programming an image to Flash, check the following:

- Does the BCD file correctly describe the Flash memory area for your board, and does it specify the correct FME file? See *Creating a BCD file* on page 4-42.
- Does your target connection have the correct BCD file assigned to it? See *Assigning your BCD file to a target connection* on page 4-45.
- Is the icon for the Flash area in the **Map** tab green, and are the Flash memory locations in the Memory pane 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 `displflash` 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 this is not, then:

1. Display the Project Properties for the project that you are using to build the Flash image.
2. Change the value for the linker setting `BUILD.Link_advanced.Ro_base` to a valid value.
3. Rebuild the FME file.

————— **Note** —————

If you built the image using the compilation tools directly, see the `--ro_base` linker option in the *RealView Compilation Tools Linker and Utilities Guide*.

- Does your Flash image have the same endianness as the FME file?
 - Build your Flash image and FME file with the correct endianness. Set either the `Compilation.Endianness` setting for the ARM compiler or `Assembly.Endianness` setting for the ARM assembler in the Project Properties.
To check the endianness of the FME file, use the `displflash` command described in *Checking the FME file with the displflash utility* on page 4-41.
 - If you built the image using the compilation tools directly, see the `--littleend` or `--bigend` options in *RealView Compilation Tools Compiler and Libraries Guide* or in *RealView Compilation Tools Assembler Guide*.

- Make sure that your target hardware and connection configurations 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? See *Creating the Flash-level and board-level AME files* on page 4-30.
 - Are your Flash algorithms correct? See *Creating algorithms for a Flash type supported by RealView Debugger* on page 4-20 or *Creating algorithms for a Flash type not provided with RealView Debugger* on page 4-24 as appropriate.

Chapter 5

Configuring Custom Connections

This chapter describes how you can configure the connection that RealView® Debugger makes to your target. It includes information on the board file groups CONNECTION and DEVICE, and explains how connections to different targets are configured. It contains the following sections:

- *Working with connection properties* on page 5-2
- *Working with RealView ICE targets* on page 5-7
- *Working with RDI targets* on page 5-14
- *Working with Simulator Broker connections* on page 5-28
- *Working with JTAG files* on page 5-36.

5.1 Working with connection properties

As described in *About target connections and configuration* on page 1-2, you configure the way that RealView Debugger connects to, and interacts with, your debug target using connection properties contained in board file entries. A debug target might be a simulator, an emulator, or a development board installed on your host workstation. Using connection entries enables you to configure:

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

This section describes how to work with connection entries. It contains the following sections:

- *Connection entries in the Connection Control window* on page 5-3
- *Enabling or disabling a board file entry* on page 5-4
- *Changing entries containing user-information values* on page 5-5
- *Restoring board file entry defaults* on page 5-6.

In the examples in the rest of this chapter, you are changing your board file. This is stored in your RealView Debugger home directory. Target configuration files might also be stored in this directory, for example .rvc files or .cnf 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. For details see:

- *Configuration files* on page 1-12 for instructions on making backups of your configuration
- *Restoring your .brd file* on page 3-62 for instructions on restoring a default configuration
- *Troubleshooting* on page 3-62 for instructions on recovering from an incorrectly configured debugger home directory, whether or not you have a backup.

Note

When you are following these examples, do not configure the board file when the debugger is connected to a target.

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 you are familiar with the procedures described in this help topic.

5.1.1 Connection entries in the Connection Control window

Board file entries that are enabled form the basis of the information displayed in the Connection Control window, shown in Figure 5-1.

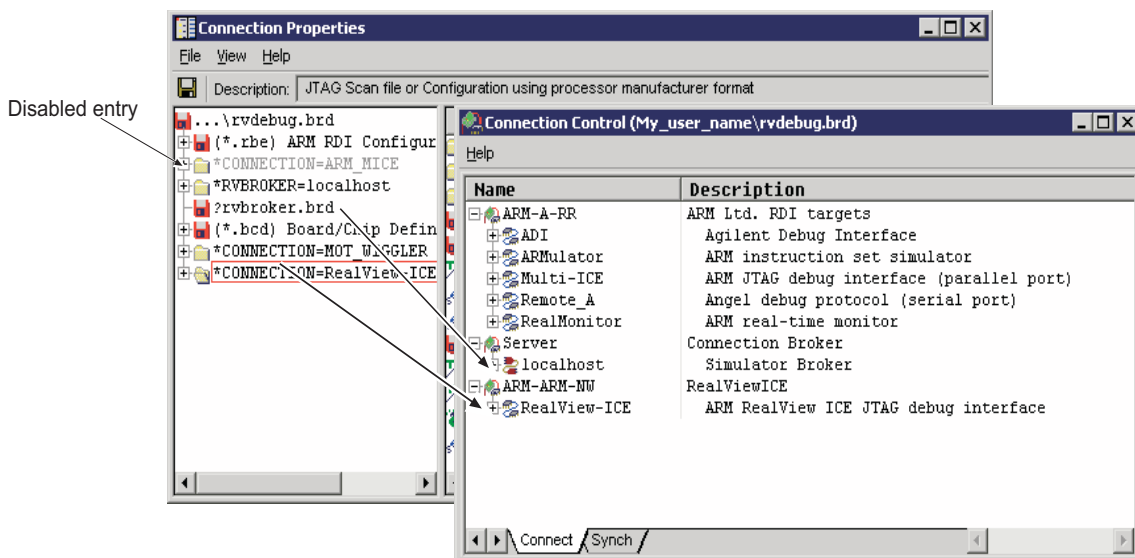


Figure 5-1 Connection properties entries in the Connection Control window

In Figure 5-1 the entry `CONNECTION=ARM_MICE` has been disabled in the board file. Therefore, this entry does not appear in the Connection Control window. Other connection entries are enabled and so are visible, for example `CONNECTION=Multi-ICE`, `CONNECTION=RealView-ICE`, and the local host simulator using RealView Connection Broker.

You can enable and disable any connection entry in the board file if required. When you view the Connection Properties window, disabled entries are grayed out in the left pane, the List of Entries pane. Disabled entries can be edited in the same way as enabled entries and then enabled when available for connection. See *Enabling or disabling a board file entry* on page 5-4 for details on how to do this.

If you make changes to values in the Connection Properties window, an asterisk is added to each entry, in the left or the right pane, to show that the defaults have changed. You can restore the default settings and so cancel any changes. See *Restoring board file entry defaults* on page 5-6 for details on how to do this.

When you display the Connection Properties window, the left pane shows the top-level entries specifying the supported vehicles, for example ARM RDI Configuration Entries. Board file entries might have duplicate names because entries are uniquely identified through the combination of three elements:

- the CONNECTION entry
- the name of the manufacturer of the simulator, emulator, or board
- the host workstation I/O device address, IP address (for remote connections), or ID, of the emulator or board.

For example, assume that you have a target board named MP3P1ayer that you want to use with two different emulators. The board file entry name for each is MP3P1ayer to reflect the target. However, the entries are differentiated by the type of connection (emulator type), and I/O device connection addresses.

————— **Note** —————

You can create your own custom entries in this hierarchy using other types of entry, for example BOARD, CHIP, COMPONENT, or DEVICE. However, if you are creating custom, lower-level entries, it is recommended that you avoid duplicate names.

For full information on the contents and values contained in different types of board file entries, both default and custom, see Appendix A *Configuration Settings Reference*.

5.1.2 Enabling or disabling a board file entry

To disable a board file entry so that the target it represents is no longer offered for selection in the Connection Control window:

1. Start RealView Debugger without connecting to a target.
2. Select **Target** → **Connection Properties...** to display the Connection Properties window.
Enabled entries in the left pane are displayed in regular type, and those that are disabled are grayed out.
3. Expand the connection that you are using, for example RVBROKER=localhost.
It becomes the selected entry and its contents are displayed in the right pane.

4. Right-click on the Disabled entry, in the right pane, and select **True** from the menu.
5. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.
6. Select **Target** → **Connect to Target....** to display the Connection Control window where this entry is no longer available.

5.1.3 Changing entries containing user-information values

Where entries contain user-information values, you can customize them as follows:

1. Select **Target** → **Connection Properties...** to display the Connection Properties window.
2. Expand the entry, in the left pane.
3. Click on the RVBROKER=localhost entry, in the left pane. It becomes the selected entry and its contents are displayed in the right pane.
4. Right-click on the Description entry, in the right pane, and select **Edit Value...** from the context menu.

The text defined by this entry appears in the Description in the Connection Control window, shown in Figure 5-1 on page 5-3. Enter a new description, for example RealView ARMulator ISS and press Enter.

5. Select **File** → **Close Window** to close the Connection Properties window without saving this change. This generates a dialog box warning that contents have changed and giving you the option of saving them. Do not save this change.

This example demonstrates:

- the Description setting has been changed in your user-specific copy of the board file (... \home\My_user_name\rvdebug.brd), that is Simulator Broker is no longer the value for this setting (the default board file in ... \etc\rvdebug.brd is not changed)
- because the default setting has been changed, an asterisk marks the entry
- RealView Debugger warns you if you have not saved changes you have made to the board file settings.

5.1.4 Restoring board file entry defaults

An entry starts with an asterisk when you have changed that entry from the default. For group entries, this might mean that a value lower down in the hierarchy has been changed.

To restore the default values for an entry:

1. Select **Target** → **Connection Properties...** to display the Connection Properties window.
2. Locate the entry that you want to reset to the default value.
For example, in *Enabling or disabling a board file entry* on page 5-4, the `RVBROKER=localhost` entry was disabled. This means that it now contains a custom setting, that is the default setting has been changed. Click on the `RVBROKER=localhost` entry, in the left pane. It becomes the selected entry and its contents are displayed in the right pane.
3. Right-click on the entry in the right pane, and select the appropriate option from the context menu:
 - **Reset to Default** for entries that have a default value in the default board file. 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. This clears the value for the entry.

The asterisk is removed.

For the example in step 2, select **Reset to Default** for the `*Disabled` entry. This sets the value of this setting to `False` as defined in the default board file.

4. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

The original contents of the Connection Control window are restored.

5.2 Working with RealView ICE targets

When you install the RealView ICE software, it adds entries to your RealView Debugger installation. This updates the default board file so that an ARM-ARM-NW target vehicle appears in the Connection Control window to provide the interface to RealView ICE targets, shown in Figure 5-2.

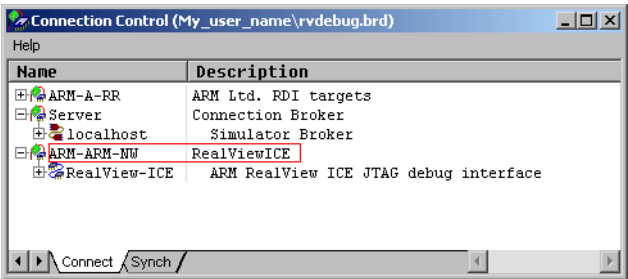


Figure 5-2 RealView ICE targets in the Connection Control window

————— **Note** —————

If the RealView-ICE target vehicle is not visible in the Connection Properties window, you must add it before continuing with this section. See the chapter that describes configuring a RealView ICE connection in *RealView ICE User Guide* for full details on how to do this.

If you expand the ARM-ARM-NW vehicle, you can see the access-provider connection defined automatically, that is RealView-ICE shown in Figure 5-2. You can then configure this connection by specifying an appropriate .rvc file, ready to make a connection. However, it is recommended that you use this default connection as the basis for new connections that you create.

This section describes:

- *Creating a new RealView ICE connection* on page 5-8
- *Configuring a RealView ICE interface unit* on page 5-10.

————— **Note** —————

You can use RealView ICE to connect to a target that incorporates DSP hardware, see *Configuring a new DSP connection with RealView ICE* on page 5-13 for details.

5.2.1 Creating a new RealView ICE connection

This example defines a new RealView ICE connection. You can do this by creating a new connection entry or by copying an existing entry.

———— Note ————

If you copy a connection entry, certain configuration details are not updated until you close the Connection Properties window. This might mean that some asterisks are not immediately visible in the copy.

This example creates a new connection entry. It assumes that a correctly configured .rvc file exists for the new target and this has been saved in the default RealView ICE installation directory. If you do not have this file, you can follow the example but you must also follow the instructions in *Configuring a RealView ICE interface unit* on page 5-10 before you can connect to the new target.

To define the new connection:

1. Select **Target** → **Connection Properties...** to display the Connection Properties window.
2. Right-click on the CONNECTION=RealView-ICE entry, in the left pane.
3. Select **Make New...** from the context menu.
4. This displays the Group Type/Name selector dialog box, shown in Figure 5-3.

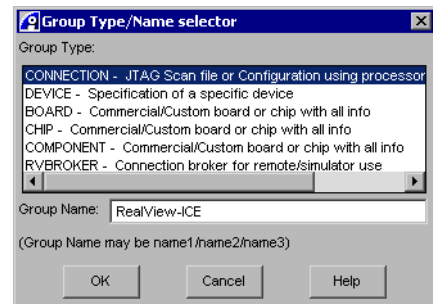


Figure 5-3 Specifying a new CONNECTION group

Leave the type of the new entry unchanged as CONNECTION.

In the Group Name data field change the name from RealView-ICE to something suitable for your target, using only alphanumeric characters, underscore _, and dash -, for example RealView-ICE_2.

This can be a descriptive name or the name of the new .rvc file that you are going to select, but without the .rvc extension.

5. Click **OK** to confirm your settings and to close the Group Type/Name selector dialog box.

The new entry appears in the left pane of the Connection Properties window. It is automatically selected, and its details are displayed in the right pane. These details are the default for a new CONNECTION and you must change at least the Connect_with/Manufacturer, the Configuration filename and target Description. The next steps explain how to make these changes.

6. In the right pane of the Connection Properties window, right-click on the Configuration entry and select **Edit as Filename** from the context menu.

The Enter New Filename dialog box is displayed to enable you to locate the required .rvc file, for example RealView-ICE_2.rvc.

7. Click **Save** to confirm your entries and to close the Enter New Filename dialog box.

The new pathname is displayed in the right pane.

8. In the right pane of the Connection Properties window, right-click on the Description field and select **Edit Value** from the context menu.

Type RealView-ICE connection to test hardware board in the entry area and press Enter.

This is the description displayed in the Connection Control window and Connection Properties window to identify the new target.

9. In the right pane of the Connection Properties window, right-click on the Connect_with entry and select **Explore** from the context menu.

10. In the right pane of the Connection Properties window, right-click on the Manufacturer entry and select the required connection type from the context menu, that is **ARM-ARM-NW**.

If you do not specify this setting, the new connection appears in the Connection Control window but, when you try to connect, RealView Debugger prompts for the connection type.

11. If you want the trace analyzer to be connected automatically when you connect to this target, you can define the analyzer settings. See *Logic_Analyzer* on page A-20 for more details.

Expand the following entries in turn:

- a. CONNECTION=New_RealView_ICE_2
- b. Advanced_Information
- c. Default

12. Select Logic_Analyzer in the left pane.
13. Right-click on Vendor in the right pane, and select, for example, **ARM** from the context menu.
14. If you want to configure other settings for your target, for example reference a specific .bcd file, make these changes now.
15. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

Your new RealView ICE target is now displayed in the Connection Control window.

5.2.2 Configuring a RealView ICE interface unit

Use RVConfig to configure a new RealView ICE interface unit:

1. Right-click on your new RealView ICE connection to display the context menu.
2. Select the option **Configure Device Info...** to display the RVConfig dialog box shown in Figure 5-4.

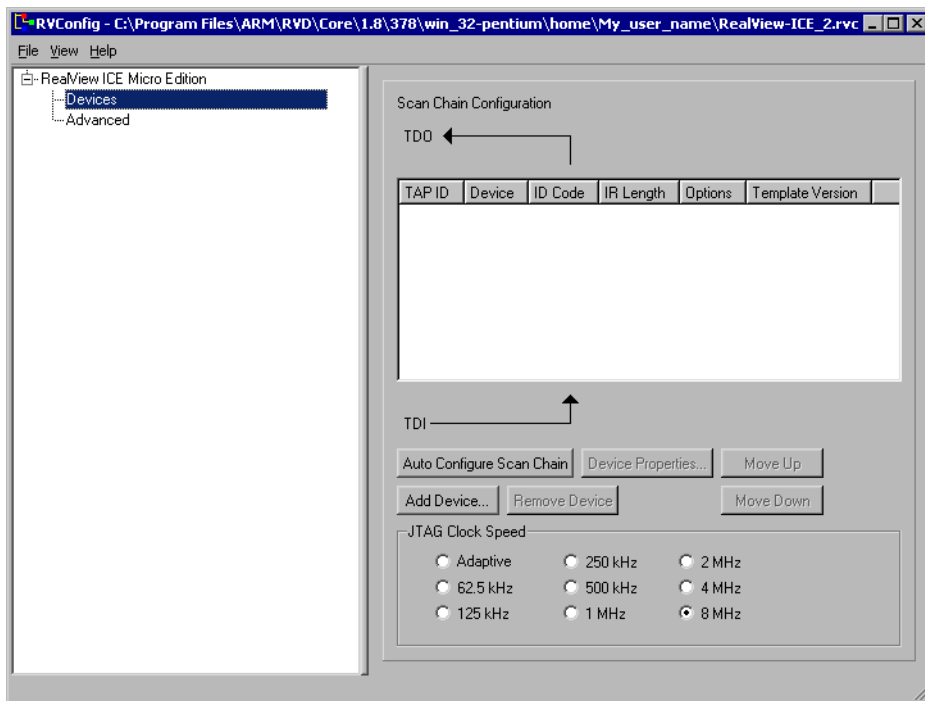


Figure 5-4 Configuring a RealView ICE interface unit

3. Click **Browse** to locate the RealView ICE unit, if you do not know the correct address.
4. Click **Connect** to connect the RVConfig dialog box to the RealView ICE unit.
5. If your target does not have any DSPs, click **AutoConfigure Scan Chain** to detect all the ARM® architecture-based processors included on the target hardware and add them to the scan chain.

If your target has DSPs, click **Add Device...** to display the Add Device dialog box. Select the DSPs, and any other processors that are on the target. Click OK. See *Configuring a new DSP connection with RealView ICE* on page 5-13 for an example of how to do this.

6. Select the device, for example ARM940T, in the left pane, to configure it using the supplied template. For example, you might want to set the Code Sequence Code Address or Code Sequence Code Size, to specify target memory available to RealView ICE.
7. Select Advanced, in the left pane, to configure advanced settings, for example, you might want to click the **Reset On Disconnect (Default)** check box so that the target hardware is reset when you disconnect.

If you are using an Integrator™ board, as in this example, it is recommended that you use the settings shown in Figure 5-5.

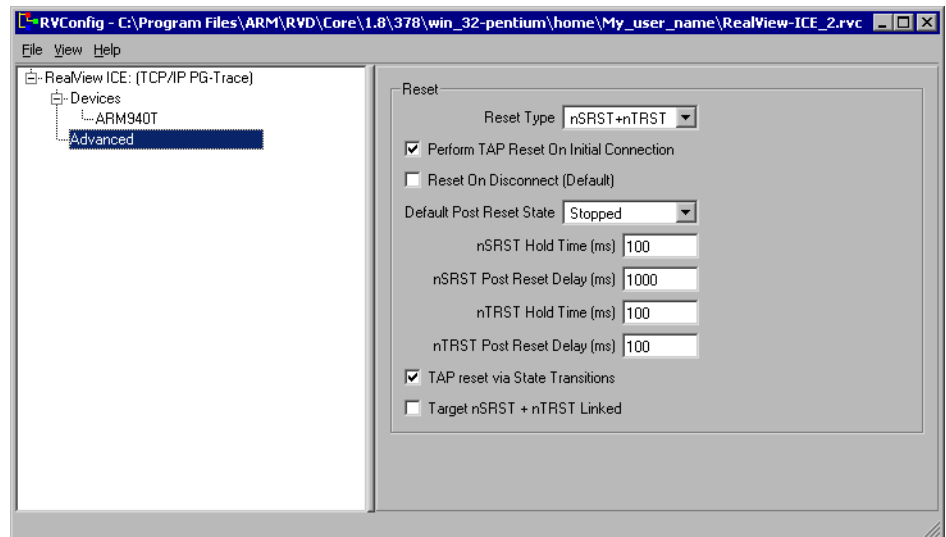


Figure 5-5 Recommended settings for an ARM Integrator board

8. Select **File** → **Save** to save your changes.

9. Select **File** → **Exit** to close the RVConfig dialog box.

You can now connect to your new target in the usual way.

Configuring RealView ICE units for different targets

You might have to use different configuration settings, from those shown in this example, depending on your debugging environment. Remember the following when specifying these settings:

- Autoconfiguration does have side effects and might be intrusive. Where this is not acceptable, you must configure the target manually.
- Autoconfiguration does not work for targets that have DSPs on the scan chain. If your target hardware contains a DSP on the scan chain, you must manually add devices to the scan chain.
- Because DSPs typically run at much lower CPU clock speeds than an ARM core, the JTAG clock speed for a multiprocessor system with an ARM core and a DSP might be lower than a system containing only ARM cores. 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 multi-core system. For example, if the target hardware scan chain contains an ARM core running at 150MHz and a DSP running at 40MHz, then the maximum JTAG clock speed is one eighth of 40MHz, that is 5MHz.
- You must use different settings if your application uses interrupts to set up pre-initialization code on the target, for example if you are using *Running System Debug* (RSD). See *RealView Debugger v1.8 Extensions User Guide* for details.
- The RealView ICE scan chain configuration lists devices in ascending order of TAP ID. This is the opposite order to that used by Multi-ICE®.

For full details on using RVConfig, and on how to add new devices to the scan chain, see the chapter that describes configuring a RealView ICE connection in *RealView ICE User Guide*.

Configuring a new DSP connection with RealView ICE

You can use RealView ICE to connect to a target that incorporates DSP hardware. For example, if your target contains an ARM core and a DSP core, you can use RealView ICE to connect to the ARM core and the DSP core.

Note

You cannot use the autoconfigure device option on the RVConfig dialog box when your target contains a DSP. See the *RealView ICE User Guide* for details on how to manually configure a device.

5.3 Working with RDI targets

This section describes how to use the Connection Control window to add and configure RDI targets. The settings defined in your configuration files control the available targets and the emulators and simulators offered. Therefore, your installation might vary from the examples shown in this section.

Note

RDI connections cannot be made if you are using Sun Solaris or Red Hat Linux.

The RealView Debugger base product includes RDI configuration files to simulate the ARM7TDMI® core using *RealView ARMulator® ISS* (RVISS). You can change this by reconfiguring RVISS, described in *Configuring RealView ARMulator ISS* on page 5-19.

Note

The RDI connection to RVISS in RealView Debugger is deprecated in this release.

When you first try to connect to another RDI target, for example Multi-ICE, you must configure the target first, described in *Configuring ARM Multi-ICE* on page 5-23.

To add a new RDI target to the RealView Debugger configuration settings, you must:

1. add the required DLL to specify the device
2. configure the new target.

See the following sections for details on these operations:

- *The RDI Target List dialog box* on page 5-15
- *Adding and configuring RDI targets* on page 5-17
- *Configuring RealView ARMulator ISS* on page 5-19
- *Configuring Remote_A* on page 5-22
- *Configuring ARM Multi-ICE* on page 5-23
- *Configuring ARM Agilent Debug Interface* on page 5-26
- *Configuring ARM RealMonitor* on page 5-26.

5.3.1 The RDI Target List dialog box

To add, remove, and edit RDI targets, you use the RDI Target List dialog box shown in Figure 5-6.

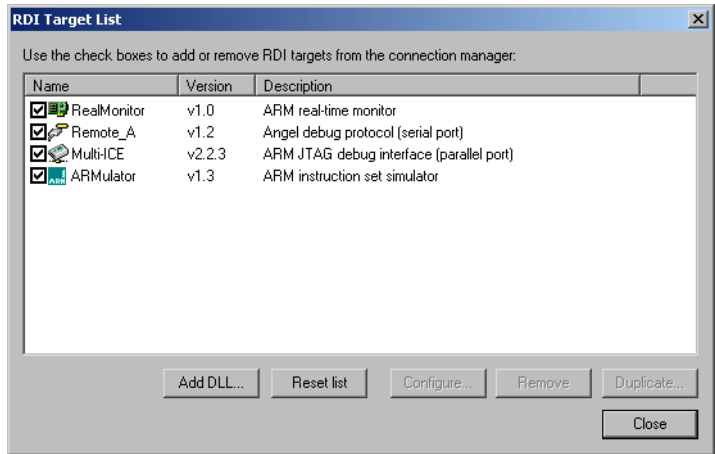


Figure 5-6 RDI Target List dialog box

Displaying the RDI Target List configuration dialog box

To display the RDI Target List dialog box:

1. Start RealView Debugger without connecting to a target.
2. Select **Target** → **Connect to Target...** to display the Connection Control window.
3. Right-click on the ARM RDI vehicle, ARM-A-RR, to display the context menu.
4. Select the **Add/Remove/Edit Devices...** option.

RDI Target List configuration dialog box interface components

Figure 5-6 shows an example RDI Target List dialog box. The entries in the display list show the autodetected targets currently available to you. If you install a new DLL separately from RealView Debugger, for example RealMonitor, the display list is automatically updated for you.

You can use the controls on this dialog box to perform the following actions:

Enable/disable devices

The check boxes show which devices are enabled. Disabling an entry removes it from the list of available connections shown in the Connection Control window. However, the entry is not removed from the RDI Target List and so can be re-enabled when required.

Add a new device

If you do want to add a target yourself, click **Add DLL....** See *Adding RDI targets* on page 5-17 for more details.

Duplicate an existing device

You can duplicate any entry in the display list so that specific processor configurations are available in the Connection Control window. Highlight the entry to be copied and click **Duplicate...** to display the Create New RDI Target dialog box. See *Adding RDI targets* on page 5-17 details on using this dialog box.

Configure a device

Any DLLs added to the list manually, or new entries created by duplicating an existing target, must be configured before you can connect. Highlight the entry in the display list and click **Configure...** to display the configuration dialog box for the chosen entry. This is described in detail in *Configuring RDI targets* on page 5-18.

Remove a device

To remove RDI targets from the list, highlight the entry to be deleted and click **Remove**.

Reset the RDI target list

Click **Reset list** to reset all entries to the installation defaults.

———— Note —————

When working with the RDI Target List dialog box:

- The Angel debug monitor, Remote_A, is autodetected if you install ADS. This DLL is installed with RealView Debugger during the RVDS installation. You must add it to the list yourself, from its location in the directory:
`install_directory\RDI\Targets\Remote_A\...\win_32-pentium`
- Autodetected targets cannot be removed from the display list.

- If you use **Reset list** to reset list entries, any settings changed, or targets added to the list, since installation are also removed. There is no undo.

Click **Close** to close the RDI Target List dialog box and return to the Connection Control window.

5.3.2 Adding and configuring RDI targets

RDI targets are automatically installed for ARM products such as Multi-ICE. RealView Debugger also includes DLLs to enable you to add an Angel debug target. The debugger provides the RDI Target List dialog box where you can add more RDI targets or configure targets detected automatically.

Adding RDI targets

If you have a third-party RDI component that uses RDI 1.5.1, you can use the RDI Target List dialog box to add it (see *The RDI Target List dialog box* on page 5-15).

To add an RDI target:

1. Display the RDI Target List dialog box, as described in *Displaying the RDI Target List configuration dialog box* on page 5-15.
2. Click **Add DLL...** to display the Select RDI DLL dialog box.
3. Locate the required DLL, then click **Open**. The Create New RDI Target dialog box, shown in Figure 5-7, is displayed, and the **Short Name** and **Description** fields are set to the DLL filename and the path, respectively.

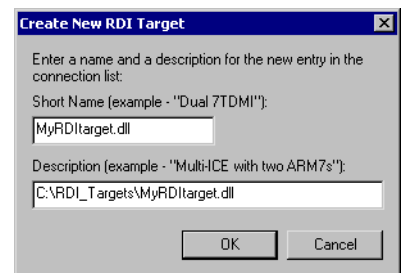


Figure 5-7 Create New RDI Target dialog box

To specify the new target, enter:

Short Name

The target name that is to be shown in the RDI Target List dialog box, the Connection Control window, and in the Connection Properties window, for example CONNECTION=My_RDI_Target.

Description

The target description as shown in the RDI Target List dialog box, the Connection Control window, and in the Connection Properties window, for example Description=ARM JTAG debug interface (parallel port).

Click **OK** to confirm your entries and to add the target to the display list.

4. Change the **Short Name** and **Description** fields as required.
5. Click **OK**. The new target is added to the RDI Target List, and is enabled.

Configuring RDI targets

RealView Debugger does not support connecting to multiple RDI targets at the same time. You can, however, configure your target connections in advance and then connect to each in turn.

Within a single Multi-ICE connection, you can connect to multiple processors provided they are all on the same scan chain. You must have the appropriate RealView Debugger multiprocessor license to make multiple connections.

You can configure your RDI target either:

- from the Connection Control window. Highlight the entry, for example the second-level entry Multi-ICE, and select **Configure Device Info...** from the **RDI Target** menu
- from the RDI Target List dialog box, shown in Figure 5-6 on page 5-15. Highlight the entry in the display list and click **Configure....**

Note

If you cannot see your RDI target in the Connection Control window then you must add it using the RDI Target List dialog box. See *Adding RDI targets* on page 5-17 for details on how to do this.

The rest of this section contains basic configuration information for the following ARM RDI interface software, with details of changes that result from the use of RealView Debugger:

- *Configuring RealView ARMulator ISS* on page 5-19
- *Configuring Remote_A* on page 5-22
- *Configuring ARM Multi-ICE* on page 5-23
- *Configuring ARM Agilent Debug Interface* on page 5-26
- *Configuring ARM RealMonitor* on page 5-26.

These instructions do not replace the original manuals.

5.3.3 Configuring RealView ARMulator ISS

RVISS is the ARM processor simulator supplied with RealView Developer Suite (RVDS). This section describes how to configure the RDI interface to RVISS.

Note

The RDI connection to RVISS in RealView Debugger is deprecated in this release.

RVISS also provides a RealView Connection Broker interface. See *Working with Simulator Broker connections* on page 5-28 for details on configuring this interface.

For the ARMulator RDI target entry, the configuration dialog box is shown in Figure 5-8.

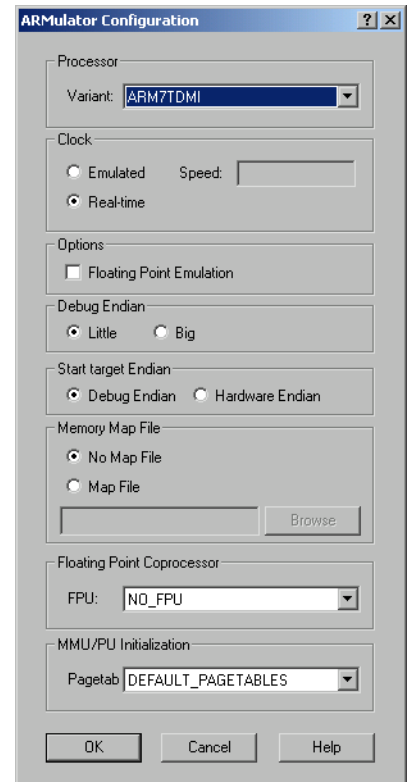


Figure 5-8 ARMulator Configuration dialog box

The ARMulator Configuration dialog box (Figure 5-8 on page 5-19) enables you to examine and change the following settings:

- Processor** Use the drop-down list to specify which ARM processor you want RVISS to simulate, for example, ARM920T or ARM966E-S.
- Clock** Choose between simulating a processor clock running at a speed that you can specify, or executing instructions in real-time by setting this value to 0. You can use units of Hz, KHz, MHz and GHz, for example 50MHz.
- 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.
- Options** Enable this to include a simulation of the *Floating Point Accelerator* (FPA) coprocessor included in the ARM7500FE processor.

Debug Endian

Select the byte order of the modeled system. 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 target Endian option is set to **Debug Endian**.

Start target 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.

The possible combinations of Debug Endian and Start target Endian are shown in Table 5-1.

Table 5-1 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 core start in little-endian mode, and switch to big-endian in the initialization code	Big	Hardware Endian

Memory Map File

Specify a memory map file for use with RVISS. When you connect, the map details are listed with the other connection-related messages.

A map file that is specified in this way is not shown in **Map** tab of the Process Control pane. Therefore, you must edit the map file directly to make changes to the memory map, then disconnect and reconnect. Alternatively, you can use the Memory_block configuration item to specify a memory map to RealView Debugger. See *Configuring a memory map* on page 3-49 for an example explaining how to do this.

———— **Note** —————

Be aware that the MAPFILETOLOAD setting in the RVISS peripherals.amf file overrides the map file you specify in this dialog box.

Floating Point Coprocessor

Use the drop-down list to specify the Vector Floating Point (VFP) coprocessor included with some ARM CPUs. The default is No_FPU.

MMU/PU Initialization

If you are simulating a processor with an active *Memory Management Unit* (MMU), specify DEFAULT_PAGETABLES, otherwise select NO_PAGETABLES. See *ARM Architecture Reference Manual* for more information.

See *RealView ARMulator ISS User Guide* for more information on how these settings apply to RVISS.

5.3.4 Configuring Remote_A

To enable RealView Debugger to communicate with an Angel debug target, use Remote_A, supplied with RealView Debugger.

———— Note ————

Remote_A.dll is installed as part of RealView Developer Suite, you must add it to the RDI Target List yourself, from its location in the directory (see *Adding RDI targets* on page 5-17 for details):

`install_directory\RDI\Targets\Remote_A\...\win_32-pentium`

For the Remote_A RDI target entry, the configuration dialog box is shown in Figure 5-9.

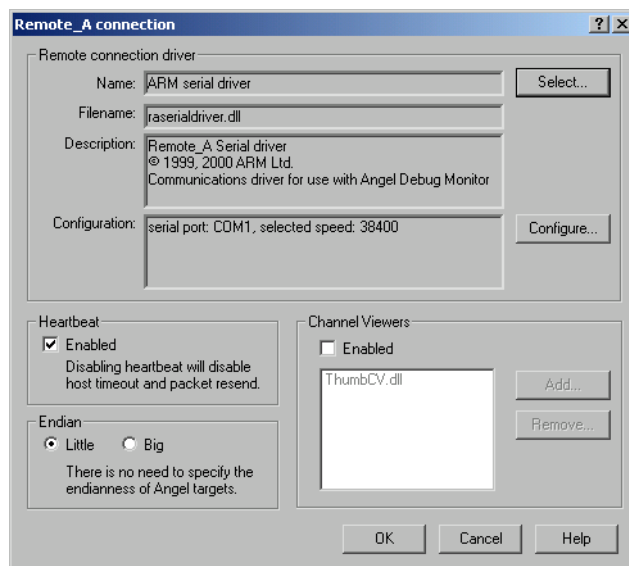


Figure 5-9 Remote_A connection dialog box

The Remote_A connection dialog box enables you to examine and change the following settings:

Remote connection driver

Click **Select...** to see a list of available drivers. This includes Serial, Serial/Parallel, and Ethernet drivers. Select one if you want to use it instead of the current driver.

To change the settings of the currently selected driver, click **Configure...**. This displays the dialog box appropriate to the chosen driver.

Heartbeat Ensures reliable transmission by sending heartbeat messages. Any errors are more easily detected when known messages are expected regularly.

Endian These radio buttons specify that the target is operating in little-endian or big-endian mode.

This setting is only used if you are connected to an EmbeddedICE® Interface Unit.

Channel Viewers

RealView Debugger does not support channel viewers.

If you add Remote_A to the RDI Target List dialog box yourself, you must configure the target before you can connect. When you first display the Remote_A connection dialog box, all settings are empty. Use **Select...** to choose the driver DLL. Configure the other settings as required.

5.3.5 Configuring ARM Multi-ICE

You can use the Multi-ICE interface unit to connect to ARM core-based processors using the RDI Multi-ICE DLL and the Multi-ICE server. For the Multi-ICE RDI target entry, the configuration dialog box is shown in Figure 5-10.

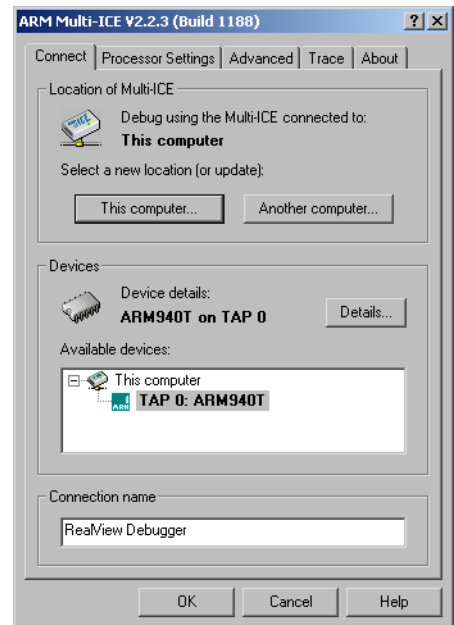


Figure 5-10 Multi-ICE DLL configuration dialog box

The ARM Multi-ICE dialog box, shown in Figure 5-10 on page 5-23, contains the tabs:

Connect This tab contains the **This computer...** and **Another computer...** buttons that enable you to select the Windows workstation that is running the Multi-ICE server, and the Connection name data field that enables you to identify each processor connection.

Processor Settings

This tab contains any processor-specific settings. See *Multi-ICE User Guide* for details of these.

Advanced This tab contains the target endianness and interface settings. Use the radio buttons in the Target Settings group as shown in Table 5-2.

Table 5-2 Multi-ICE Endian settings

Usage	Selection
A target that is always little-endian. This is the default.	Little-endian
A target that is always big-endian.	Big-endian

Disable the Read-ahead cache if you are accessing read-sensitive memory with the debugger.

Trace Use this tab to enable and configure your Trace Capture tool. Where enabled, select the required Trace Capture DLL from the list, or use **Add...** to locate a new DLL.

About Displays information about the version numbers of the Multi-ICE DLL and RealView Debugger.

Configuring multiple targets using Multi-ICE

If you connect to a Multi-ICE server that is configured with more than one ARM processor, the configuration dialog box includes a side-panel showing icons for each of these processors. This side-panel is not available if you connect to a single processor, shown in Figure 5-10 on page 5-23.

Selecting a processor in the side-panel enables you to independently configure properties for that processor using the **Processor Settings** and **Advanced** tabs. For example, you can set the processor setting **Cache clean code address**, differently on two processors:

1. Click **Processor Settings** tab.

2. Select the first processor in the side-panel.
3. Change the value of **Cache clean code address**.
4. Select the second processor in the side-panel.
5. Change the value of **Cache clean code address**.

With RealView Debugger all the available processors are configured into the Connection Control window and you use that window to connect and disconnect from each processor as required.

Connecting to multiple targets using Multi-ICE

It is recommended that you turn off the cache mechanism in Multi-ICE when debugging multiple processors:

1. Select **Target** → **Connect to Target...** to display the Connection Control window.
2. Right-click on the Multi-ICE entry and select **Configure Device Info...** from the context menu.
The Multi-ICE DLL configuration dialog box is displayed.
3. Click the **Advanced** tab.
4. Ensure that the **Start-up with cache enabled** check box is not selected, and click **OK**.

More information about configuring Multi-ICE is given in *Multi-ICE User Guide* and in the online help available from the dialog box.

Note

- Releases of Multi-ICE before 1.4 are not compatible with RealView Debugger.
 - RealView Debugger supports DCC semihosting with Multi-ICE. When this mode is used, the target processor is not stopped while semihosting takes place.
 - RealView Debugger does not support multiple simultaneous connections to Multi-ICE.
-

5.3.6 Configuring ARM Agilent Debug Interface

The *ARM Agilent Debug Interface* (ADI) configuration dialog box enables you to change the following settings:

Network details

The network (Ethernet) address of the Agilent JTAG probe.

JTAG Frequency

The frequency of the IEEE1149.1 **TCK** signal. A frequency of 10MHz is suitable for most ARM processor cores.

Device Configuration

The **Specify Devices...** button displays the Specify Devices dialog box, and enables you to select the processor with which to connect. RealView Debugger does not support multiple simultaneous connections with ARM ADI Version 1.0.

See *ARM Agilent Debug Interface User Guide* for more information.

Note

Agilent Probe support in RVDS is available as a custom installation option.

5.3.7 Configuring ARM RealMonitor

The ARM RealMonitor configuration dialog box enables you to change the following settings:

JTAG Controller

The RDI-compliant JTAG controller DLL, for example your Multi-ICE DLL.

RDI Module Server

This enables you to view special target registers in RealView Debugger:

Use RDI Module Server

Tickled by default, click to disable the RDI Module Server. If disabled, this grays out the second option in this group.

Fetch module information from target

Ticked by default, this provides the module server with information about the target system. This might be embedded in RMTTarget. If you unselect this option then you must provide:

- the target processor you are using
- the target board you are using.

See *ARM RMTTarget Integration Guide* for more information.

Configure Click to configure the JTAG controller you have selected, for example the Multi-ICE server, shown in Figure 5-10 on page 5-23.

See *Multi-ICE User Guide* or the documentation that accompanies your JTAG unit for more details.

See *ARM RMHost User Guide* for more information.

5.4 Working with Simulator Broker connections

RealView Connection Broker, `rvbroker.exe`, handles connections to vehicles that reside on the same workstation as RealView Debugger or any other workstation on your network.

For more details on using RealView Connection Broker see:

- the chapter that describes getting started in *RealView Debugger v1.8 User Guide*.
- Chapter 6 *Working with Remote Targets*.



Operating as RealView Simulator Broker, RealView Connection Broker runs on your local workstation and enables you to access targets on the local workstation. This means that local host simulators are available immediately from the Connection Control window. If you expand the Simulator Broker entry, ready to connect to a local host simulator, RealView Debugger starts RealView Connection Broker in local mode to manage your connections.

The RealView Debugger base product includes support for simulators to emulate the instruction sets of different processors and their supporting architecture. Where you have installed the software, and you have the appropriate licenses, you can use the default board file to make Simulator Broker connections to RealView ARMulator ISS.

This section describes:

- *Connecting to RealView ARMulator ISS*
- *Configuring RealView ARMulator ISS* on page 5-29
- *Configuring a new simulator connection* on page 5-31.

5.4.1 Connecting to RealView ARMulator ISS

RVISS runs on the same host computer as the debugger, and includes facilities for communicating with the debugger.

RVISS is an *Instruction Set Simulator (ISS)*. It simulates the instruction sets and architecture of ARM processors, together with a memory system and peripherals. You can extend it to simulate other peripherals and custom memory systems. See *Configuring a new simulator connection* on page 5-31 for details.

The Simulator Broker connection to RVISS enables the debugger to connect to a target using a network connection. Simulator Broker connections to RVISS support the following features:

- instruction and data trace
- hardware breakpoints
- multiple instances of RVISS.

Note

RVISS is the only way to connect to simulated ARM cores with RealView Debugger on Sun Solaris or Red Hat Linux.

To connect to RVISS:

1. Select **Target** → **Connect to Target...** to display the Connection Control window.
2. Expand the entry Server Connection Broker.
3. Expand the entry localhost Simulator Broker.
4. Double-click on the new_arm entry to connect to the ARM simulator (RVISS).

The RealView Connection Broker interface of RVISS enables you to connect to multiple targets at the same time. For details see the chapter that describes multiprocessor support in *RealView Debugger v1.8 Extensions User Guide*.

5.4.2 Configuring RealView ARMulator ISS

To configure your RVISS connections from the Connection Control window:

1. Right-click on the required entry, new_arm.
2. Select **Configure Device Info...** from the context menu to display the configuration dialog box is shown in Figure 5-11.

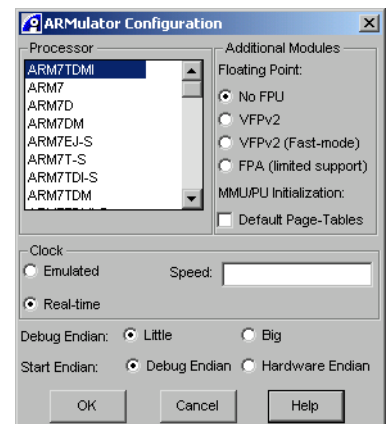


Figure 5-11 ARMulator configuration dialog box

The ARMulator Configuration dialog box enables you to examine and change the following settings:

- Processor** Use the list to specify which ARM processor you want RVISS to simulate.
- The list of processors includes all available variants currently supported on Simulator Broker connections.
- Clock** Choose between simulating a processor clock running at a speed that you can specify, or executing instructions in real-time.
- Specify speed in Hz, for example 50000Hz.
- 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 modeled system. 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 target 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.

The possible combinations of Debug Endian and Start target Endian are shown in Table 5-3.

Table 5-3 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 core start in little-endian mode, and switch to big-endian in the initialization code	Big	Hardware Endian

Floating Point

Use the radio buttons to specify the Vector Floating Point (VFP) coprocessor included with some ARM CPUs. The default is No FPU.

MMU/PU Initialization

If you are simulating a processor with an active *Memory Management Unit* (MMU), specify Default Page-Tables. See *ARM Architecture Reference Manual* for more information.

Be aware of the following features when using this dialog box:

- Some cores are not available when you are connected using RVISS, for example StrongARM™, ARM8 and the simulated *Embedded Trace Macrocell™* (ETM) functionality.
- Use the Additional Modules control group to define *Floating Point Accelerator* (FPA) and *Vector Floating Point* (VFP) options (VFPv1 is no longer available).
- If you choose **Real-time**, do not enter a value for Speed.

5.4.3 Configuring a new simulator connection

Simulator Broker connections to local host simulators are configured with custom settings by default. However, you might want to create new connections to simulate other peripherals and custom memory systems. For example, during the early stages of your development project, you might want to simulate an RTOS-enabled connection to reference a vendor-supplied board file. This enables accurate benchmarking of software targeted at ARM core-based processors although performance is slower than with real hardware.

This example creates a new local host connection using RVISS. It uses the default configuration file for ARM simulators, `default.auc`, created in your RealView Debugger home directory when you installed the base product. This simulates an ARM7TDMI core. It is recommended that you back up this file before starting the example, so that you can restore your original configuration later. If you want to create different simulator connections, for example to simulate an ARM940T core, make copies of this file using appropriate names.

For details on using RealView Connection Broker in remote mode, and how to create a connection to a remote host, see Chapter 6 *Working with Remote Targets*.

The example is split into the following sections:

- *Creating the configuration file*
- *Creating the new connection*
- *Viewing details of the new connection* on page 5-35.

Creating the configuration file

To create a configuration for the new connection:

1. Locate the file `default.auc` in the directory that RealView Debugger is using for your session.
See *The home directory* on page 1-14 for more information about the location of your debugger home directory.
2. Copy the file.
3. Rename the copy, for example `Tst_SIM_Broker_ARM940T.auc`.

It is not necessary but it is recommended that you create a new `.auc` file when you create a new simulator connection.

Creating the new connection

To define the new connection:

1. Select **Target** → **Connection Properties...** to display the Connection Properties window.
2. Right-click on the `...\rvdebug.brd` entry, in the left pane, to display the context menu.
3. Select **Make New Group...** from the context menu, to display the Group Type/Name selector dialog box, shown in Figure 5-12 on page 5-33.

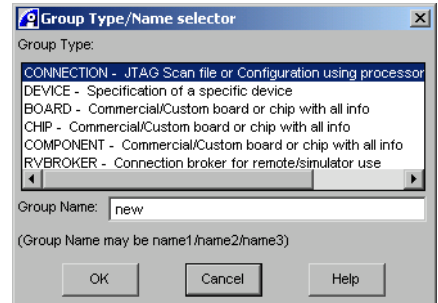


Figure 5-12 Specifying a new CONNECTION group

4. Define the new group:
 - Leave the type of the new entry unchanged as CONNECTION.
 - 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 example New_Test_SIM.
This can be a descriptive name or the name of the .auc file that you are going to select, but without the .auc extension.
5. Click **OK** to confirm your settings and to close the Group Type/Name selector dialog box.
The new entry appears in the left pane of the Connection Properties window. It is automatically selected, and its details are displayed in the right pane. These details are the default for a new CONNECTION and you must change at least the Connect_with/Manufacturer, the Configuration filename and target Description. These changes are described in the following steps.
6. In the right pane of the Connection Properties window, right-click on the Configuration entry and select **Edit as Filename...** from the context menu.
The Enter New Filename dialog box is displayed to enable you to locate the required .auc file, for example default.auc or Tst_SIM_Broker_ARM940T.auc, the processor-specific file created in *Creating the configuration file* on page 5-32.
If RealView Debugger cannot locate the specified file, you are offered the option to create the copy now or to define a new file for this connection.
7. Click **Save** to confirm your entries and to close the Enter New Filename dialog box.
The new pathname is displayed in the right pane.

8. In the right pane of the Connection Properties window, right-click on the new *Configuration entry and select **Edit Configuration-File Contents...** from the context menu.

Use the configuration dialog box, shown in Figure 5-11 on page 5-29, to configure your new RVISS connection (see *Configuring RealView ARMulator ISS* on page 5-29 for details).

In this example, specify that the ARM940T core is simulated.

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

9. In the right pane of the Connection Properties window, right-click on the Description field, and select **Edit Value** from the context menu.

Type ARM940T + RVISS to new test hardware in the entry area and press Enter.

This is the description displayed in the Connection Control window and Connection Properties window to identify the new target.

10. In the right pane of the Connection Properties window, right-click on the Connect_with entry and select **Explore** from the context menu.

11. In the right pane of the Connection Properties window, right-click on the Manufacturer entry and select the required connection type from the context menu, that is **ARM-A-SW**.

If you do not specify this setting, the new connection appears in the Connection Control window but, when you try to connect, RealView Debugger prompts for the connection type.

12. If you want to configure your target, for example by defining ARM configuration settings such as semihosting, make these changes now.

Expand the following entries in turn:

- a. CONNECTION=New_Test_SIM
- b. Advanced_Information
- c. Default
- d. ARM_config

13. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

Your new simulator is now displayed in the Connection Control window where you can connect in the usual way, shown in Figure 5-13 on page 5-35.

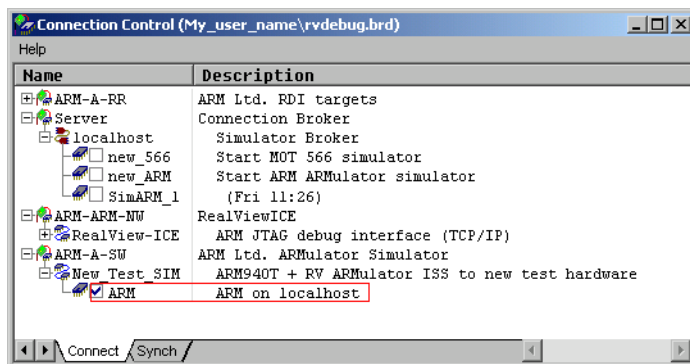


Figure 5-13 Connecting to a new simulator

Viewing details of the new connection

When you connect to the new simulator, the Output pane shows details about the connection to RVISS, shown in Figure 5-14.

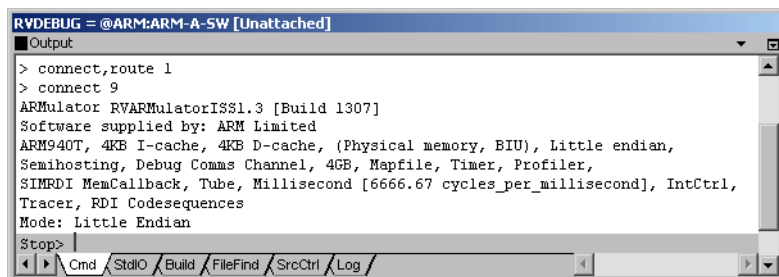


Figure 5-14 Simulator details in the Output pane

Here you can see that the RVISS is simulating the ARM940T core specified in the new .auc file.

Connecting to the new simulator also creates a new entry in the Connection Control window under localhost Simulator Broker, shown in Figure 5-13. If you try to click on this entry to connect, this fails. The Output pane displays details of the error, for example:

Error: Failed on remote SIM/EMU/EVM. In use by my_user_name

This is because Simulator Broker views this as a remote connection. When you disconnect from the new connection, this entry also disappears.

5.5 Working with JTAG files

On-Chip Debugging (OCD) based emulators use JTAG files to define the devices on the JTAG scan chain and their order. This information might be supplied by the manufacturer or might be configured after installation. RealView Debugger detects the JTAG files and uses them to complete the configuration settings. This enables the debugger to access emulator targets on the local host for each supported processor.

Each emulation is specified in a .jtg file named after the supported processor, for example arm.jtg. By default, supported .jtg files are stored in the default settings directory \etc when you install RealView Debugger.

5.5.1 Editing JTAG files

To access the .jtg file editor:

1. Start RealView Debugger without connecting to a target.
2. Right-click on the required connection in the Connection Control window to display the context menu.
3. Select **Connection Properties...** to display the Connection Properties window.
4. Right-click on the *Configuration entry in the right pane, for the required .jtg file, and select **Edit Configuration-File Contents...** from the context menu.

This displays the Device JTAG-File Editor dialog box. An example is shown in Figure 5-15 on page 5-37.

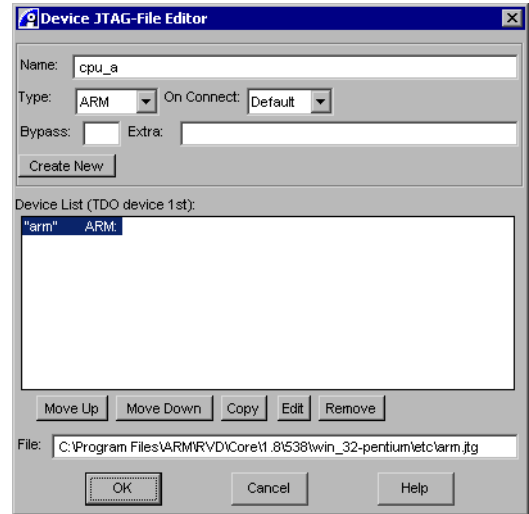


Figure 5-15 Device JTAG-File Editor dialog box

Use this dialog box to amend the current device list, or to add new devices to the stored scan chain. The dialog box contains the following controls:

Name: Enter the name of a new device to be added to the configuration. This name identifies the new processor in the Connection Control window.

Type: Click on the down arrow to see a list of predefined processor types, for example ARM.

———— **Note** ————

The choices displayed in the drop-down list are determined by the vehicle defined for this connection, as set by the Connect_with/Manufacturer (see Appendix A *Configuration Settings Reference*).

On Connect:

Click on the down arrow to see a list of processor reset options when making a connection to the chosen device, for example Reset.

Bypass: This is used to ignore a particular TAP controller, preventing the debugger from connecting to it. Insert in the text field the number of bits in the *Scan Chain Select Register* (SCSR) for the bypassed device. For ARM CPUs this depends on CPU model, but is normally either 4 or 5.

The bypass option is not available for some vehicle types.

Extra: If your configuration file contains non-generic, target-specific entries, use this field to specify these parameters.

Create New When you have entered the device details, click here to add the new device to the top of the display list.

Device List This shows all the devices currently configured in the JTAG file. As you create a new device it is added to the top of the list.

The list is ordered as shown in Figure 5-16. That is, the top list entry corresponds to the device that has its *Test Data In* (TDI) connected to the host interface *Test Data Out* (TDO) pin and the bottom list entry has its TDO connected to the TDI pin of the host interface.

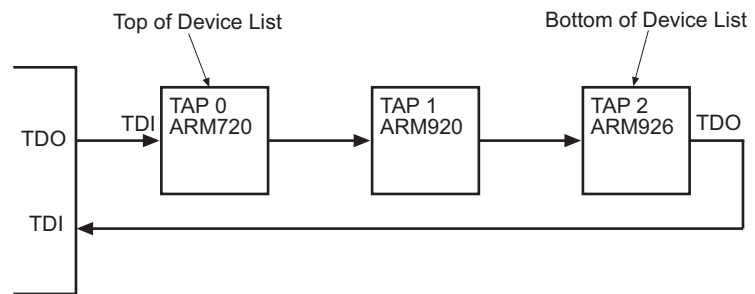


Figure 5-16 JTAG chain ordering

Move Up Use this to change the order of entries in the display list. Highlight an entry and click **Move Up** to move it up towards the top of the list by one place.

Move Down Use this to change the order of entries in the display list. Highlight an entry and click **Move Down** to move it down towards the bottom of the list by one place.

Copy Click to copy an existing entry and add it to the device list.

Edit Click to edit a device definition and so rename it in the device list. Editing a device also means that you can change the text given in the Description column in the Connection Control window.

There is a limit on the number of characters that can be displayed in the Description column.

Remove Click to remove a highlighted entry from the device list.

File: Use this data field to specify the full pathname to the .jtg file that defines the device list.

- OK** Click to confirm your entries and so update the specified .jtg file. This closes the Device JTAG-File Editor dialog box.
- Cancel** Click to close the Device JTAG-File Editor dialog box without making any changes to the device list.
- Help** Click to get online help text about this dialog box.

Viewing changes

You must save the amended board file entries to update the contents of the Connection Control window. After confirming entries in the Device JTAG-File Editor dialog box, select **File** → **Save and Close** to close the Connection Properties window.

If the chosen entry was expanded in the Connection Control window, saving the updated board file settings collapses the second-level entry.

Expand the entry to see the new or updated devices available for connection.

Chapter 6

Working with Remote Targets

This chapter describes how RealView® Debugger handles remote connections. It includes information on the board file group RVBROKER, and explains how connections to different targets are configured to enable network access. It contains the following sections:

- *Accessing remote targets with RealView ICE and Multi-ICE* on page 6-2
- *Working with RealView Network Broker* on page 6-3
- *Connecting to a remote host simulator* on page 6-6
- *Using a hosts file* on page 6-11
- *Disconnecting remote connections* on page 6-17.

6.1 Accessing remote targets with RealView ICE and Multi-ICE

RealView Debugger enables you to debug software running on targets connected to remote hosts, that is where hardware is connected to workstations accessed across the network. To do this, you must use the appropriate configuration dialog to locate the target:

- If you are using RealView ICE, enter the address of the remote unit in the RVConfig dialog box. See *Configuring a RealView ICE interface unit* on page 5-10 for details.

If you do not know these details, use the RealView ICE Config IP software to scan for all available units on your network. See the chapter that describes configuring RealView ICE networking in *RealView ICE User Guide* for details on how to do this.

- If you are using Multi-ICE® on Windows, click **Another computer...** on the Multi-ICE DLL configuration dialog box to locate all workstations on your network that are running the Multi-ICE server software. See *Configuring ARM Multi-ICE* on page 5-23 for details.

6.2 Working with RealView Network Broker

RealView Connection Broker, `rvbroker.exe`, handles connections to vehicles that reside on the same workstation as RealView Debugger or any other workstation on your network.

For more details on using RealView Connection Broker see:

- the chapter that describes getting started in *RealView Debugger v1.8 User Guide*.
- Chapter 5 *Configuring Custom Connections*.



Operating as RealView Network Broker, this runs on a remote workstation and makes specified targets on that workstation available to other workstations connected to the same network.

This section describes how to work with remote connection entries. It contains the following sections:

- *Accessing remote targets using RealView Network Broker*
- *Specifying a remote host simulator*
- *Using the examples* on page 6-4.

6.2.1 Accessing remote targets using RealView Network Broker

RealView Network Broker enables you to connect to remote simulators and OCD-based emulators. To do this, target configuration files are held on the local workstation and pushed across the network connection to the remote host. RealView Network Broker must be running on the remote workstation to provide access to the local simulators and emulators.

6.2.2 Specifying a remote host simulator

To access a remote host simulator or emulator using RealView Network Broker you must use the Connection Properties window to define the location of the remote workstation in your connection entries and target configuration settings in your board file, shown in Figure 6-1 on page 6-4.

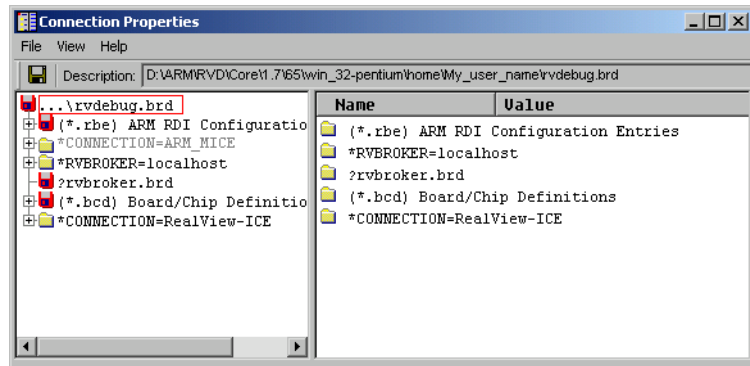


Figure 6-1 Remote connections in the Connection Properties window

There are two ways to do this:

RVBROKER

This entry specifies the location of a host that supports simulators. The Connection Control window uses this entry to define remote targets. See *Connecting to a remote host simulator* on page 6-6 for details.

rvbroker.brd

This entry enables the inclusion of a special hosts file containing a list of all network nodes that can run, or are running, RealView Network Broker. See *Using a hosts file* on page 6-11 for details.

You can combine both types of entry in the same board file.

6.2.3 Using the examples

In the examples in the rest of this chapter, you are changing your board file. This is stored in your RealView Debugger home directory. Target configuration files might also be stored in this directory, for example .rvc files or .cnf 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. For details see:

- *Configuration files* on page 1-12 for instructions on making backups of your configuration
- *Restoring your .brd file* on page 3-62 for instructions on restoring a default configuration

- *Troubleshooting* on page 3-62 for instructions on recovering from an incorrectly configured debugger home directory, whether or not you have a backup.

If you are using a hosts file this might be stored in the default settings directory (\etc) or in your home directory. See *Using a hosts file* on page 6-11 for details.

Note

When you are following these examples, do not configure the board file when the debugger is connected to a target.

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 you are familiar with the procedures described in this help topic.

6.3 Connecting to a remote host simulator

To connect to a remote host simulator, you must configure a new target description in the board file stored on the local workstation. This enables you to access a remote host simulator across the network.

First, you must start the remote connection broker, that is RealView Network Broker, on the remote workstation. See the chapter that describes getting started in *RealView Debugger v1.8 User Guide* for details on how to do this.

Note

RealView Debugger does not have to be running on the remote workstation. Only RealView Network Broker is required to make the simulators and emulators visible across the network.

The rest of this section describes how to configure and connect to a remote simulator. It contains the following sections:

- *Configuring the new target*
- *Connecting to the new target* on page 6-9
- *Viewing remote simulator connections* on page 6-10.

6.3.1 Configuring the new target

To configure the new target on the local workstation:

1. Start RealView Debugger without connecting to a target.
2. Select **Target** → **Connection Properties...** to display the Connection Properties window.
3. Right-click on the RVBROKER=localhost entry and select **Make Copy...** from the context menu.
4. Use the Group Type/Name selector dialog box, shown in Figure 6-2 on page 6-7, to name the new connection, for example Remote_Debug.

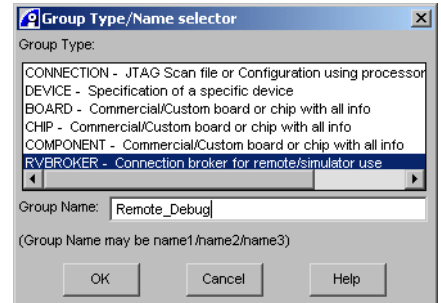


Figure 6-2 Creating a new connection

5. Click **OK**. The new entry, RVBROKER=Remote_Debug, is added to the List of Entries in the Connection Properties window, shown in Figure 6-3.

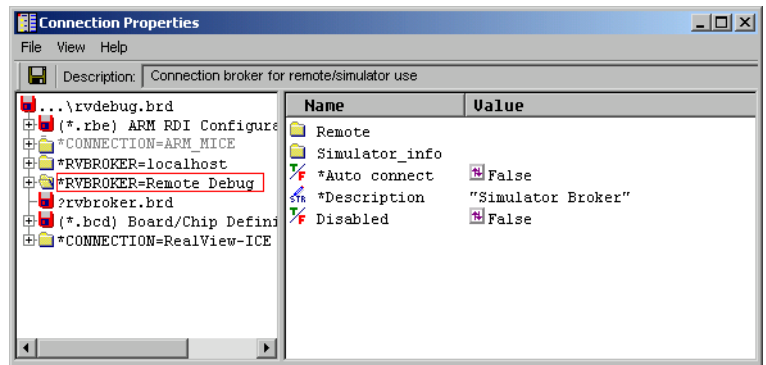


Figure 6-3 Displaying the new Remote_Debug connection properties

6. Right-click on Remote in the right pane, shown in Figure 6-3, and select **Explore** from the context menu.
7. Right-click on Hostname in the right pane to display the context menu.
8. Select **Edit Value** and enter the hostname of the remote workstation, for example Hostname=TESTPC1, shown in Figure 6-4 on page 6-8.

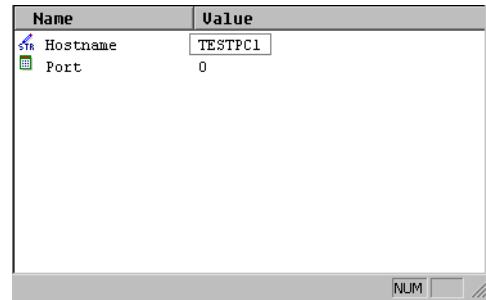


Figure 6-4 Specifying a remote host

Specify the full domain name where necessary, for example
 Hostname=testpc.ournet.tst.com. You can use the IP address, for example
 Hostname=123.145.6.170.

It is not necessary to specify the host name of the remote workstation if the RVBROKER group name is also the host name of the remote workstation on your network, for example RVBROKER=TESTPC1. Where the Hostname entry is filled in, this is used and the group name is irrelevant.

9. Select *RVBROKER=Remote_Debug in the left pane.
10. Right-click on Description in the right pane, shown in Figure 6-3 on page 6-7, to display the context menu. Select **Edit Value** and enter a short description for the new connection, for example Remote sim on network TESTPC1.
11. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

The new target is added to the Connection Properties window (see Figure 6-5 on page 6-9).

You can create multiple RVBROKER groups if required. Use unique names to identify the different remote hosts.

6.3.2 Connecting to the new target

To connect to the remote target.

1. Display the Connection Control window to see the new remote connection, shown in Figure 6-5.

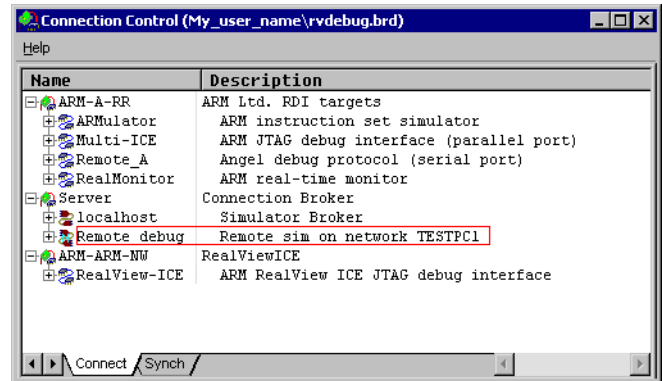


Figure 6-5 Remote target in the Connection Control window

2. Click on the icon next to the new entry, Remote_Debug, to expand it.
This shows the available simulators made visible by the remote connection broker.
This entry is empty if there are no remote simulators available on the specified workstation.
3. Click on the entry under the new Connection Broker entry to connect to the target. RealView Debugger retrieves information specific to the target.

6.3.3 Viewing remote simulator connections

When you connect to a remote target, a new entry appears in the Connection Control window, for example RSimarm_1, as shown in Figure 6-6.

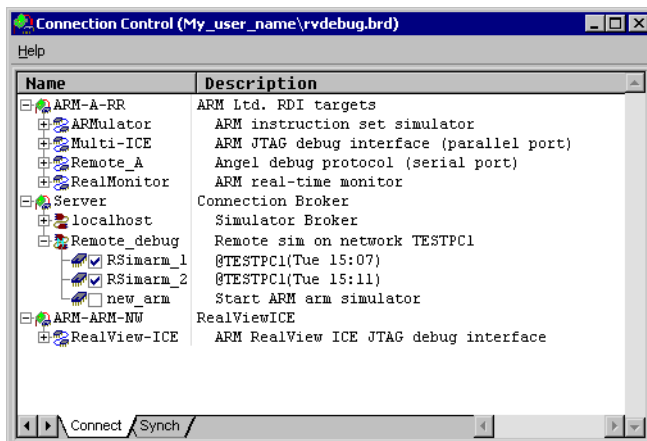


Figure 6-6 Remote connections in the Connection Control window

Here we can see two connections to remote RealView ARMulator® ISS targets. In this example, the IP address has been used to specify the host workstation. Other remote connections are visible depending on the type of installation at the remote workstation. Where you have the appropriate licenses, you can connect to these simulators in the usual way.

— Note —

Ending a debugging session on the remote workstation, and closing down RealView Debugger, does not terminate the remote connection broker. RealView Network Broker must be shut down explicitly when it is no longer required.

6.4 Using a hosts file

The `rvbroker.brd` entry in the board file enables RealView Debugger to include a special hosts file containing a list of all network nodes that can run or are running RealView Connection Broker. This enables multiple users to access simulators, emulators, and EVMs anywhere on the network by sharing this single file.

If you are using a hosts file, this is the same as adding `RVBROKER=` entries to your board file to manage remote connections across your network. However, the connection details are contained in the `rvbroker.brd` file and not in the `rvdebug.brd` file.

When RealView Debugger reads a board file, for example `rvdebug.brd`, it searches for an `rvbroker.brd` file to use. The debugger searches:

1. in the default settings directory, `install_directory\RVD\Core\...\etc`
2. your home directory
3. the current working directory.

Where you are using a local hosts file, it must be installed in your home directory. Install the hosts file in the default settings directory (`\etc`) for shared access. The search order defines the precedence when using multiple files in different locations.

This section describes how to create and use a hosts file. It includes the following subsections:

- *Creating a local hosts file*
- *Configuring a local hosts file* on page 6-12
- *Connecting using a local hosts file* on page 6-14
- *Using multiple connections in a local hosts file* on page 6-14
- *Using a network hosts file* on page 6-15.

6.4.1 Creating a local hosts file

To create a hosts file:

1. Use a text editor to create a file named `rvbroker.brd` containing a single line:
`[RVBROKER=Remote_Hosts]`
 Save this file in your home directory, for example
`home\my_user_name\rvbroker.brd`.

6.4.2 Configuring a local hosts file

To configure a hosts file:

1. Start RealView Debugger without connecting to a target.
2. Display the Connection Control window where the new connection entry is visible, shown in Figure 6-7.

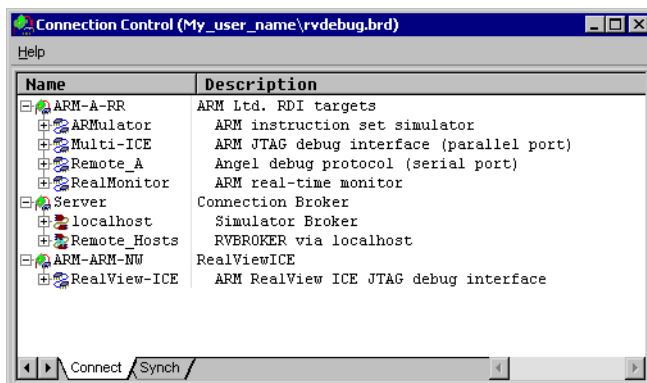


Figure 6-7 Hosts file in the Connection Control window

RealView Debugger locates the `rvbroker.brd` file in your home directory, or in `\etc`, and includes the configuration details in the board file, ready for editing.

3. Right-click on the `Remote_Hosts` entry and select **Connection Properties...** from the context menu to display the Connection Properties window, shown in Figure 6-8.

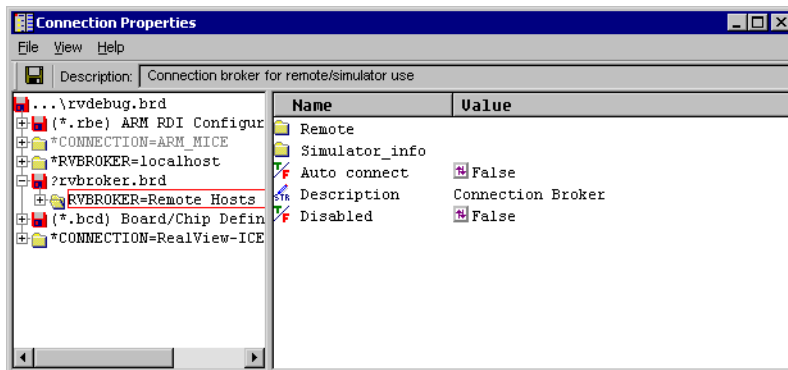


Figure 6-8 Configuring a hosts file

4. Edit the RVBROKER=Remote_Hosts group as follows:
 - a. Edit the Description settings value in the right pane, to identify the remote connection, for example Remote to test board on TESTPC1.
 - b. Ensure that the Disabled settings value in the right pane is set to False, the default.
 - c. Expand the Remote group, in the right pane, to display the contents. Edit the Remote settings values page to enter the network address of the remote workstation, for example Hostname=TESTPC1.ournet.tst.com, shown in the example in Figure 6-4 on page 6-8.
5. Select **File** → **Save and Close** to save your changes and close the Connection Properties window.

The hosts file, `rvbroker.brd`, is automatically updated with the new configuration information. You can view the amended file using a text editor, for example:

```
[RVBROKER=Remote_Hosts]
remote.hostname="TESTPC1.ournet.tst.com"
description="Remote to test board on TESTPC1"
```

You can also use the network IP address to specify the remote target, for example:

```
[RVBROKER=Remote_Hosts]
remote.hostname="123.145.6.170"
description="Remote to test board on TESTPC1"
```

Custom entries in the hosts file

Default board file entries are not shown in the hosts file. For example, the setting `Disabled=False` (shown in Figure 6-8 on page 6-12) is the default setting for the new RVBROKER entry and so there is no corresponding entry in the hosts file. However, if you edit this setting so that it becomes `*Disabled=False`, an entry is created in the hosts file, for example:

```
[RVBROKER=Remote_Hosts]
remote.hostname="123.145.6.170"
description="Remote to test board on TESTPC1"
disabled=False
```

You can specify entries in the hosts file even where these are the default settings for the new connection.

6.4.3 Connecting using a local hosts file

To connect to a remote target using the hosts file:

1. Display the Connection Control window to see the new remote connection. An icon identifies that this connection uses a hosts file to store configuration details, shown in Figure 6-9.

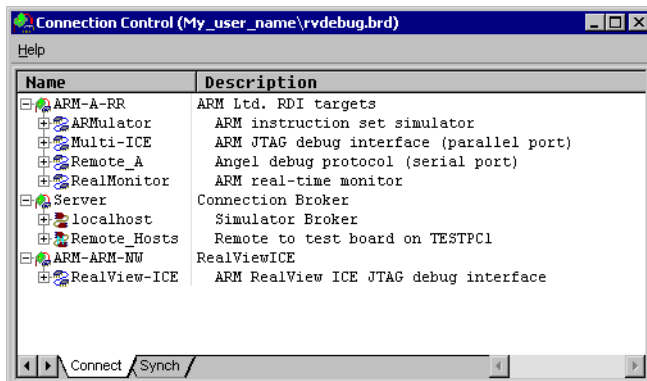


Figure 6-9 Remote target in the Connection Control window

2. Expand the new Remote_Hosts entry to view the available target connections made visible from the remote connection broker.
3. Click on the new entry to make the remote connection.

6.4.4 Using multiple connections in a local hosts file

You can copy the RVBROKER=Remote_Hosts entry to make a list of hosts entries. Configure each one to point to a specified network node.

The hosts file is automatically updated with the new configuration information. You can view the amended file using a text editor, for example:

```
[RVBROKER=Remote_Hosts]
remote.hostname="123.145.6.170"
description="Remote to test board on TESTPC1"
disabled=False
[RVBROKER=Remote_Hosts_2]
remote.hostname="TESTPC2.ournet.tst.com"
description="Remote to test board on TESTPC2"
disabled=False
```

Each entry in the hosts file appears as an available connection in the Connection Control window, shown in Figure 6-10 on page 6-15.

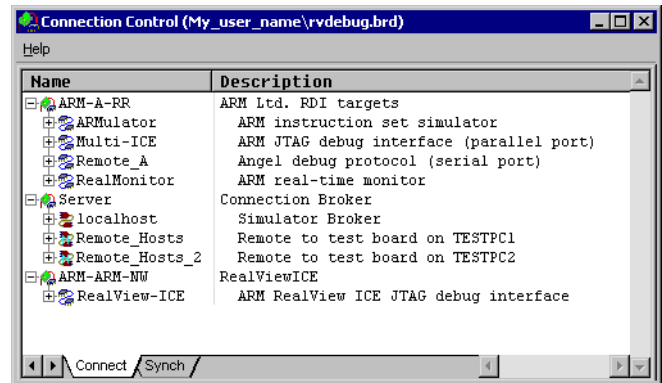


Figure 6-10 Multiple remote targets in the Connection Control window

6.4.5 Using a network hosts file

If required, a hosts file can be installed in a network location, for shared access by other network users. You create the network hosts file in the same way as a local hosts file and save it in a shared location on your network, for example

H:\ournet\devel\rvd\shared\rvbroker.brd. The rvbroker.brd entry in your board file is edited to specify the full network pathname.

To configure a network hosts file:

1. Use a text editor to create a network hosts file named rvbroker.brd containing:

```
[RVBROKER=Network_Hosts]
remote.hostname="TESTPC3.ournet.tst.com"
description="Remote to shared network test board on TESTPC3"
disabled=False
[RVBROKER=Network_Hosts_2]
remote.hostname="TESTPC4.ournet.tst.com"
description="Remote to shared network test EVM on TESTPC4"
disabled=False
```

Save this file in your shared network location, for example

H:\ournet\devel\rvd\shared\rvbroker.brd.

2. Start RealView Debugger without connecting to a target.
3. RealView Debugger locates the rvbroker.brd file in your home directory, or in \etc, and includes the configuration details in the Connection Control window, ready for making a connection, shown in Figure 6-10.
4. Select **Target** → **Connection Properties...** to display the Connection Properties window.

5. To include the connection details given in the network hosts file, you must add a hot link from the board file:
 - a. Right-click on the top-most entry `...\rvdebug.brd`.
 - b. Select **Add Hot Link to a File...** from the context menu.
 - c. Click **OK** to accept the warning message and continue.
 - d. Locate the network hosts file in the Select file to link to dialog box.
 - e. Click Open to confirm your choice.
6. Select **File** → **Save and Close** to save the new settings and close the Connection Properties window.

The board file is automatically updated to include the specified network hosts file and the Connection Control window displays the remote hosts as specified in the local hosts file and the remote network hosts file, shown in Figure 6-11.

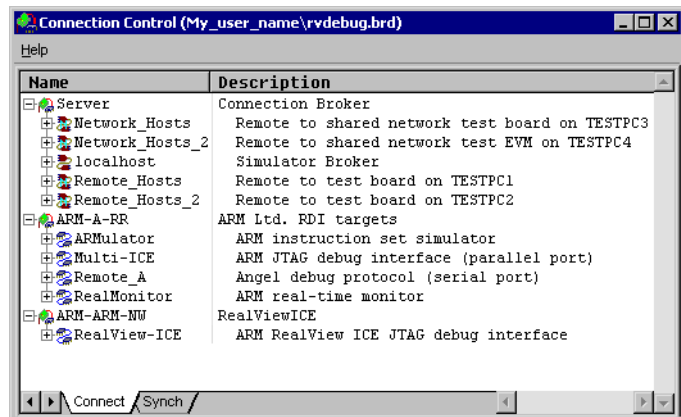


Figure 6-11 Network hosts file in the Connection Control window

Now other users can access the shared hosts file on your network by amending their board files in the same way.

6.5 Disconnecting remote connections

You can use the CLI command DISCONNECT to disconnect from a remote connection using either the connection id or the target name, for example:

```
disconnect 9
```

```
disconnect @RSimarm_12
```

where the name RSimarm_12 identifies the remote target in the Connection Control window.

See the description of the DISCONNECT command in *RealView Debugger v1.8 Command Line Reference Guide* for details on disconnecting this way.

Appendix A

Configuration Settings Reference

This appendix contains reference details about board file entries that define target configurations and custom connections. It contains the following sections:

- *About this appendix* on page A-2
- *Target configuration and connections* on page A-3
- *Generic groups and settings* on page A-5
- *Target configuration reference* on page A-8
- *Custom connection reference* on page A-27.

A.1 About this appendix

You configure the way that RealView® Debugger connects to, and interacts with, your debug target using a board file. The default board file, called `rvdebug.brd`, is set up when you first install RealView Debugger.

The contents of the Connection Control window are defined by elements of the board file. To change target configuration, or to add new connections, you change the entries in the board file and this modifies the elements displayed in the Connection Control window. RealView Debugger provides a GUI interface, the Connection Properties window, to make these changes. See *Connection Properties window* on page 1-5 for details.

To follow this appendix, you must display the Connection Properties window, either:

- Select **Target → Connection Properties....**
- Select an entry in the Connection Control window. Right-click and select **Connection Properties...** from the context menu.

Use this method to open the Connection Control window at the parent group for the chosen connection, target or processor.

This appendix describes the settings and groups of settings that appear in the board file. It 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 target configuration files of your own, your window looks different.

A.2 Target configuration and connections

RealView Debugger makes a distinction between target configuration, and how a target is accessed, target connection. Both are configured using different types of entry in the board file.

A.2.1 Types of entry

A CONNECTION group is normally used to specify connection details, how the target is accessed. This includes the nature and addresses of the hardware interface, for example the port name of the JTAG interface that is connected to the target. This information is described in the Connect_with block of the board file and in the file associated with the Configuration setting.

Note

Occasionally, you can use a DEVICE group in place of a CONNECTION group.

Within a specific CONNECTION, one or more BoardChip_name entries are used to associate the connection with a BOARD, CHIP, or COMPONENT description that defines peripherals and memory maps.

It is recommended that the descriptions of the target are only defined in BOARD or CHIP definitions, and that these descriptions are stored in .bcd files. For example, the definition of the registers and peripherals of the ARM® Integrator™/AP motherboard is stored in the file AP.bcd. Figure A-1 shows how these groups are linked.

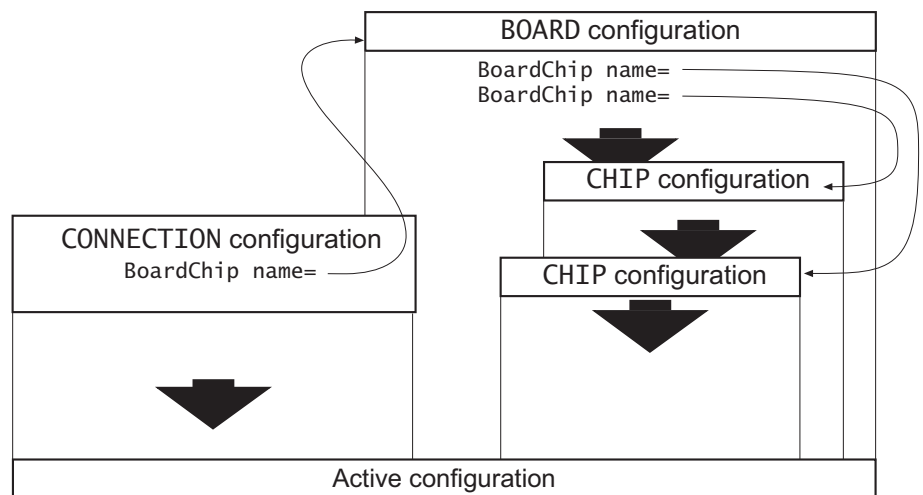


Figure A-1 How connections, boards, and chips fit together

Therefore, the board file consists of the following types of entry:

- Connection entries using the following groups:
 - CONNECTION or DEVICE
 - RVBROKER.
- Configuration entries using the following groups:
 - BOARD
 - CHIP
 - COMPONENT.

This appendix describes the groups and individual settings that appear in the board file. It assumes that you are familiar with the contents of the Connection Properties window as described in *Connection Properties window* on page 1-5.

A.3 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.

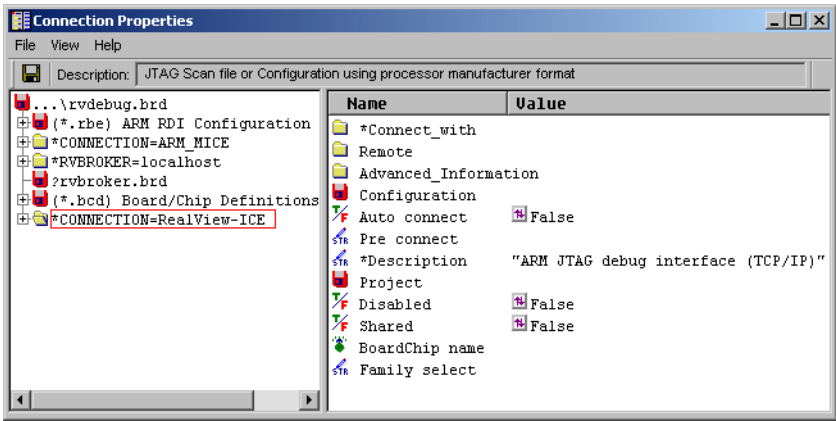


Figure A-2 Viewing generic settings

These settings are found in CONNECTION groups. In some cases, some or all of these settings are also found in other groups, for example BOARD=AP in the AP.bcd entry. This section describes these generic groups and settings:

Connect_with	Specifies the vehicle making the connection. It includes the settings values:
Manufacturer	The name and the type of the connection. Right-click to see a list of available connection types, but you also require an appropriate license and the hardware to use them.
IOdevice	This field contains additional information about the target hardware. Not available in the current release of RealView Debugger.
Speed	You can set the emulation speed for some emulators. Not available in the current release of RealView Debugger.

Remote	This specifies how remote connections are configured. It includes the settings values:
Hostname	The hostname, or IP address, of the remote workstation.
Port	Specifies that a non-default TCP/IP port is used to make the connection to the remote workstation.
Advanced_Information	
Provides ETV target configuration information about the debug target.	
This feature is described in detail in <i>The Advanced_Information block</i> on page A-8.	
Configuration	Specifies a named configuration file. The default is to search for the .jtg file with the same name as this group (after the = sign). The name in this field does not have to be the same as the name of this group. If the file specified here is a full pathname, then only that location is used. Otherwise, it is searched for. When working with RealView ICE targets, .jtg files are replaced by .rvc configuration files. When working with RDI targets, .jtg files are replaced by .cnf configuration files. When working with custom simulator connections, .jtg files are replaced by .auc configuration files.
Aux_Configuration	
Not used in this release.	
Auto_connect	If set to True, this expands the list of devices on the debug target. The list of devices comes from the .jtg file specified by the BOARD or CHIP group. The list of devices is shown in the Connection Control window. You can then connect to the devices with a single click.
Pre_connect	Forces an order for device connection. When you connect to a device within the .jtg file, this ensures that one or more specific devices are connected first regardless of which device is selected for connection. This enables pre-setup of the specific devices to guarantee correct operation, such as initializations. You can specify the device(s) to connect to first by name, by processor name, or by processor type, as used in .cnf files that contain target configuration settings.

Description	Description of what board, processor, or emulator this is for.							
Project	Opens one or more projects automatically when connecting to this board. If more than one device is in the scan chain and they are the same processor type, you must set the <code>Specific_device</code> field of the project to bind the project to the correct device(s). If the devices are different processor types, this is not necessary. See <i>RealView Debugger v1.8 Project Management User Guide</i> for details on working with projects in RealView Debugger.							
Disabled	Disables this entry so that it is grayed out in the Connection Control window.							
Shared	Enables the sharing of target configurations for remote connections.							
BoardChip_name	Refers to the BOARD, CHIP, or COMPONENT group this connection is derived from, using its name. 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.							
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: <table><tr><td><code>name=family_name</code></td><td>This enables you to specify the name of the device. Use either the name defined in the .jtg file or the processor name. This is used when multiple chips are housed on the same target but from different families.</td></tr><tr><td><code>family_name</code></td><td>Choose from the preconfigured list.</td></tr><tr><td><code>silicon_id</code></td><td>Can be expressed as <code>num.num.num.num</code>, for example 15.255.15.15, or as a value, for example 41029401.</td></tr></table>		<code>name=family_name</code>	This enables you to specify the name of the device. Use either the name defined in the .jtg file or the processor name. This is used when multiple chips are housed on the same target but from different families.	<code>family_name</code>	Choose from the preconfigured list.	<code>silicon_id</code>	Can be expressed as <code>num.num.num.num</code> , for example 15.255.15.15, or as a value, for example 41029401.
<code>name=family_name</code>	This enables you to specify the name of the device. Use either the name defined in the .jtg file or the processor name. This is used when multiple chips are housed on the same target but from different families.							
<code>family_name</code>	Choose from the preconfigured list.							
<code>silicon_id</code>	Can be expressed as <code>num.num.num.num</code> , for example 15.255.15.15, or as a value, for example 41029401.							

For information about connection configuration, including the CONNECTION and DEVICE entries, see *Custom connection reference* on page A-27.

A.4 Target configuration reference

This section describes in detail the target configuration entries supported by RealView Debugger. It contains the following sections:

- *The Advanced_Information block*
- *BOARD, CHIP, and COMPONENT groups* on page A-25.

A.4.1 The Advanced_Information block

The Advanced_Information block enables you to provide ETV information about the debug target, for example extended memory mapping, mapped registers, and peripherals. Because more than one device might be present, and each might have different details, you can create more than one Advanced_Information block. The base group is called Default and is used if you provide no other information. Entry names can be the same as processor or device names in configuration files, for example in the .rvc file, or .jtg file, for the connection. These apply more specifically to matching devices.

Advanced information settings can be nested so that one might refer to another, which in turn might refer to another. These references cause the information to be concatenated. References are made to board and chip definitions.

The Default group in an Advanced_Information block contains:

- *Application_Load* on page A-9
- *Memory_block* on page A-9
- *Map_rule* on page A-13
- *Register_enum* on page A-14
- *Register* on page A-14
- *Concat_Register* on page A-16
- *Peripherals* on page A-17
- *Register_Window* on page A-18
- *ARM_config* on page A-18
- *Logic_Analyzer* on page A-20
- *Cross_trigger* on page A-21
- *RTOS* on page A-21
- *Pre_connect* on page A-22
- *Commands* on page A-23
- *Connect_mode* on page A-23
- *Disconnect_mode* on page A-24
- *Id_chip* on page A-25
- *Id_match* on page A-25

- *Chip_name* on page A-25
- *Endianness* on page A-25.

It is recommended that you copy the Default group to create your own, named, groups. Do not delete the Default group so that it is available for future configuration tasks.

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.

The Application_Load group contains the following settings:

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-0 return aborts the load. In all cases, the output of the command is shown in the **Log** tab in the Output pane.

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.

Memory_block

Entries in the Memory_block group are used to build up fixed, enabled, or based memory regions. A fixed block is one that refers to memory that is always enabled, always at the same place, and always the same size. An enabled memory block is enabled or disabled

by a register value (see *Map_rule* on page A-13). A based memory block has its start or length adjusted or set by a register value. This value might be added to an offset or might itself be the required value.

This contains 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 is internal to the processor core 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 processors.

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 destroys contents or the contents change in response to external events.

Shared This field indicates if the memory is shared with other processors. If it is, it also indicates if directly shared (accessed directly using the bus) or indirectly shared (using the host workstation port of this processor). If this is set, this field indicates what other processors or devices see this memory.

Shared_id

This field contains a number that identifies this memory block. You must use the same number for each device when referring to it. RealView Debugger can then correctly update all device views when this memory is modified.

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 (see *Map_rule* on page A-13). 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.
Length	The block length of this memory unit. If Length is set by a register, this must be 0 or the amount to add to that register.
Type	The type of memory is dependent on the device type. The default is to map to data space. Otherwise, a memory space can be specified. Valid values are: • default • program • data • IO.
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.
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\Core\...\flash\examples* directory. Files are collected in subdirectories based on the target Flash device, for example *...\flash\examples\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.

For instructions on how to create your own FME file, see Chapter 4 *Programming Flash with RealView Debugger*. For more details on how using the FME file, see the chapter that describes reading and writing memory, registers, and Flash in the *RealView Debugger v1.8 User Guide*.

Description Description of memory space.

Volatile Enables you to define a memory block that is volatile on read (and so is marked specially in the Memory pane). 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.

Map_rule

These entries control the enabling and disabling of memory blocks using target registers. You specify a register to be watched, 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. The following settings are available:

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 (see *Register* on page A-14).

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* ($1 \ll 3$) 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:

<code>init_time</code>	Test when connecting to device.
<code>update_init</code>	Test on connect and when register changes.
<code>stop_update</code>	Test on connect, change, and execution stop.

Register_enum

Enumerations can be used, instead of values, when a register is displayed in the Register pane. This setting enables you to define the names associated with different values. Names defined in this group are displayed in the Register pane, and can be used to switch the register.

Register bit fields are numbered 0, 1, 2,... regardless of their position in the register.

The following setting is available:

Names You can specify a list of names, either in the form *name, name, name, ...* or in the form *name=value, name=value, name=value, ...*

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.

The Register group contains a Default group which contains a child group called Bit_fields which, in turn, contains the Default settings:

Position	Bit position from 0 (LSbit).
Size	Size in bits.
Signed	Set to True if signed, otherwise set to False.
Enum	Enumeration name to show values in the Register pane, derived from the Register_enum group.
Read_only	Set to True if read-only (cannot modify).

Volatile 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 Register pane. Right-click to see available options.

Gui_name Optional name for showing in Register pane.

Other settings in the Register entry, Default group, are:

Start The base address of the block. If the register is mapped using a memory block, this is the offset from that register (see Base).

Length The block length of this memory unit. If Length is set by a register, this must be 0 or the amount to add to that register.

Base This specifies how to interpret Start. If Base is Absolute, Start is the memory address of the register. Otherwise, Base can be set to the name of a memory block, and Start is an offset from the base address of that memory block.

Memory_type

The type of memory is dependent on the device type. The default is to map to data space. Otherwise, a memory space can be specified. Valid values are:

- default
- program
- data
- IO.

Type Specifies how to interpret the value contained in the register. The type names are as in the C language.

Read_only If set to True, the register is read-only and the debugger does not let you write to it. Otherwise, you can modify the value using the Register pane in the Code window and using CLI expressions.

Write_only If set to True, the register cannot be read. The debugger does not attempt to query the hardware for the current value when the Register pane is displayed.

Volatile If set to True, 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.

Enum The name of a Register_enum block that maps a register value to a textual string describing the value.

Gui_name The name of the register as it appears in the Register pane.

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 suggested 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.

The following settings are available:

Low_name Name of low register.

Low_shift Amount to shift low register by (<0 for left shift).

Low_mask Amount of low register to mask (after shift).

High_name Name of high register.

High_shift Amount to shift high register by (<0 for left shift).

High_mask Amount of high register to mask (after shift).

Length Length of register, in memory units.

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 to show values in the Register pane, derived from Register_enum group.

Gui_name Optional name for showing in Register pane.

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. The following settings are available:

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.

Register 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. See *Register* on page A-14 for details.

Start Start address of first peripheral register.

Length Block length.

Base Controls how the start field is interpreted. The default is Absolute (from 0), but can be relative to a memory block (if the block is disabled, the peripheral is too).

Type Basic type of the device. The available values are:

- serial
- parallel
- block
- network
- display
- other.

Description Description of the device.

Register_Window

This entry contains a set of lines to show in the Register pane. The name of the block is the tab name used for the lines. Each line contains a list of mapped registers displayed in the Register pane, see *Register_enum* on page A-14, *Register* on page A-14, and *Concat_Register* on page A-16 for more details.

The format of a line is *name,name,name,...* where each *name* is the name of a register or bit field. Be aware of the following:

- If the string starts with an equals sign, =, all the registers are shown as name=value in the window, else shown in table form (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 a expanded block
 - \$\$ this ends a previously opened block.

ARM_config

This group enables control of ARM processor settings used for ARM emulators, monitors, or simulators. These control features such as semihosting, vector catching, and memory control (for stack and heap assignment) which must be set or unset depending on the type of runtime you have linked into your application.

You can also set many of these at runtime using pseudo-registers. To do this, name the block Default if it applies to all devices or give it the name of the scan chain device to which it applies.

The following settings are available:

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.

Vectors	If <code>Vector_catch</code> is set to <code>True</code>, the fields within this group enable individual control over each vector. The following settings are available: <table> <tr> <td>Reset</td><td>Set this to catch Reset exceptions.</td></tr> <tr> <td>Undefined</td><td>Set this to catch undefined instruction exceptions.</td></tr> <tr> <td>SWI</td><td>Set this to catch Software Interrupts.</td></tr> <tr> <td>P_Abort</td><td>Set this to catch Prefetch abort exceptions.</td></tr> <tr> <td>D_Abort</td><td>Set this to catch Data abort exceptions.</td></tr> <tr> <td>Address</td><td>Set this to catch Address exceptions.</td></tr> <tr> <td>IRQ</td><td>Set this to catch normal interrupt exceptions.</td></tr> <tr> <td>FIQ</td><td>Set this to catch Fast Interrupt exceptions.</td></tr> <tr> <td>Error</td><td>Set this to catch errors.</td></tr> </table>	Reset	Set this to catch Reset exceptions.	Undefined	Set this to catch undefined instruction exceptions.	SWI	Set this to catch Software Interrupts.	P_Abort	Set this to catch Prefetch abort exceptions.	D_Abort	Set this to catch Data abort exceptions.	Address	Set this to catch Address exceptions.	IRQ	Set this to catch normal interrupt exceptions.	FIQ	Set this to catch Fast Interrupt exceptions.	Error	Set this to catch errors.
Reset	Set this to catch Reset exceptions.																		
Undefined	Set this to catch undefined instruction exceptions.																		
SWI	Set this to catch Software Interrupts.																		
P_Abort	Set this to catch Prefetch abort exceptions.																		
D_Abort	Set this to catch Data abort exceptions.																		
Address	Set this to catch Address exceptions.																		
IRQ	Set this to catch normal interrupt exceptions.																		
FIQ	Set this to catch Fast Interrupt exceptions.																		
Error	Set this to catch errors.																		
Semihosting	Enables programs to communicate with the host workstation. Semihosting operations supported include stack and heap assignment and console I/O (<code>printf</code> and <code>scanf</code> type calls). Semihosting is implemented using the SWI instruction. You can change the semihosting vector during debug using the <code>@semihost_vector</code> pseudo-register. On some targets you can also define a window or file number to display semihosting <code>printf</code> messages using <code>setreg @SEMIHOST_WINDOW=number</code>. The window numbers match the <code>VOPEN</code> command numbers, and the file numbers match the <code>FOPEN</code> command numbers. However, if your program displays prompts, it is suggested that you use a window. Contains: <table> <tr> <td>Enabled</td><td>Set this to enable semihosting.</td></tr> <tr> <td>Vector</td><td>Address of SWI vector catch to use.</td></tr> <tr> <td>Arm_swi_num</td><td>ARM SWI instruction for semihosting.</td></tr> <tr> <td>Thumb_swi_num</td><td>Thumb® SWI instruction for semihosting.</td></tr> </table>	Enabled	Set this to enable semihosting.	Vector	Address of SWI vector catch to use.	Arm_swi_num	ARM SWI instruction for semihosting.	Thumb_swi_num	Thumb® SWI instruction for semihosting.										
Enabled	Set this to enable semihosting.																		
Vector	Address of SWI vector catch to use.																		
Arm_swi_num	ARM SWI instruction for semihosting.																		
Thumb_swi_num	Thumb® SWI instruction for semihosting.																		
Armulator	Contains: <table> <tr> <td>Clock_speed</td><td>Clock speed in MHz as <code>num.num</code>.</td></tr> <tr> <td>Fpoint_emu</td><td>True if floating point emulation.</td></tr> <tr> <td>Config_file</td><td>The name of the configuration file.</td></tr> </table>	Clock_speed	Clock speed in MHz as <code>num.num</code> .	Fpoint_emu	True if floating point emulation.	Config_file	The name of the configuration file.												
Clock_speed	Clock speed in MHz as <code>num.num</code> .																		
Fpoint_emu	True if floating point emulation.																		
Config_file	The name of the configuration file.																		
Top_memory	Enables the semihosting mechanism to return the top of stack and base of heap. If not defined here, the default for each tool is used (different for RealView ICE, Multi-ICE®, or Angel). Any defined value is set into each																		

tool to force this address base. You can use explicit stack and heap sizes and locations below, but this might not be supported by all debug targets. You can also set this during a debugging session using the @top_mem pseudo-register.

Vector_catch

Used to catch possible program errors by setting breakpoints on (or otherwise trapping) the vectors. The default is to catch error-type vectors but leave IRQ, FIQ and SWI alone. SWI is caught separately by semihosting if enabled. To use this, the vectors must be writable. These can also be set during debugging using the @vector_catch pseudo-register. In this case, each bit, starting with 1, represents the vectors from reset to FIQ.

Properties This enables free-form definition of the properties required by a vehicle (emulator or simulator). The form of the string is *name=value*, where *name* is the name for the property as defined by the vehicle and *value* is a numeric value in hex or decimal.

Logic_Analyzer

This group is used to define settings for external trace analyzer hardware.

RealView Debugger currently supports:

- ARM ETM-based targets
- XScale™ onchip trace.

By default, RealView Debugger is automatically configured with tracing enabled for all targets using preset values.

If you have set up a new ARM ETM-based target using a new CONNECTION group (as described in *Creating new target descriptions* on page 3-20) you can configure these settings if you want the trace analyzer to be connected automatically when you connect to the target:

- right-click on Vendor and select **ARM**
- right-click on Load_when and select **connect**.

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...

The following settings are available in the Logic_Analyzer group:

Vendor Supported vendors.

Machine	Name of analyzer or device.
Config	Configuration file required to support your analyzer or device.
Load_when	Defines when RealView Debugger enables tracing: • enabled on connection, with Load_when set to connect • enabled on image load, with Load_when set to image_load • enabled when a specified symbol is matched, with Load_when set to symbol_match • enabled when first trigger is specified, with Load_when set to first_use.
Sym_match	Defines the symbol match to enable tracing.

For full details on tracing with RealView Debugger, see *RealView Debugger v1.8 Extensions User Guide*.

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, because of a break or other stop condition:

- input triggering means that the processor is stopped by others
- output triggering means that the processor can stop others.

The available settings are:

Trig_in_ena	List of commands to enable input triggering.
Trig_in_dis	List of commands to disable input triggering.
Trig_out_ena	List of commands to enable output triggering.
Trig_out_dis	List of commands to disable output triggering.

RTOS

This group enables automatic loading of RTOS and kernel awareness. This group enables you to configure an RTOS-enabled connection when you have no vendor-supplied .bcd file. When the RTOS is symbol-hooked, the RTOS plugin is only loaded when the RTOS, or its symbols, are loaded to the target. See the instructions from your vendor for proper setup. If supporting your own RTOS and kernel, use the method that best matches your DLL.

The following settings are available:

Vendor This three letter value identifies the RTOS plugin, that is the *.dll file supplied by your vendor.

Load_when Defines when RealView Debugger loads the RTOS plugin:

- load the plugin on connection, with Load_when set to connect
- wait until an RTOS image is loaded, with Load_when set to image_load.

The RTOS features of the debugger are not enabled until the plugin is loaded and has found the RTOS on your target.

Base_address

Defines a base address, overriding the default address used to locate the RTOS data structures. See your RTOS documentation for details.

Exit_Options

Defines how RTOS 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.

For full details on RTOS support with RealView Debugger, see *RealView Debugger v1.8 Extensions User Guide*.

Pre_connect

Forces an order for device connection. When you connect to a device within the .jtg file, this ensures that one or more specific devices are connected first no matter which you selected for connection. This enables pre-setup of the specific devices to guarantee

correct operation, such as initializations. You can specify the device(s) to connect to first by name, by processor name, or by processor type, as used in .cnf files that contain target configuration settings.

Commands

This enables you to specify RealView Debugger commands to run after a connection is established. The most common example is `INCLUDE` to include commands from a file. The commands are run immediately after the connection is completed. If `Pre_connect` is set, and the pre-connected device is running this command, the command executes before the original device is connected.

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` (see *Setting connect mode* on page 2-12 for details).

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, or in any .bcd file linked to the connection. If such a setting is found, it becomes the default connect mode for this connection.

Note

A blank entry in the top-level `Connect_mode` setting in the board file ensures that any setting in a linked Board/Chip definition file is used instead. This might be important if you are using .bcd files to enable RTOS awareness in RealView Debugger.

Use this setting to specify a connection mode. The options are:

<code>no_reset_and_stop</code>	Do not submit a reset and halt any process currently running.
<code>no_reset_and_no_stop</code>	Do not submit a reset or halt any process currently running.
<code>reset_and_stop</code>	Do a processor reset and halt any process currently running.
<code>reset_and_no_stop</code>	Do a processor reset but do not halt any process currently running.
<code>prompt</code>	Display a prompt for the connection mode to use.

The options available for the `Connect_mode` setting are generic to all vehicles and supported processors and so might include options that are not supported by your target vehicle.

If you set connect mode from the Connection Control window, using **Connect (Defining Mode)...**, this temporarily overrides any user-defined setting(s) in your target configuration file.

Note

If a prompt is specified in your board file, or in any .bcd file linked to the connection, 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.

Disconnect_mode

When you disconnect from a target, RealView Debugger attempts to disconnect using the default disconnect mode, that is *As-is with Debug* (see *Setting disconnect mode* on page 2-23 for details).

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, or in any .bcd file linked to the connection. If such a setting is found, it becomes the default disconnect mode for this connection.

Note

A blank entry in the top-level `Disconnect_mode` setting in the board file ensures that any setting in a linked Board/Chip definition file is used instead. This might be important if you are using .bcd files to enable RTOS awareness in RealView Debugger.

Use this setting to specify a disconnection mode. The options are:

- | | |
|----------------------------------|---|
| <code>as_is_with_debug</code> | Leave the target in its current state, whether stopped or running, and maintain any debugging controls. |
| <code>as_is_without_debug</code> | Leave the target in its current state, whether stopped or running, and remove any debugging controls. |
| <code>prompt</code> | Display a prompt for the disconnection mode to use. |

The options available for the `Disconnect_mode` setting are generic to all vehicles and supported processors and so might include options that are not supported by your target vehicle.

If you set disconnect mode from the Connection Control window, using **Disconnect (Defining Mode)...**, this temporarily overrides any user-defined setting(s) in your target configuration file.

Note

If a prompt is specified in your board file, or in any .bcd file linked to the connection, 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.

Id_chip

By default, the chip-id, or silicon-id, is loaded from the processor. When accessing special custom chips, it might be necessary 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.

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.

Chip_name

This defines the manufacturer name of the actual device, such as family name or core 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.

Endianness

This field applies to ARM cores. Use it to set the byte order of the simulated processor.

A.4.2 BOARD, CHIP, and COMPONENT groups

Use these groups to hold connection settings when a standard board or chip, exists. Use a BOARD group to define a target board from a commercial hardware vendor, for example the ARM Evaluator-7T, or a custom design. Similarly, use a CHIP group to define significant devices on a target board or where the device itself is complex. You can use a COMPONENT group to define a core 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 .jtg file, settings, and ETV information (ASIC, peripherals, and memory).

This type of entry can specify the default connection information which can then be overridden by settings in the CONNECTION group (see *CONNECTION groups* on page A-27 for details).

When you create a new group, the following entries are available, as described in *Generic groups and settings* on page A-5:

- Connect_with
- Advanced_Information
- Configuration
- Description
- Project
- Family_select
- BoardChip_name.

Note

If you create a new BOARD, CHIP, or COMPONENT entry, the Connect_with group also contains an Ethernet group of settings. These are not supported in this release.

A.5 Custom connection reference

To configure custom connections, you must set up one or more of the following groups in the board file:

- *CONNECTION groups*
- *DEVICE groups* on page A-28
- *RVBROKER groups* on page A-28
- *Working with JTAG configuration files* on page A-29
- *Working with ARM RDI configuration files* on page A-30.

A.5.1 CONNECTION groups

CONNECTION entries are used to get a list of one or more target devices. This setting specifies:

- the type of the device
- the position of the device in the scan chain
- a name used to specify what to connect to.

In some JTAG file forms, additional information such as speed adjust can also be specified. Using a CONNECTION group automatically pulls the list of devices from the named file and provides an easy way to keep the two locked together.

The following entries are available, as described in *Generic groups and settings* on page A-5:

- Connect_with
- Remote
- Advanced_Information
- Configuration
- Auto_connect
- Pre_connect
- Description
- Project
- Disabled
- Shared
- BoardChip_name
- Family_select.

A.5.2 DEVICE groups

Use a DEVICE entry when only one device exists on the scan chain or when you have to specify a lot of information for a specific device. The name of this group must be a name within the .jtg file.

The following entries are available, as described in *Generic groups and settings* on page A-5:

- Connect_with
- Remote
- Advanced_Information
- Description
- Project
- Configuration
- Disabled
- Shared
- Family_select
- BoardChip_name.

You can use a CONNECTION entry instead of a DEVICE entry.

A.5.3 RVBROKER groups

Use an RVBROKER entry to configure connections managed by RealView Connection Broker. This group enables you to access local host simulators or remote emulators, or simulators, running on another workstation on your network.

To access remote connections, debug target configuration files are held on the local workstation and are pushed across the network connection to the remote host.

Specific to this entry is the Simulator_info group that defines the simulator being used. The Default group contains:

Sim_family Specifies which member of the processor family to use.

Sim_dll Specifies the DLLs to load for custom ASICs or boards being simulated.

Endianess Used for ARM cores, this specifies the byte order of the processor being simulated.

The following entries are also available, as described in *Generic groups and settings* on page A-5:

- Remote
- Auto_connect

- Description
- Disabled.

Using a hosts file

The `rvbroker.brd` entry enables inclusion of a special hosts file containing a list of all network nodes that can run or are running RealView Connection Broker. This enables multiple users to access simulators, emulators, and EVMs anywhere on the network by sharing this single file.

This is the same as using RealView Connection Broker to manage remote connections across your network. However, the connection details are contained in the `rvbroker.brd` file and not in the board file. This also enables multiple users to share the configuration details from a central location on the network.

For full details on working with remote connections, see Chapter 6 *Working with Remote Targets*.

A.5.4 Working with JTAG configuration files

JTAG configuration files, `*.jtg`, define the devices on the JTAG scan chain and their order. This information might be supplied either by the manufacturer or configured after installation. RealView Debugger uses JTAG files to access emulator targets on the local host for each supported processor.

Groups use the Configuration setting to name a file defining the JTAG scan chain, for example CONNECTION or DEVICE (see *Generic groups and settings* on page A-5.) These files are expected to use the extension `.jtg`. There are some shortcuts you can use in defining these files:

- If you provide the name of a `.jtg` file without specifying a path, RealView Debugger searches for it, first in the current working directory, then in your home directory, and then in the default settings directory \etc.
- If you use the same name for the `.jtg` file as the name of the CONNECTION, and the Configuration entry is blank, RealView Debugger searches for a file called `connectionname.jtg`, first in the current working directory, then in your home directory, and then in the default settings directory \etc.
- If the Configuration entry is blank and a `.jtg` file cannot be found, RealView Debugger prompts you to complete the configuration details.

Based on the information contained in the `.jtg` file, RealView Debugger determines the appropriate scan length and access sequence for the processor you are communicating with.

You use the JTAG file editor, accessed using the Connection Properties window, to edit a .jtg file, and you use the Connection Properties window itself to supplement this information if required, for example to define how the connection is made to the board.

The RealView Debugger base product includes the following JTAG files:

arm.jtg	Specifies a single ARM processor on the scan chain.
arm_mp.jtg	Specifies two ARM processors on the scan chain
m56621.jtg	Specifies a Motorola M56621 DSP processor on the scan chain. You must have the RealView Debugger DSP license to access the Motorola M56621 processor.

———— **Note** ————

Where you have chosen a Custom installation, other .jtg files might be available to configure custom connections, for example for multiprocessor access such as ARM and DSP.

For details on working with JTAG files, see *Working with JTAG files* on page 5-36.

A.5.5 Working with ARM RDI configuration files

RDI configuration entries in the board file are generated by the RDI configuration utilities, for example (*.rbe) ARM RDI Configuration Entries.

You can expand this entry to see the second-level entries that list autodetected targets. Expand one of these, for example multiice.rbe, to see the CONNECTION entry, CONNECTION=Multi-ICE, where the configuration file, multiice.cnf, is specified.

For details on working with RDI files, see *Working with RDI targets* on page 5-14.

Glossary

The items in this glossary are listed in alphabetical order, with any symbols and numerics appearing at the end.

Access-provider connection

A debug target connection item that can connect to one or more target processors. The term is normally used when describing the RealView Debugger Connection Control window.

Address breakpoint

A type of breakpoint.

See also Breakpoint.

ADS

See ARM Developer Suite.

Angel

Angel is a software debug monitor that runs on the target and enables you to debug applications running on ARM-based hardware. Angel is commonly used where a JTAG emulator, such as Multi-ICE, is not available.

ARM Developer Suite (ADS)

A suite of software development applications, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of *RISC* processors. ADS is superseded by RealView Developer Suite (RVDS).

See also RealView Developer Suite.

ARM instruction	A word that encodes an operation for an ARM processor operating in ARM state. ARM instructions must be word-aligned.
ARM state	A processor that is executing ARM instructions is operating in ARM state. The processor switches to Thumb state (and to recognizing Thumb instructions) when directed to do so by a state-changing instruction such as BX, BLX. <i>See also</i> Thumb state.
Asynchronous execution	<i>Asynchronous execution</i> of a command means that the debugger accepts new commands as soon as this command has been started, enabling you to continue do other work with the debugger.
Backtracing	<i>See</i> Call Stack.
Big-endian	Memory organization where the least significant byte of a word is at the highest address and the most significant byte is at the lowest address in the word. <i>See also</i> Little-endian.
Board	RealView Debugger uses the term <i>board</i> to refer to a target processor, memory, peripherals, and debugger connection method.
Board file	The <i>board file</i> is the top-level configuration file, normally called rvdebug.brd, that references one or more other files.
Breakpoint	A user defined point at which execution stops in order that a debugger can examine the state of memory and registers. <i>See also</i> Conditional breakpoint, Default breakpoint, Hardware breakpoint, Software breakpoint, and Unconditional breakpoint.
Call Stack	This is a list of procedure or function call instances on the current program stack. It might also include information about call parameters and local variables for each instance.
Conditional breakpoint	A breakpoint that halts execution when a particular condition becomes True. The condition normally references the values of program variables that are in scope at the breakpoint location. <i>See also</i> Breakpoint, Default breakpoint, Hardware breakpoint, Software breakpoint, and Unconditional breakpoint.
Context menu	<i>See</i> Pop-up menu.

Core module	In the context of Integrator, an add-on development board that contains an ARM processor and local memory. Core modules can run stand-alone, or can be stacked onto Integrator motherboards. <i>See also</i> Integrator
Current Program Status Register (CPSR)	<i>See</i> Program Status Register.
DCC	<i>See</i> Debug Communications Channel.
Debug Communications Channel (DCC)	A debug communications channel enables data to be passed between RealView Debugger and the EmbeddedICE logic on the target using the JTAG interface, without stopping the program flow or entering debug state.
Debug With Arbitrary Record Format (DWARF)	ARM code generation tools generate debug information in DWARF2 format by default. From RVCT v2.2, you can optionally generate DWARF3 format (Draft Standard 9).
Default breakpoint	A <i>breakpoint</i> that does not have a condition qualifier or an action assigned. A default breakpoint is always an <i>unconditional breakpoint</i> . <i>See also</i> Breakpoint, Conditional breakpoint, Hardware breakpoint, Software breakpoint, and Unconditional breakpoint.
Deprecated	A deprecated option or feature is one that you are strongly discouraged from using. Deprecated options and features are to be removed in future versions of the product.
Digital Signal Processor (DSP)	DSPs are special processors designed to execute repetitive, maths-intensive algorithms. Embedded applications might use both ARM processor cores and DSPs.
Doubleword	A 64-bit unit of information.
DSP	<i>See</i> Digital Signal Processor.
DWARF	<i>See</i> Debug With Arbitrary Record Format.
ELF	Executable and Linking Format. ARM code generation tools produce objects and executable images in ELF format.
Embedded Trace Macrocell (ETM)	A block of logic, embedded in the hardware, that is connected to the address, data, and status signals of the processor. It broadcasts branch addresses, and data and status information in a compressed protocol through the trace port. It contains the resources used to trigger and filter the trace output.

EmbeddedICE logic	The EmbeddedICE logic is an on-chip logic block that provides TAP-based debug support for ARM processor cores. It is accessed through the TAP controller on the ARM core using the JTAG interface. <i>See also</i> IEEE1149.1.
Emulator	In the context of target connection hardware, an emulator provides an interface to the pins of a real core (emulating the pins to the external world) and enables you to control or manipulate signals on those pins.
Endpoint connection	A debug target processor, normally accessed through an <i>access-provider connection</i>.
ETM	<i>See</i> Embedded Trace Macrocell.
ETV	<i>See</i> Extended Target Visibility.
Extended Target Visibility (ETV)	Extended Target Visibility enables RealView Debugger to access features of the underlying target, such as chip-level details provided by the hardware manufacturer or SoC designer.
Floating Point Emulator (FPE)	Software that emulates the action of a hardware unit dedicated to performing arithmetic operations on floating-point values.
FPE	<i>See</i> Floating Point Emulator.
Halfword	A 16-bit unit of information.
Hardware breakpoint	A breakpoint that is implemented using non-intrusive additional hardware. Hardware breakpoints are the only method of halting execution when the location is in <i>Read Only Memory</i> (ROM). Using a hardware breakpoint often results in the processor halting completely. This is usually undesirable for a real-time system. <i>See also</i> Breakpoint, Conditional breakpoint, Default breakpoint, Software breakpoint, and Unconditional breakpoint.
IEEE Std. 1149.1	The IEEE Standard that defines TAP. Commonly (but incorrectly) referred to as JTAG. <i>See also</i> Test Access Port
Integrator	A range of ARM hardware development platforms. <i>Core modules</i> are available that contain the processor and local memory.

Joint Test Action Group (JTAG)

An IEEE group focussed on silicon chip testing methods. Many debug and programming tools use a *Joint Test Action Group* (JTAG) interface port to communicate with processors. For more information see IEEE Standard, Test Access Port and Boundary-Scan Architecture specification 1149.1 (JTAG).

JTAG

See Joint Test Action Group.

JTAG interface unit

A protocol converter that converts low-level commands from RealView Debugger into JTAG signals to the processor, for example to the EmbeddedICE logic and to the ETM.

Little-endian

Memory organization where the least significant byte of a word is at the lowest address and the most significant byte is at the highest address of the word.

See also Big-endian.

Multi-ICE

The ARM JTAG emulator debug tool for embedded systems. ARM registered trademark.

Pop-up menu

Also known as *Context menu*. A menu that is displayed temporarily, offering items relevant to your current situation. Obtainable in most RealView Debugger windows or panes by right-clicking with the mouse pointer inside the window. In some windows the pop-up menu can vary according to the line the mouse pointer is on and the tabbed page that is currently selected.

Processor core

The part of a microprocessor that reads instructions from memory and executes them, including the instruction fetch unit, arithmetic and logic unit and the register bank. It excludes optional coprocessors, caches, and the memory management unit.

Profiling

Accumulation of statistics during execution of a program being debugged, to measure performance or to determine critical areas of code.

Program Status Register (PSR)

Contains information about the current execution context. It is also referred to as the *Current PSR* (CPSR), to emphasize the distinction between it and the *Saved PSR* (SPSR), which records information about an alternate processor mode.

PSR

See Program Status Register.

RDI

See Remote Debug Interface.

RealMonitor

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

RealView ARMulator ISS (RVISS)

The most recent version of the ARM simulator, RealView ARMulator ISS is supplied with RealView Developer Suite. It communicates with a debug target using RV-msg, through the RealView Connection Broker interface, and RDI.

See also RDI and RealView Connection Broker.

RealView Compilation Tools (RVCT)

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

RealView Connection Broker

RealView Connection Broker is an execution vehicle that enables you to connect to simulator targets on your local system, or on a remote system. It also enables you to make multiple connections to the simulator.

See also RealView ARMulator ISS.

RealView Debugger Trace

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

RealView ICE (RVI) A JTAG-based debug solution to debug software running on ARM processors.

Remote Debug Interface (RDI)

The *Remote Debug Interface* (RDI) is an ARM standard procedural interface between a debugger and the debug agent. RDI gives the debugger a uniform way to communicate with:

- a simulator running on the host (for example, RVISS)
- a debug monitor running on hardware that is based on an ARM core accessed through a communication link (for example, Angel)
- a debug agent controlling an ARM processor through hardware debug support (for example, RealView ICE or Multi-ICE).

Remote_A

Remote_A is a software protocol converter and configuration interface. It converts between the RDI 1.5 software interface of a debugger and the Angel Debug Protocol used by Angel targets. It can communicate over a serial or Ethernet interface.

RTOS

Real Time Operating System.

RVCT

See RealView Compilation Tools.

RVISS

See RealView ARMulator ISS.

Scan chain	A scan chain is made up of serially-connected devices that implement boundary-scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain. Processors might contain several shift registers to enable you to access selected parts of the device.
Scope	The range within which it is valid to access such items as a variable or a function.
Script	A file specifying a sequence of debugger commands that you can submit to the command-line interface using the <code>include</code> command.
Semihosting	A mechanism whereby I/O requests made in the application code are communicated to the host system, rather than being executed on the target.
Simulator	A simulator executes non-native instructions in software (simulating a core).
Software breakpoint	A <i>breakpoint</i> that is implemented by replacing an instruction in memory with one that causes the processor to take exceptional action. Because instruction memory must be altered software breakpoints cannot be used where instructions are stored in read-only memory. Using software breakpoints can enable interrupt processing to continue during the breakpoint, making them more suitable for use in real-time systems. <i>See also</i> Breakpoint, Conditional breakpoint, Default breakpoint, Hardware breakpoint, and Unconditional breakpoint.
Software Interrupt (SWI)	An instruction that causes the processor to call a programmer-specified subroutine. Used by the ARM standard C library to handle semihosting.
SPSR	Saved Program Status Register. <i>See also</i> Program Status Register.
SWI	<i>See</i> Software Interrupt.
Synchronous execution	<i>Synchronous execution</i> of a command means that the debugger stops accepting new commands until this command is complete.
Synchronous starting	Setting several processors to a particular program location and state, and starting them together.
Synchronous stopping	Stopping several processors in such a way that they stop executing at the same instant.
TAP	<i>See</i> Test Access Port.

TAP Controller	Logic on a device which enables access to some or all of that device for test purposes. The circuit functionality is defined in Std. IEEE1149.1. <i>See also</i> Test Access Port and IEEE Std. 1149.1.
Target	The target board, including processor, memory, and peripherals, real or simulated, on which the target application is running.
Target vehicle	Target vehicles provide RealView Debugger with a standard interface to disparate targets so that the debugger can connect easily to new target types without having to make changes to the debugger core software.
Target Vehicle Server (TVS)	Essentially the debugger itself, this contains the basic debugging functionality. TVS contains the run control, base multitasking support, much of the command handling, and target knowledge, such as memory mapping, lists, rule processing, board file and .bcd files, and data structures to track the target environment.
Test Access Port (TAP)	The port used to access the TAP Controller for a given device. Comprises TCK , TMS , TDI , TDO , and nTRST (optional).
Thumb instruction	One halfword or two halfwords that encode an operation for an ARM processor operating in Thumb state. Thumb instructions must be halfword-aligned.
Thumb state	A processor that is executing Thumb instructions is operating in Thumb state. The processor switches to ARM state (and to recognizing ARM instructions) when directed to do so by a state-changing instruction such as BX, BLX. <i>See also</i> ARM state.
Tracepoint	A tracepoint can be a line of source code, a line of assembly code, or a memory address. In RealView Debugger, you can set a variety of tracepoints to determine exactly what program information is traced.
Trigger	In the context of breakpoints, a trigger is the action of noticing that the breakpoint has been reached by the target and that any associated conditions are met. In the context of tracing, a trigger is an event that instructs the debugger to stop collecting trace and display the trace information around the trigger position, without halting the processor. The exact information that is displayed depends on the position of the trigger within the buffer.
TVS	<i>See</i> Target Vehicle Server.

Unconditional breakpoint

A *breakpoint* that does not have a conditional qualifier assigned, and so always halts execution.

See also Breakpoint, Conditional breakpoint, Default breakpoint, Hardware breakpoint, and Software breakpoint.

Vector Floating Point (VFP)

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

VFP

See Vector Floating Point.

Watch

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

Watchpoint

In RealView Debugger, this is a hardware breakpoint.

Word

A 32-bit unit of information.

Index

The items in this index are listed in alphabetical order, with symbols and numerics appearing at the end. The references given are to page numbers.

A

- About this book viii
- Access configuration settings A-12
- Access-provider connections
 - in Connection Control window 2-3
 - menu 2-7, 2-9
- Access-provider disconnecting
 - menu 2-10
- Access-rule configuration settings
 - A-10
- Access_Method configuration settings
 - A-17
- Access_size configuration settings
 - A-10
- Address configuration settings A-19
- ADI 5-26
 - Agilent JTAG probe 5-26
 - configuration files 1-12
 - configuring devices 5-26
 - configuring JTAG Frequency 5-26
 - configuring targets 5-26
- ADI configuration dialog 5-26

- Advanced_Information block 1-3, 1-10,
 - A-8
 - Connect mode 2-14
 - Disconnect mode 2-25
- Advanced_Information configuration
 - settings A-6
- Agilent JTAG probe
 - ADI 5-26
- AME file
 - board-level 4-8
 - board-level file format 4-33
 - Flash-level 4-8
 - Flash-level file format 4-30
 - overview 4-8
- Angel debug monitor 2-3, 5-16, 5-22
 - see also* RDI, Remote_A
- Application_Load configuration settings
 - A-9
- ARM Agilent Debug Interface 5-26
- Armulator
 - configuration settings A-19
- ARMulator Configuration dialog 5-20,
 - 5-30

- ARM_config configuration settings A-18
- Arm_swi_num configuration settings
 - A-19
- ASCII MMethod file
 - see* AME file
- Attributes configuration settings A-10
- Audience, intended viii
- Auto-disconnect
 - in single processor mode 2-22
- Auto_connect configuration settings
 - A-6
- Aux_Configuration setting A-6

B

- Backup files
 - automatic 1-16
 - manual 1-16
 - restoring 1-17
- Backups
 - configuration files 3-21
 - files 1-16

- restoring 1-17
 - Base configuration settings A-15, A-17
 - Base_address configuration settings A-22
 - Base_mask configuration settings A-11
 - Base_scale configuration settings A-11
 - BCD files 1-3, 1-9
 - assigning to connection 3-11
 - creating for Flash programming 4-42
 - ETV 3-5
 - Flash programming 4-9
 - in search path 1-15
 - linking to connection 3-11
 - location 1-15
 - supplied in installation 3-7
 - BOARD
 - and CHIP entries 3-18
 - entries in board file 1-9, 3-9, 3-18
 - linking entries 3-12, 3-14, 3-16
 - settings reference A-25
 - Board file 1-2
 - about 1-2, 5-2
 - Advanced_Information block
 - configuration settings A-8
 - asterisk 1-7, 1-8, 5-6
 - BOARD groups A-25
 - CHIP groups A-25
 - COMPONENT groups A-25
 - configuration entries A-4, A-8
 - connection entries A-4, A-27
 - disabled entries 5-3
 - disabling entries 5-5
 - duplicate entries 5-4
 - enabled entries 5-3
 - enabling entries 5-4
 - entries 1-6, A-3
 - in the workspace 2-5
 - location 1-2, 3-3
 - merging hardware settings 3-14
 - ordering hardware 3-14, 3-15, 3-26
 - ordering hardware descriptions 3-14, 3-15, 3-26
 - restoring defaults 5-6
 - restoring entries 5-6
 - selecting 2-5
 - settings reference A-3
 - target configuration 3-3
 - BoardChip_name configuration settings A-7
 - Board/Chip Definitions
 - creating for Flash programming 4-42
 - creating new 3-21
 - Flash programming 4-9
 - linking 3-10
 - referring to 3-9
 - see also* BCD files
 - .bcd files 3-5
 - Book, about this viii
- ## C
- Channel viewers
 - in Remote_A 5-23
 - CHIP
 - and BOARD entries 3-18
 - entries in board file 1-9, 3-9, 3-18
 - settings reference A-25
 - Chip_name configuration settings A-25
 - Clock_speed configuration settings A-19
 - Command line arguments
 - home 1-14
 - install 1-14
 - user 1-15
 - Commands configuration settings A-23
 - Commands (CLI)
 - CONNECT 2-18
 - DISCONNECT 6-17
 - Comments
 - on documentation xii
 - on RealView Debugger xii
 - COMPONENT
 - entries in board file 1-9, 3-9
 - settings reference A-25
 - Concat_Register configuration settings A-16
 - Config configuration settings A-21
 - Configuration
 - backups 1-16
 - entries 1-9
 - entries in board file A-4, A-8
 - files 1-12
 - Configuration configuration settings A-6
 - Configuration example
 - connect mode 3-47
 - connect mode rules 3-46
 - custom register 3-51
 - disconnect mode 3-47
 - disconnect mode rules 3-46
 - Integrator 3-39
 - memory block 3-56
 - memory map 3-49
 - new RVISS connections 5-31
 - RealView ICE 5-8
 - RealView ICE with Integrator 3-39
 - setting connection mode 3-47
 - setting disconnection mode 3-47
 - setting memory block 3-56
 - setting memory map 3-49
 - setting registers 3-51
 - top_of_memory 3-59
 - Configuration settings
 - Access A-12
 - Access-rule A-10
 - Access_Method A-17
 - Access_size A-10
 - Address A-19
 - Advanced_Information A-6
 - Application_Load A-9
 - Armulator A-19
 - ARM_config A-18
 - Arm_swi_num A-19
 - Attributes A-10
 - Auto_connect A-6
 - Base A-15, A-17
 - Base_address A-22
 - Base_mask A-11
 - Base_scale A-11
 - BoardChip_name A-7
 - Chip_name A-25
 - Clock_speed A-19
 - Commands A-23
 - Concat_Register A-16
 - Config A-21
 - Configuration A-6
 - Config_file A-19
 - Connect_mode A-23
 - Connect_with A-5
 - Cross_trigger A-21
 - Description A-7, A-13
 - Disabled A-7
 - Disconnect_mode A-24

- D_Abort A-19
- Enabled A-19
- Endianness A-25, A-28
- Enum A-16, A-17
- Enum A-14, A-16, A-17
- Error A-19
- Exit_Options A-22
- Family_select A-7
- FIQ A-19
- Flash_type A-13
- Fpoint_emu A-19
- generic settings A-5
- Gui_name A-15, A-16
- High_mask A-16
- High_name A-16
- High_shift A-16
- Hostname A-6
- Id_chip A-25
- Id_match A-25
- Internal A-10
- IOdevice A-5
- IRQ A-19
- Length A-12, A-15, A-16, A-17
- Len_mask A-11
- Len_scale A-11
- Len_table A-12
- Load_command A-9
- Load_set_pc A-9
- Load_using A-9
- Load_when A-21, A-22
- Logic_Analyzer A-20
- Low_mask A-16
- Low_name A-16
- Low_shift A-16
- Machine A-21
- Manufacturer A-5
- Map_rule A-13
- Mask A-13
- Memory_block A-10
- Memory_type A-15
- Method_name A-17
- Names A-14
- On_equal A-13
- ordering 3-26
- Peripherals A-17
- Port A-6
- Position A-14
- precedence 3-26
- Pre_connect A-6, A-23
- priority 3-26
- Project A-7
- Properties A-20
- P_Abort A-19
- Read_only A-14, A-15
- RealView ARMulator ISS A-19
- reference A-3
- Register A-13, A-14, A-17
- Register_base A-11
- Register_enum A-14
- Register_length A-11
- Register_Pos_Len A-11
- Register_Window A-18
- Remote A-6
- Reset A-19
- RSD A-22
- RTOS A-21
- Semihosting A-19
- Shared A-7, A-10
- Shared_id A-10
- Signed A-14
- Sim_dll A-28
- Sim_family A-28
- Size A-14
- Speed A-5
- Start A-12, A-15, A-17
- SWI A-19
- Sym_match A-21
- System_Stop A-22
- Thumb_swi_num A-19
- Top_memory A-20
- Trig_in_dis A-21
- Trig_in_ena A-21
- Trig_out_dis A-21
- Trig_out_ena A-21
- Type A-12, A-15, A-16, A-17
- Undefined A-19
- Update_rule A-12, A-14
- Value A-13
- Vector A-19
- Vectors A-19
- Vector_catch A-20
- Vendor A-20, A-22
- Volatile A-10, A-13, A-15
- Wait_states A-12
- Write_only A-15
- Config_file configuration settings A-19
- CONNECT command 2-18
- Connect mode 2-12
 - configuration example 3-47
 - configuration rules 3-46
 - default 2-12
 - default state 2-16
 - defining 2-14
 - defining for a connection 2-14
 - in Advanced_Information block 2-14
 - No Reset and No Stop** 2-15
 - No Reset and Stop** 2-15
 - not honored 2-13, 3-46
 - prompting for a connection 2-16
 - prompting for a disconnection 2-28
 - Reset and No Stop** 2-15
 - Reset and Stop** 2-15
 - selection box 2-15
 - state options 2-15
 - substitution 2-13, 3-46
 - user-defined 2-13, 2-14
- Connecting 1-2, 1-3, 1-8, 2-4, 2-5
 - default files 2-5
 - to Multi-ICE 5-25
 - to multiple targets 2-17
 - to RealView ARMulator ISS 2-12
 - to remote targets 6-9
 - to targets 2-11
 - unconfigured targets 2-12
 - using names 2-17
 - using numbers 2-17
- CONNECTION
 - entries in board file A-27
- Connection 1-2
 - collapsed in Connection Control window 2-5
 - configuration 1-2
 - creating custom 1-9
 - DLLs 1-8
 - entries 1-8
 - entries in board file A-4, A-27
 - menu 2-12
 - viewing details 5-35
 - viewing remote connections 6-10
- Connection Broker 5-28
 - configuring RealView ARMulator ISS 2-8
 - local host connection 2-5, 5-32, 6-3
 - local mode 5-28
 - remote host connection 5-32, 6-3
 - Simulator Broker 5-28

- using a hosts file 6-11
- using DSP simulators 1-3
- Connection Control** button 2-2
- Connection Control window 1-3, 1-8, 2-2
 - Access-provider connections 2-3
 - adding RDI targets 5-15, 5-17
 - collapsing connections 2-5
 - collapsing entries 2-4
 - collapsing vehicles 2-4
 - configuring RDI targets 5-18
 - connect mode 2-12
 - connecting 2-5, 2-11
 - Connection menu 2-12
 - default connections 2-5
 - defining connect mode for a connection 2-14, 2-16
 - defining connect mode for a connection (default) 2-16
 - defining disconnect mode for a connection 2-25, 2-28
 - defining disconnect mode for a connection (default) 2-27
 - Description 2-3
 - disconnect mode 2-23
 - disconnecting 2-23
 - Endpoint connections 2-4
 - entries 2-3
 - expanding entries 2-4
 - expanding vehicles 2-4
 - failing to connect 2-19
 - JTAG target menu 2-8
 - managing connections 2-4, 2-6
 - Name 2-3
 - tabs 2-2
 - Target vehicles 2-3
 - user-defined connect mode 2-14
 - user-defined disconnect mode 2-25
 - vehicle 1-8
- Connection properties
 - entries 5-2
 - see also* Board file
- Connection Properties window 1-4, 1-6
 - Advanced_Information block A-8
 - asterisk 5-4
 - configuration settings reference A-3
 - connection entries A-27
- Connections
 - connect mode using the CLI 2-17
 - connecting using the CLI 2-17
 - failing 2-19
 - in multiprocessor mode 2-17
 - in the workspace 2-17
 - named using the CLI 2-17
 - numbered using the CLI 2-17
 - Connect_mode configuration settings A-23
 - Connect_with configuration settings A-5
 - Create New RDI Target dialog 5-17
 - Cross_trigger configuration settings A-21
 - Custom register
 - configuration example 3-51
- D**
- DCC semihosting 5-25
- Debug target
 - bcd configuration files 3-5
 - board file 1-2, 5-2
 - configuration 1-2, 3-2, 5-2
 - configuration files 3-6
 - configuring JTAG files 5-36
 - configuring Multi-ICE 2-11
 - configuring RealView ICE 2-11
 - connect defining mode 2-13, 2-14, 2-16
 - connect mode 2-12
 - connecting 1-2, 2-2, 2-4, 2-5, 2-11
 - connecting in single processor mode 2-11, 2-22
 - connection 1-2
 - creating configuration group 3-20
 - creating connection 3-20
 - creating new 3-20
 - creating new group 3-23
 - creating .bcd file 3-20
 - default configuration files 3-2
 - default connections 2-5
 - disconnect defining mode 2-24, 2-25, 2-28
 - disconnect mode 2-23
 - disconnecting 2-21, 6-17
 - disconnecting and reconnecting 2-29
 - examples configurations 3-38
 - extended visibility 1-3, 1-10
 - failing to connect 2-19
 - hardware 3-2
 - JTAG configuration files 3-4
 - linking target configuration to connection 3-20
 - RDI configuration files 3-4
 - RDI targets 5-14
 - RealView ARMulator ISS 5-19
 - RV-msg configuration files 3-3
 - troubleshooting configurations 3-62
 - troubleshooting connections 2-20, 3-62
 - user-defined connect mode 2-13, 2-14
 - user-defined disconnect mode 2-24, 2-25
- Default settings directory 1-15
- Description configuration settings A-7, A-13
- DEVICE
 - entries in board file A-28
- Device JTAG-File Editor dialog 5-36
- Dialog box
 - ADI configuration 5-26
 - ARMulator Configuration 5-20, 5-30
 - Create New RDI Target 5-17
 - Device JTAG-File Editor 5-36
 - Multi-ICE configuration 5-24
 - RDI Target List 5-15, 5-17
 - RealMonitor configuration 5-26
 - Remote_A connection 5-22
 - RVConfig 2-8, 5-11
- Directories
 - backups 1-16
 - default settings 1-15
 - home 1-14
 - installation 1-13
- Disabled configuration settings A-7
- DISCONNECT command 6-17
- Disconnect mode 2-23
 - As-is with Debug** 2-26
 - As-is without Debug** 2-26
 - configuration example 3-47
 - configuration rules 3-46
 - default 2-23, 2-24
 - default state 2-27

- defining 2-26
- defining for a disconnection 2-25
- in Advanced_Information block 2-25
- not honored 2-24, 2-27
- selection box 2-26
- state options 2-26
- substitution 2-24, 2-27
- user-defined 2-24, 2-25
- Disconnecting 6-17
 - auto-disconnect 2-22
 - by exiting 2-28
 - Code window behavior 2-21
 - confirming 2-28
 - from Connection Control window 2-23
 - in multiprocessor mode 2-21
 - project behavior 2-22
 - reconnecting 2-29
 - using the CLI 2-29
 - using the menu 2-22
- Disconnect_mode configuration settings A-24
- dispflash utility
 - checking the FME file 4-41
- Documentation feedback xii
- Domain name
 - configuring in remote targets 6-8
- DSP
 - connecting to hardware 5-13
 - connecting with RealView ICE 5-13
- D_Abort configuration settings A-19

E

- Enabled configuration settings A-19
- Endianess 5-24
- Endianess configuration settings A-25, A-28
- Endpoint connections
 - in Connection Control window 2-4
 - menu 2-8
- Enquiries xii
- Enum configuration settings A-16, A-17
- Enum configuration settings A-14, A-16, A-17
- Environment variables

- RVDEBUG_HOME 1-14
- RVDEBUG_INSTALL 1-14, 1-15
- RVDEBUG_SHARE 1-15
- Error configuration settings A-19
- ETV 1-3, 1-10, 3-2
- Evaluator-7T 3-7
- Exit_Options configuration settings A-22
- Extended Target Visibility 3-2
 - ETV 3-2

F

- Family_select configuration settings A-7
- Feedback
 - on documentation xii
 - on RealView Debugger xii
- File search path 1-15
- Files
 - board file 1-2
 - configuration 1-12
 - JTAG 5-36
 - rvbroker.brd 6-4, 6-11
 - rvdebug.brd 1-12
 - using a hosts file 6-11
 - *.auc 5-32
 - *.bcd 1-3, 1-9, 1-12
 - *.bcd supplied 3-7
 - *.brd 1-12
 - *.cnf 1-12
 - *.jtg 5-36
 - *.prc 1-13
 - *.rbe 1-13
 - *.rvc 1-13
 - .jtg 3-4
 - .rvc 3-3
- FIQ configuration settings A-19
- Flash
 - checking contents of 4-47
- Flash MEmory file
 - see FME file
- Flash programming
 - see also AME file
 - ARM boards 4-11
 - assembler settings 4-38
 - board-level code 4-8
 - Board/Chip Definition file 4-9, 4-42

- building the project 4-40
- compiler settings 4-38
- creating algorithms for custom Flash types 4-24
- creating algorithms for support Flash types 4-20
- custom platforms 4-16
- custom platforms with custom Flash type 4-17
- custom platforms with supported Flash type 4-16
- example 3-62
- files used for ARM platforms 4-3
- files used for custom platforms and custom Flash 4-6
- files used for custom platforms and supported Flash 4-5
- Flash-level code 4-7
- see also FME file
- gathering target information 4-19
- generating the FME file 4-36
- linker settings 4-39
- running the pakflash utility 4-39
- troubleshooting 4-48
- Flash_type configuration settings A-13
- FME file
 - checking with dispflash 4-41
 - generating 4-36
 - overview 4-9
 - pakflash utility 4-10
 - specifying in BCD file 4-44
- Format, of registers 3-7
- Fpoint_emu configuration settings A-19

G

- Glossary Glossary-1
- Group Name/Type selector dialog 5-8, 5-32, 5-33
- Gui_name configuration settings A-15, A-16

H

- High_mask configuration settings A-16
- High_name configuration settings A-16
- High_shift configuration settings A-16

Home
 see Directories
 Hostname
 configuring in remote targets 6-7
 Hostname configuration settings A-6
 Hosts
 accessing remote targets 6-11
 Hosts file 6-11
 configuring local 6-12
 configuring network 6-15
 connecting local 6-14
 contents 6-13, 6-14
 creating 6-13
 creating local 6-11
 editing 6-13
 in board file 6-12
 in Connection Control window
 6-12, 6-14
 multiple targets 6-14
 network 6-15

I

Id_chip configuration settings A-25
 Id_match configuration settings A-25
 Integrator
 configuration example 3-39
 Integrator/AP 3-7, 3-9
 Integrator/CM 3-9
 Integrator/CP 3-8, 3-9
 Intended audience viii
 Internal configuration settings A-10
 IODEVICE configuration settings A-5
 IP address
 configuring in remote targets 6-8
 IRQ configuration settings A-19

J

JTAG
 configuration files 3-4
 JTAG files
 adding new devices 5-38
 configuring Bypass 5-37
 configuring connect options 5-37
 configuring device name 5-37
 configuring devices 5-38

 configuring extra entries 5-38
 configuring processor name 5-37
 copying devices 5-38
 Edit Configuration-File Contents
 5-36
 editing 5-36
 in the Connection Control window
 5-36
 location 3-4
 target configuration 5-36
 TDI 5-38
 TDO 5-38
 Test Data In 5-38
 Test Data Out 5-38
 testing 2-8
 JTAG targets
 JTAG target menu 2-8

L

Length configuration settings A-12,
 A-15, A-16, A-17
 Len_mask configuration settings A-11
 Len_scale configuration settings A-11
 Len_table configuration settings A-12
 Licenses, valid 2-19
 Linking
 a board to a connection 3-12
 avoiding conflicts when 3-11
 boards to a connection 3-14
 boards to boards 3-16
 Load_command configuration settings
 A-9
 Load_set_pc configuration settings A-9
 Load_using configuration settings A-9
 Load_when configuration settings A-21,
 A-22
 Logic_Analyzer configuration settings
 A-20
 Low_mask configuration settings A-16
 Low_name configuration settings A-16
 Low_shift configuration settings A-16

M

Machine configuration settings A-21

Manufacturer configuration settings
 A-5
 Map_rule configuration settings A-13
 Mask configuration settings A-13
 Memory
 setting configuration example 3-56
 Memory block
 configuration example 3-56
 Memory map
 configuration example 3-49
 Memory_block configuration settings
 A-10
 Memory_type configuration settings
 A-15
 Method_name configuration settings
 A-17
 Multi-ICE
 accessing remote targets 6-2
 and RMHost 3-34
 compatibility with RealView
 Debugger 5-25
 configuration 5-24
 configuration dialog 5-24
 configuration files 1-12
 configuring cache clean 5-24
 configuring connection 5-24
 configuring endianness 5-24
 configuring for RealMonitor 3-34
 configuring in multiprocessor mode
 5-24
 configuring processor settings 5-24
 configuring RDI targets 5-23
 configuring targets 5-23
 configuring the target 2-5, 2-9, 2-20
 configuring trace 5-24
 connections 5-25
 DCC semihosting 5-25
 DLL version numbers 5-24
 setting per-processor options 5-24
 troubleshooting connections 2-20

N

Names configuration settings A-14
 Network hosts file 6-15
 configuring 6-15
 contents 6-15
 creating 6-15

- editing 6-15
 - in board file 6-15
 - in Connection Control window 6-16
- O**
- On_equal configuration settings A-13
- P**
- pakflash utility
 - command 4-10
- Peripherals configuration settings A-17
- Plugins
 - for RealMonitor 3-5
 - for RTOS 3-5
- Port configuration settings A-6
- Position configuration settings A-14
- Pre_connect configuration settings A-6, A-23
- Problem solving xii
- Product feedback xii
- Project configuration settings A-7
- Projects
 - on disconnecting 2-22
- Properties configuration settings A-20
- P_Abort configuration settings A-19
- Q**
- Queries xii
- R**
- RDI
 - adding devices 2-6, 2-8, 5-15, 5-17
 - adding DLLs 5-15
 - adding targets 2-6, 2-8, 5-15, 5-17
 - adding third-party 2-6, 2-8, 5-15, 5-17
 - configuration files 3-4
 - configuring targets 5-14, 5-18
 - creating new target 5-17
 - *.cnf files 3-4
 - *.rbe files 3-4
 - *.cnf files 3-4
 - RDI Target List dialog 2-6, 2-8, 5-15, 5-17
 - adding DLLs 5-16
 - enabling RDI connections 5-16
 - RDI 1.5.1 5-15, 5-17
 - Read_only configuration settings A-14, A-15
 - RealMonitor
 - Board/Chip definition files 3-5
 - configuration 5-26
 - configuration dialog 3-35, 5-26
 - configuration files 1-12
 - configuring JTAG Controller 5-26
 - configuring RDI Module Server 5-26
 - configuring the target 2-5, 2-9, 2-20
 - connecting RealView ICE to a running target 3-28
 - Multi-ICE and RMHost 3-34
 - plugins 3-5
 - RealView ICE restrictions 3-28
 - RMHost 3-29, 3-34
 - using with Multi-ICE 3-34
 - using with RealView ICE 3-28
 - working with 3-37
 - RealView ARMulator ISS 5-19, 5-28, 5-29, 5-31
 - configuration settings A-19
 - configuring 2-8, 5-19
 - configuring clock 5-20, 5-30
 - configuring connections 5-29
 - configuring devices 5-29
 - configuring endianness 5-20, 5-21, 5-30, 5-31
 - configuring FPA 5-20
 - configuring map files 5-21
 - configuring MMU 5-21, 5-31
 - configuring new connection 5-31
 - configuring pagetables 5-21, 5-31
 - configuring processor 5-20, 5-30
 - configuring targets 5-20
 - configuring VFP 5-21, 5-31
 - Simulator Broker connections 5-28
 - RealView Debugger
 - compatibility with Multi-ICE 5-25
 - connecting to a target 1-2
 - target configuration 1-2
 - RealView ICE
 - accessing remote targets 6-2
 - adding targets 5-7
 - configuration files 1-12, 1-17
 - configuring devices 5-10
 - configuring for RealMonitor 3-28
 - configuring interface unit 5-10
 - configuring new connection 5-8
 - configuring the target 2-5, 2-9, 2-20
 - Connect mode 2-13, 3-46
 - Disconnect mode 2-13, 3-46
 - in the Connection Control window 5-7
 - Integrator configuration example 3-39
 - limitations 3-39
 - RVConfig 2-8, 5-10
 - troubleshooting connections 2-20
 - viewing targets 5-7
 - RealView Network Broker 6-3
 - closing down 6-10
 - see also Connection Broker
 - Register configuration settings A-13, A-14, A-17
 - Register format 3-7
 - Registers
 - setting configuration example 3-51
 - Register_base configuration settings A-11
 - Register_enum configuration settings A-14
 - Register_length configuration settings A-11
 - Register_Pos_Len configuration settings A-11
 - Register_Window configuration settings A-18
 - Remote configuration settings A-6
 - Remote targets
 - accessing 6-2
 - accessing Multi-ICE 6-2
 - accessing RealView ICE 6-2
 - accessing simulators 6-3
 - configuring domain name 6-8
 - configuring hostname 6-7
 - configuring IP address 6-8
 - configuring new simulator 6-6
 - connecting 6-9
 - RVBROKER 6-4
 - rvbroker.brd 6-4

- simulators 6-6
- viewing connections 6-10
- Remote_A
 - channel viewers 5-23
 - connection driver 5-22
 - endianess 5-23
 - heartbeat 5-23
 - target configuration 5-22
- Remote_A connection dialog 5-22
- Reset configuration settings A-19
- RMHost 3-29, 3-34
- RSD configuration settings A-22
- RTOS
 - Board/Chip definition files 3-5
 - plugins 3-5
- RTOS configuration settings A-21
- RVBROKER
 - entries in board file 6-4, A-28
- rvbroker.brd
 - entries in board file 6-4
 - in search path 6-11
- RVConfig
 - Advanced settings 5-11
 - and .rvc files 1-13
 - AutoConfigure Scan Chain** 5-11
 - with RealView ICE 5-10
- RVConfig dialog 2-8, 5-11
- RVDEBUG_HOME 1-14
- RVDEBUG_INSTALL 1-14, 1-15
- RVDEBUG_SHARE 1-15
- RVISS 5-19
- RV-msg
 - configuration files 1-12, 3-3

S

- Search path, file 1-15
- Semihosting configuration settings A-19
- Shared configuration settings A-7, A-10
- Shared_id configuration settings A-10
- Signed configuration settings A-14
- Simulator Broker
 - Connection Broker 5-28
 - connections to RealView ARMulator ISS 5-28
 - connections to simulators 5-28

- Simulators 6-6
 - accessing 6-3
 - configuring new 6-6
- Sim_dll configuration settings A-28
- Sim_family configuration settings A-28
- Size configuration settings A-14
- Speed configuration settings A-5
- Stack heap
 - relationship to top of memory 3-60
- Start configuration settings A-12, A-15, A-17
- Structure of this book viii
- SWI configuration settings A-19
- Sym_match configuration settings A-21
- Synch** tab
 - Connection Control window 2-2
- System_Stop configuration settings A-22

T

- Target
 - see* Debug target
- Target vehicle
 - in Connection Control window 2-3
- Terminology Glossary-1
- Thumb_swi_num configuration settings A-19
- Top_memory configuration settings A-20
- top_of_memory
 - configuration example 3-59
 - setting 3-61
- Trig_in_dis configuration settings A-21
- Trig_in_ena configuration settings A-21
- Trig_out_dis configuration settings A-21
- Trig_out_ena configuration settings A-21
- Troubleshooting
 - configurations 3-62
 - connections 2-20
 - Flash programming 4-48
 - Multi-ICE connections 2-20
 - RealView ICE connections 2-20
- Type configuration settings A-12, A-15, A-16, A-17

U

- Undefined configuration settings A-19
- Update_rule configuration settings A-12, A-14

V

- Valid licenses 2-19
- Value configuration settings A-13
- Vector configuration settings A-19
- Vectors configuration settings A-19
- Vector_catch configuration settings A-20
- Vehicle 1-8
 - connecting to RealView ARMulator ISS 1-8
 - connecting to RealView ICE 1-8
 - menu 2-6
- Vendor configuration settings A-20, A-22
- Volatile configuration settings A-10, A-13, A-15

W

- Wait_states configuration settings A-12
- Windows
 - Connection Control 1-3, 1-8, 2-2
 - Connection Properties 1-4, 1-5
- Write_only configuration settings A-15